ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

«Ανάπτυξη τρισδιάστατου παιχνιδιού για κινητές πλατφόρμες με αυτόνομους διαδραστικούς χαρακτήρες»

Διπλωματική Εργασία

Βερυκοκίδη Ελένη Αγγέλα

2014

Εξεταστική Επιτροπή:

Αν.Καθηγήτρια Αικατερίνη Μανιά (Επιβλέπουσα)
Αν.Καθηγητής Μιχαήλ Γ. Λαγουδάκης
Αν.Καθηγητής Αντώνιος Δεληγιαννάκης

Χανιά

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω την κυρία Μανιά που μου έδωσε την ευκαιρία να υλοποιήσω στη διπλωματική μου εργασία το παιχνίδι που ήθελα, και να ασχοληθώ με τους τομείς που με ενδιαφέρουν, για το χρόνο που μου αφιέρωσε, και για την καθοδήγησή της ώστε να υλοποιηθεί η διπλωματική εργασία!

Επίσης, Θα ήθελα να ευχαριστήσω τον κύριο Λαγουδάκη και τον κύριο Δεληγιαννάκη για τη βοήθειά τους, τις συμβουλές τους, για το χρόνο τους!

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου και τα κοντινά μου πρόσωπα για την υποστήριξη, το κουράγιο και την πολύτιμη βοήθεια που μου πρόσφεραν, το χρόνο που αφιερωσαν χρόνο στο παιχνίδι και τη συνεισφορά τους στην βελτίωσή του με τις συμβουλές και τις προτάσεις τους!

ΠΕΡΙΛΗΨΗ

Η διπλωματική εργασία αφορά την υλοποίηση ενός τρισδιάστατου παιχνιδιού για κινητές συσκευές αφής, λειτουργικού συστήματος Android ή iOS, με την πλατφόρμα Unity 3D. Μέσα από το παιχνίδι εκμεταλλευτήκαμε τις παραπάνω δυνατότητες που μας προσφέρουν οι συσκευές αυτές σε αντίθεση με τους υπολογιστές, όπως είναι το επιταχυνσιόμετρο, το γυροσκόπιο και οι ασθητήρες αφής, τα οποία χρησιμοποιήθηκαν για τη διαχείριση του παιχνιδιού και την κίνηση του ήρωα.

Μεγάλη έμφαση δόθηκε στο γραφικό και σχεδιαστικό κομμάτι του παιχνιδιού, καθώς ο ήρωας και σχεδόν κάθε αντικείμενο που χρησιμοποιείται στο παιχνίδι σχεδιάστηκε εξολοκλήρου στα πλαίσια της διπλωματικής εργασίας.

Το παιχνίδι είναι κατάλληλα σχεδιασμένο ώστε να είναι εύχρηστο και φιλικό ως προς τον χρήστη λαμβάνοντας υπόψιν ότι θα έχει μπροστά του μια μικρή οθόνη και ο τρόπος χειρισμού του παιχνιδιού περιορίζεται μόνο στην αφή και την κλίση της συσκευής.

Στα πλαίσια αυτά προσπαθήσαμε να το κάνουμε όσο πιο ενδιαφέρον γίνεται χρησιμοποιώντας μια ποικιλία στους τρόπους χειρισμού του ήρωα, στα είδη και τον αριθμό των αντιπάλων, στην δυσκολία της κάθε πίστας, στο περιβάλλον του κάθε κόσμου, στη μουσική υπόκρουση, στις δυνάμεις του ήρωα, στο συνολικό σκορ που διαμορφώνει σε κάθε πίστα, στις ζωές που έχει και κυρίως στην αυτονομία των χαρακτήρων στους οποίους προστέθηκε τεχνητή νοημοσύνη για να αλληλεπιδρούν με τον ήρωα.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

1.1 Εσαγωγή

Τα τελευταία χρόνια η χρήση κινητών τηλεφώνων smartphone ή iPhone, καθώς και των υπολογιστών χεριού tablet ή iPad, έχει αυξηθεί σημαντικά παγκοσμίως και συνεχίζει να επεκτείνεται με ταχύ ρυθμό. Σύμφωνα με έρευνες οι χρήστες smartphone σε παγκόσμιο επίπεδο ξεπεράσαν το 1 δισεκατομμύριο το 2012 και αναμένεται πως το 2014 θα ανέλθουν στο 1,75 δισεκατομμύριο.

Το 2013 από τα λειτουργικά συστήματα των smartphones το Android έχει το μεγαλύτερο μερίδιο αγοράς με ποσοστό που ανέρχεται στο 79% διαθέτοντας περισσότερες από 1.000.000 εφαρμογές, ενώ το iOS έχει το 14.2% διαθέτοντας περισσότερες από 1.000.000 εφαρμογές. Καταλαβαίνουμε λοιπόν ότι ο τομέας των εφαρμογών έχει πολλά περιθώρια ανάπτυξης και εξέλιξης καθώς το ενδιαφέρον του κόσμου είναι μεγάλο προς τον συγκεκριμένο τομέα της τεχνολογίας.

Η απασχόληση των χρηστών με τα παιχνιδια καταλαμβάνει το 66% του χρόνου της καθημερινής χρήσης των Smartphones . Κατά μέσο όρο μηνιαίως ένας χρήστης iPhone κατεβάζει το μέγιστο 48 εφαρμογές από τις οποίες το 26% είναι παιχνίδια, ενώ ένας χρήστης Android κατεβάζει 35 εφαρμογές από τις οποίες το 20% είναι παιχνίδια. Όπως φαίνεται η κατηγορία των παιχνιδιών έχει μεγάλη ζήτηση και καταλαμβάνει σημαντικό κομμάτι στη χρήση των smartphones και στην δραστηριότητα των χρηστών, για αυτό και απασχολεί πολλούς προγραμματιστές και αυτός είναι ένας βασικός λόγος που κινηθήκαμε σε αυτόν τον τομέα όσον αφορά τη διπλωματική εργασία.

Η διπλωματική εργασία πραγματεύεται τη δημιουργία ενός τρισδιάστατου παιχνιδιού, κατηγορίας Arcade, Action και Adventure, για κινητές συσκευές με λειτουργικό σύστημα Android ή iOS (smartphones, iPhone, tablet computer και iPad). Για αυτό το λόγο είναι κατάλληλα σχεδιασμένο ώστε να είναι εύχρηστο και ευχάριστο σε μικρές οθόνες διαφορετικού μεγέθους, διαφορετικής ανάλυσης, διαφορετικού επεξεργαστή της συσκευής και διαφορετικού λογισμικού, καθώς η αγορά ποικίλει στα διάφορα είδη κινητών συσκευών.

Το παιχνίδι υλοποιήθηκε στην πλατφόρμα Unity3D η οποία υποστηρίζει και τα δύο λογισμικά Android και iOS , και ως γλώσσα προγραμματισμού χρησιμοποιήθηκε η JavaScript. Για την μετατροπή του παιχνιδιού σε εφαρμογή Android χρησιμοποιήθηκε το Android SDK Manager.

Ο ήρωας σχεδιάστηκε εξολοκλήρου με το πρόγραμμα Maya της Autodesk. Ακολουθήθηκαν όλες οι διαδικασίες μοντελοποίησης, οι οποίες περιλαμβάνουν την κατασκευή του τρισδιάστατου μοντέλου (3D Modeling), την ύφανση (Texturing), τη δυνατότητα της κίνησης (Rigging), και τις κινήσεις (Animation) που βλέπουμε στο παιχνίδι.

1.2 Περιγραφή Παιχνιδιού

Ο ήρωας μας είναι ένας πιγκουίνος ο οποίος περιπλανιέται στα χιόνια και προσπαθεί να βρει το θησαυρό σε κάθε πίστα. Στο δρόμο του μαζεύει τα νομίσματα που πέσανε από το σεντούκι όταν προσπάθησαν να τον κρύψουν οι κακοί βίκινγκς! Πρέπει όμως να τους αποφεύγει ή να τους εξουδετερώνει και αυτούς και τα κανόνια τους γιατί θα τον σκοτώσουν.

Σκοπός είναι η όσο το δυνατόν καλύτερη εκμετάλλευση των δυνατοτήτων των κινητών συσκευών μέσω της διαχείρησης του ήρωα από τον χρήστη. Για να υπάρξει ποικιλία δημιουργήθηκαν τέσσερις κόσμοι παιχνιδιού ανάλογα με τον τρόπο κίνησης του ήρωα ο οποίος είναι διαφορετικός στον κάθε κόσμο. Ο κάθε κόσμος αποτελείται από οχτώ διαφορετικές πίστες κλιμακώμενης δυσκολίας.

Στον πρώτο και δεύτερο κόσμο για την κίνηση και περιστροφή του ήρωα χρησιμοποιείται ένα χειριστήριο αφής, το οποίο κινείται ελάχιστα πάνω στην οθόνη σύμφωνα με την κατεύθυνση του δαχτύλου του χρήστη. Στον τρίτο κόσμο για την κίνηση χρησιμοποιείται το επιταχυνσιόμετρο όπου αναλόγως την κλίση που δίνει ο χρήστης στο κινητό κινείται ο ήρωας δεξιά και αριστερά, μέσω ενός φίλτρου που δημιουργήσαμε για να γίνεται ομαλή η κίνηση. Στον τέταρτο κόσμο μπορεί να επιλέξει ο χρήστης αν θα χρησιμοποιήσει για την κίνηση το επιταχυνσιόμετρο ή τα κουμπιά αφής.

Το γυροσκόπιο χρησιμοποιείται για την περιστροφή της οθόνης ανάλογα με την περιστροφή της συσκευής. Χρησιμοποιούνται και οι αισθητήρες αφής με τους οποίους αναγνωρίζονται συγκεκριμένες περιοχές της οθόνης για την υλοποίηση κουμπιών, υπολογίζεται ο αριθμός των δαχτύλων που βρίσκονται στην οθόνη την κάθε στιγμή, και μετράμε τις φορές που ο χρήστης ακουμπάει την οθόνη. Επίσης υπολογίζεται η κατεύθυνση της κίνησης του δαχτύλου του χρήστη πάνω στην οθόνη για την κίνηση του ήρωα στον τέταρτο κόσμο, αν σύρει το δάχτυλο προς τα πάνω ο ήρωας πηδάει, αν το σύρει προς τα κάτω κυλιέται με τούμπα για να αποφύγει τα εμπόδια, αν το σύρει δεξιά ή αριστερά στρίβει 90° αντίστοιχα.

Έχει προστεθεί τεχνητή νοημοσύνη σε δύο χαρακτήρες σε διαφορετικούς κόσμους, στον βίκινγ και στη φάλαινα, ώστε να μπορούν να βρουν τη συντομότερη διαδρόμη ως προς το στόχο τους αποφεύγοντας τα εμπόδια στο δρόμο τους και χωρίς να βγαίνουν έξω από το επιτρεπτό μονοπάτι.

Στη μία περίπτωση η φάλαινα δείχνει στον πιγκουίνο το δρόμο για το θησαυρό από τη πιο σύντομη διαδρομή μέσα από το λαβύρινθο. Αν απομακρυνθεί ο πιγκουίνος από αυτήν, τον περιμένει να γυρίσει για να συνεχίσει το δρόμο της. Στην άλλη περίπτωση ο βίκινγκ κυνηγάει τον πιγκουίνο μέσα στον λαβύρινθο, ώστε να τον πλησιάζει από την συντομότερη διαδρομή ανεξαρτήτως από τη διαδρομή που διάλεξε ο χρήστης.

1.3 Πλατφόρμα

Το παιχνίδι είναι σχεδιασμένο για κινητές συσκευές αφής με λειτουργικό σύστημα Android και iOS, όπως είναι τα smartphone, iPhone, τα tablet pc και iPad.

Υλοποιήθηκε στην πλατφόρμα Unity 3D και ο κώδικας είναι γραμμένος σε γλώσσα JavaScript. Το Unity ως πολυπλατφόρμα υποστηρίζει το παχνίδι για όλα τα λειτουργικά συστήματα με κατάλληλες μετατροπές φυσικά. Ο κώδικας είναι γραμμένος έτσι ώστε να αναγνωρίζεται και από Android και από iOS.

Η Unity υποστηρίζει την ανάπτυξη σε πολλαπλές πλατφόρμες. Μέσα από το ίδιο project μπορούμε να εκδόσουμε ένα παιχνίδι σε κινητές συσκευές, σε προγράμματα

περιήγησης στο διαδίκτυο, σε υπολογιστές και σε κονσόλες. Επίσης συμπιέζει τις υφές και ρυθμίζει την ανάλυση για κάθε πλατφόρμα που υποστηρίζει το παιχνίδι.

Οι υποστηριζόμενες πλατφόρμες περιλαμβάνουν τα Xbox One, BlackBerry 10, Windows 8, Windows Phone 8, Windows, Mac, Linux, Android, iOS, Unity Web Player, Adobe Flash, PlayStation 3, Xbox 360, Wii U και Wii. Επίσης υποστηρίζει ανεπίσημα και το PlayStation Vita. Προσεχώς πλατφόρμες περιλαμβάνουν το PlayStation 4.

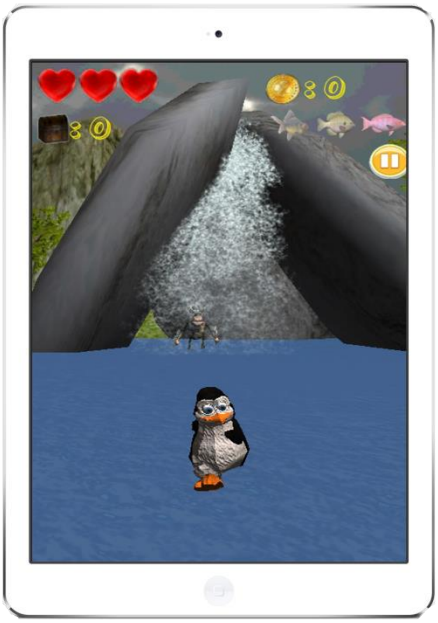
1.3.1 Απαιτήσεις συστήματος για Android ανάπτυξη

Εκτός από τις γενικές απαιτήσεις συστήματος, χρειαζόμαστε το Android SDK και τη Java Development Kit (JDK). Οι συσκευές πρέπει να έχουν από την έκδοση Android OS 2.3.1 και πάνω για να λειτουργήσει σωστά. Η συσκευή πρέπει να τροφοδοτείται από μια ARMv7 (Cortex family) CPU και η GPU συνιστάται να υποστηρίζει OpenGLES 2.0. Έχοντας κάνει τα παραπάνω και επιλέγοντας την έκδοση της Unity για Android, μετατρέπεται το παιχνίδι σε εφαρμογή Android με τη μορφή αρχείου .apk .

1.3.2 Απαιτήσεις συστήματος για iOS ανάπτυξη

Εκτός από τις γενικές απαιτήσεις συστήματος, μόνο η Unity OS X έκδοση υποστηρίζει το iOS ως πλατφόρμα κατασκευής, που σημαίνει ότι χρειαζόμαστε ένα υπολογιστή MAC. Επίσης χρειαζόμαστε το πρόγραμμα Xcode 4.3 το οποίο διατίθεται μόνο επί πληρωμής. Έχοντας κάνει τα παραπάνω και επιλέγοντας την έκδοση της Unity για iOS, μετατρέπεται το παιχνίδι σε εφαρμογή iOS με τη μορφή αρχείου .ipa .

Το ίδιο παιχνίδι που υλοποιήσαμε για κινητές συσκευές, είναι διαθέσιμο και για υπολογιστές, PC και MAC , με μικρές μετατροπές στον κώδικα όσον αφορά το χειρισμό καθώς χειριζόμαστε τον ήρωα από το πληκτρολόγιο και το μενού από το ποντίκι.



1.4 Τι είναι η Unity

Η Unity είναι μια πλήρως ολοκληρωμένη μηχανή ανάπτυξης που παρέχει πλούσια out-of-the-box λειτουργικότητά για τη δημιουργία παιχνιδιών και άλλα διαδραστικά περιεχόμενα 3D. Μπορούμε να να συγκεντρώσουμε την τέχνη και τα περιουσιακά μας στοιχεία σε σκηνές και περιβάλλοντα, προσθέτοντας φωτισμό, ήχο, ειδικά εφέ, φυσική και animation, και ταυτόχρονα μπορούμε να δοκιμάσουμε και να επεξεργαστούμε το παιχνίδι μας, και να το δημοσιεύσουμε σε πλατφόρμες που θέλουμε, όπως Mac, PC και Linux desktop υπολογιστές, Windows Store στο Web, iOS, Android, Windows Phone 8, Blackberry 10, Wii U, PS3 και Xbox 360.

Προσφέρει ένα πλήρες σύνολο εργαλείων, ένα διαισθητικό χώρο εργασίας και ταχείες παραγωγικές ροές εργασίας που βοηθούν τους χρήστες να μειώσουν δραστικά το χρόνο, την προσπάθεια και το κόστος κατασκευής διαδραστικού περιεχομένου.

Η Unity είναι ένα σύστημα ανάπτυξης παιχνιδιού: μια ισχυρή μηχανή rendering πλήρως ενσωματωμένη με ένα πλήρες σύνολο διαισθητικών εργαλείων και ταχείων ροών εργασίας για τη δημιουργία διαδραστικών 3D και 2D περιεχομένων, εύκολη multiplatform δημοσίευση, πολύ υψηλή ποιότητα, έτοιμα περιουσιακά στοιχεία του Asset Store και γνώση μέσω της ενεργής κοινότητάς της.

Για τους ανεξάρτητους προγραμματιστές και τα στούντιο, η Unity σπάει τα χρονικά και οικονομικά εμπόδια για τη δημιουργία μοναδικών και όμορφων παιχνιδιών σε οποιαδήποτε πλατφόρμα.

Συγκεντρώνουμε γρήγορα τις σκηνές σε ένα διαισθητικό, επεκτάσιμο χώρο εργασίας Editor. Με το «Play» δοκιμάζουμε και να επεξεργαζόμαστε το ολοκληρωμένο παιχνίδι μας. Η εισαγωγή οποιουδήποτε asset γίνεται με τον πιο ολοκληρωμένο asset αγωγό στην αγορά. Η δημιουργία σύνθετου κόσμου επιτυγχάνεται με κλιμακωτές δομικά σκηνές. Ο κώδικας

γράφεται σύφωνα με τα πρότυπα γλωσσών και έχει πολύ γρήγορους χρόνους μεταγλώττισης. Επίσης μπορούμε να σώσουμε χρόνο χρησιμοποιώντας τα έτοιμα assets που υπάρχουν στο Unity Asset Store, να εμπνευστούμε από αυτά, ή να ζητήσουμε βοήθεια στα Unity Forums.

Μπορούμε να δημιουργήσουμε ένα παιχνίδι με AAA οπτική πιστότητα, rendering δύναμη, ατμόσφαιρα, λεπτομέρειες, φως, σκιά ήχο, δράση και ειδικά εφέ που εκτελούνται ομαλά και καθαρά σε οποιαδήποτε οθόνη. Μπορούμε ακόμα να έχουμε φωτεινή ημέρα, φανταχτερή λάμψη των πινακίδων νέον τη νύχτα, sunshafts ή αγωνιστικά σύννεφα, αμυδρό φωτισμό στους δρόμους τα μεσάνυχτα, σκιερές σήραγγες, εκρήξεις, πυροτεχνήματα, τελειοποιημένα ηχητικά εφέ, πανέμορφο φωτισμό, ακολουθούμενα από ομαλή απόδοση, τα οποία δημιουργούν μια υποβλητική δυναμική παιχνίδι για να γοητεύει τους παίκτες σε οποιαδήποτε πλατφόρμα.

Υπάρχουν ειδικά εργαλεία για τη δημιουργία 2D και 3D περιεχομένου με αποτελεσματικά workflows που χρησιμοποιούνται. Η Unity μπορεί να εισάγει τα μοντέλα και κινούμενα σχέδια από σχεδόν οποιαδήποτε εφαρμογή 3D. Αποθηκεύοντας στο Maya, 3ds Max, Modo, Cinema 4D, Blender ή σε οποιοδήποτε από τα πολλά άλλα εργαλεία δημιουργίας περιεχομένου που υποστηρίζει το λογισμικό, η Unity αμέσως επανεισάγει το ανανεωμένο περιουσιακό στοιχείο και εφαρμόζει αλλαγές σε ολόκληρο το project. Η εισαγωγή των sprites είναι τόσο απλή όσο η τοποθέτησή (drag and drop) τους στο σχετικό φάκελο. Η Unity μπορεί να χωρίσει τα spritesheet αυτόματα και ο χειροκίνητος τεμαχισμός είναι πραγματικά εύκολος, απλά επιλέγουμε με το ποντίκι την επιθυμητή περιοχή.

Διαθέτει ένα μοναδικά ισχυρό και ευέλικτο σύστημα animation, το Mecanim, που φέρνει ανθρώπινους και μη ανθρώπινους χαρακτήρες στη ζωή με απίστευτα φυσικές και ρευστές κινήσεις. Επειδή είναι εγγενώς ενσωματωμένο με τη Unity, το Mecanim καταργεί την ανάγκη ανάπτυξης ενσωμάτωσης ακριβού 3rd part middleware. Μπορούμε να πάρουμε όλα τα εργαλεία και τις ροές εργασίας που χρειάζόμαστε για να δημιουργήσουμε και να χτίσουμε το muscle clips, μείγματα δέντρων, μηχανές καταστάσεων και ελεγκτές απευθείας από τη Unity. Μπορούμε να χρησιμοποιήσουμε το Mecanim για να δημιουργήσουμε animation για τα πάντα, sprites, συνδυασμένα σχήματα και την ένταση του φωτός. Επιπλέον με AnimationEvents μπορούμε να καλέσουμε οποιαδήποτε λειτουργία που θέλουμε από την αναπαραγωγή animation μέσω του κώδικα. Η σταθερότητα και η δύναμη που προσφέρει, σε συνδυασμό με τις νέες βελτιστοποιήσεις, όπως skinned mesh instancing, εξασφαλίζουν εξαιρετικά ομαλή απόδοση της εκτέλεσης.

Παρέχει αξιόπιστη απόδοση και ομαλό framerate σε όλες τις πλατφόρμες που υποστηρίζει ώστε να έχει εξαιρετικό αποτέλεσμα το παιχνίδι. Δημιουργεί τα παιχνίδια που έχουν καλύτερες επιδόσεις κατά το χρόνο εκτέλεσης, μείωση των σημείων συμφόρησης γραφικών, και παίρνει τον έλεγχο της φόρτωσης των assets.

Καμία άλλη μηχανή παιχνιδιού δεν προσφέρει την επιλογή τόσων πολλών εκδόσεων πλατφόρμων με σχεδόν καμία επιλέον ανάπτυξη. Μπορούμε να δημιουργήσουμε μεγάλο gameplay σε όλες τις μεγάλες παγκόσμιες πλατφόρμες με εξαιρετικά αποδοτική δημοσίευση multiplatform. Με ισχυρή μηχανή και εργαλεία, διαισθητικές ροές εργασίας και γρήγορη επανάληψη, έχουμε τον πλήρη έλεγχο για να δημιουργήσουμε και να αναπτύξουμε ομαλά ένα παιχνίδι σε οποιαδήποτε οθόνη.

Υπάρχει πλήρης έλεγχος έκδοσεων για όλα τα περιουσιακά στοιχεία του παιχνιδιού, ενημερώνονται αμέσως από τυχών αλλαγές από άλλα μέλη της ομάδας, και επεκτείνεται η Unity για γενικότερη υποστήριξη VCS. Η Team License είναι ένα προϊόν προσθήκης που επεκτείνει τον επεξεργαστή της Unity για εύκολη τοπική και απομακρυσμένη συνεργασία συνδυάζοντας τη δύναμη και την απλότητα.

Η Unity Pro προσφέρει μια πλήρη επαγγελματική έκδοση της απόδοσης ισχύος και των εργαλείων για τη δημιουργία οποιουδήποτε είδους παιχνιδιού και άλλα πλούσια διαδραστικά περιεχόμενα. Τα εκτεταμένα high-end γραφικά , τα ειδικά εφέ , ο ήχος ,ο φωτισμός , η φυσική και τα εργαλεία βελτιστοποίησης των επιδόσεων είναι διαθέσιμα για να διατηρήσουν τη μεγάλη εμφάνιση του περιεχομένου να λειτουργεί ομαλά σε οποιοδήποτε υλικό.

Η Unity Pro υποστηρίζει τις πλατφόρμες Web και desktop. Τα παιχνίδια κινητών και το περιεχόμενο μπορεί να αναπτυχθεί με επιπλέον προσθήκες κινητού της Unity Pro. Μια ειδική άδεια απαιτείται για την ανάπτυξη των παιχνιδιών κονσόλας με το Unity Pro .

Στο Online Store μπορούμε να αγοράσουμε τη Unity Pro και τις Unity Pro προσθήκες (add – ons) για δημοσίευση σε κινητό. Μπορούμε επίσης να κατεβάσουμε την δωρεάν έκδοση της Unity που περιλαμβάνει μια δοκιμαστική έκδοση τριάντα ημερών της Unity Pro. Η δωρεάν έκδοση του Unity μας προσφέρει περιορισμένα , αλλά ισχυρά , ευέλικτα και υψηλής ποιότητας εργαλεία για την αναπτύξη δημιουργικού και εμπορικού περιεχομένου, και υποστηρίζει την ανάπτυξη για υπολογιστές, κινητά τηλέφωνα και Web πλατφόρμες .

Χρησιμοποιώντας τη Unity μπορούμε να δημοσιεύσουμε το παιχνίδι μας στις ακόλουθες πλατφόρμες: Mac OSX App, Windows Executable, Windows Store, desktop Linux, προγράμματα περιήγησης στο Web μέσω του Unity Web Player, iPhone, iPad, Android τηλέφωνα και ταμπλέτες, Windows Phone 8, Blackberry 10, Wii U, PS3 και Xbox 36.

Η Unity υποστηρίζει τη δημιουργία σχεδόν κάθε διαδραστικού περιεχομένου 2D ή 3D που μπορεί να φανταστεί κανείς, όπως παιχνίδια που βασίζονται σε πρόγραμμα περιήγησης MMOGs, πρώτου προσώπου σκοπευτές (first-person shooters), αγωνιστικά παιχνίδια, παιχνίδια στρατηγικής σε πραγματικό χρόνο, shooters τρίτου προσώπου, παιχνίδια ρόλων, Side-scrollers, σοβαρά παιχνίδια όπως στρατιωτική προσομοίωση, ιατρική εκπαίδευση, εικονική πραγματικότητα, αρχιτεκτονικές εφαρμογές, διαφήμιση και λιανική πώληση.

Οι πόροι είναι ελεύθερα διαθέσιμοι για να βοηθήσουν στη δημιουργία του παιχνιδιού κάνοντας τη διαδικασία μικρότερη, πιο γρήγορη και πολύ πιο διασκεδαστική. Η κοινότητα της Unity, το Asset Store, τα ολοκληρωμένα άρθρα αναλυτικής περιγραφής της λειτουργίας της Unity, η δωρεάν εκπαίδευση και τα tutorials στο διαδίκτυο βοηθάνε τους χρήστες ώστε να ξεπεράσουν τυχόν εμπόδια και να μάθουν συντομεύσεις και βελτιστοποιήσεις.

1.5 Δομή της διπλωματικής εργασίας

Το κεφάλαιο αυτό αποτελεί μια εισαγωγή της διπλωματικής εργασίας. Ακολουθούν άλλα έξι κεφάλαια στα οποία αναλύουμε εις βάθος ολόκληρη τη διαδικασία δημιουργίας και υλοποίησης του παιχνιδιού.

Στο δεύτερο κεφάλαιο παρουσιάζουμε την σχετική έρευνα που διεξάχθηκε στα πλαίσια της διπλωματικής εργασίας όσον αφορά την τεχνολογία και τις γνώσεις που χρειάζονται για την ανάπτυξη τρισδιάστατου παιχνιδιού για φορητές συσκευές. Αναφέρονται τα προγράμματα που υποστηρίζουν την ανάπτυξη τέτοιου είδους παιχνιδιού καθώς και τα σχεδιαστικά προγράμματα τα οποία υποστηρίζουν τη δημιουργία ηρώων για τα αντίστοιχα παιχνίδια και εξηγούμε τους λόγους για τους οποίους επιλέξαμε συγκεκριμένα τη Unity και το Maya .

Στο τρίτο κεφάλαιο αναλύουμε τη τεχνολογική βάση των δύο προγραμμάτων αυτών που χρησιμοποιήσαμε για τη δημιουργία του παιχνιδιού.

Στο τέταρτο κεφάλαιο περιγράφουμε το σχεδιασμό του παιχνιδιού. Αναλύουμε την βασική ιδέα, το σενάριο, και τον σχεδιασμό της γραφικής διεπαφής ώστε να είναι το παιχνίδι φιλικό προς το χρήστη.

Στο πέμπτο αναλύουμε λεπτομερώς την υλοποίηση του παιχνιδού καθώς και όλες τις διαδικασίες που ακολουθήθηκαν έως την ολοκλήρωσή του.

Στο έκτο κεφάλαιο αναφέρουμε τις εντυπώσεις του παιχνιδιού από χρήστες που το δοκίμασαν και το αξιολόγησαν., και τις αλλαγές που κάναμε στο παιχνίδι σύμφωνα με τις προτιμήσεις τους και τις προτάσεις τους για βελτιώσεις.

Στο έβδομο κεφάλαιο ανακεφαλαιώνουμε και αναφέρουμε τα συμπεράσματα που προκύπτουν από την όλη διαδικασία, και προτείνουμε μελλοντικές επεκτάσεις.

ΚΕΦΑΛΑΙΟ 2

ΕΠΙΣΚΟΠΗΣΗ ΣΧΕΤΙΚΗΣ ΈΡΕΥΝΑΣ

2.1 Εισαγωγή

Στο κεφάλαιο αυτό δείχνουμε το ενδιαφέρον των χρηστών προς τις εφαρμογές Android και iOS και την ταχύτατη άνοδό τους στην αγορά.

Αναλύουμε τις διαφορετικές κατηγορίες παιχνιδιών που υπάρχουν στα κινητά και τα κύρια χαρακτηριστικά της συγκεκριμένης κατηγορίας παιχνιδιού που υλοποιήσαμε.

Στη συνέχεια εξηγούμε τα χαρακτηριστικά των παιχνιδιών των κινητών συσκευών καθώς και ποιές πλατφόρμες υποστηρίζουν την ανάπτυξή τους.

Έπειτα αναφέρουμε τις μηχανές παιχνιδιών αναλύοντας τα στοιχεία που τις αποτελούν και τον τρόπο που λειτουργούν.

Εμβαθύνοντας στο hardware που υποστηρίζει τις μηχανές παιχνιδιών, αναλύουμε τον αγωγό γραφικών, τις διεπαφές γραφικών και βασικές σχετικές έννοιες και όρους.

Αναφέρουμε τις κατηγορίες μηχανών παιχνιδιού υποστηρίζουν τη δημιουργία τρισδιάστατων παιχνιδιών για φορητές συσκευές, καθώς και τα κύρια χαρακτηριστικά τους, τις διαφορές τους και τα πλεονεκτήματά τους.

Εξηγούμε τον λόγο που επιλέξαμε τη Unity ανάμεσα στις άλλες μηχανές και αναφέρουμε χαρακτηριστικά της και βασικά πλεονεκτήματά της.

Αναλύουμε τον τρόπο που συνδέονται τα γραφικά του υπολογιστή με τις μηχανές παιχνιδιού και πως λειτουργούν. Αναφέρουμε ποιά προγράμματα σχεδίασης γραφικών και ανάπτυξης τρισδιάστατων χαρακτήρων υποστηρίζει η Unity, ποια είναι τα βασικά τους χαρακτηριστικά και τα συγκρίνουμε μεταξύ τους ώστε να καταλήξουμε στο καταλληλότερο για μας πρόγραμμα.

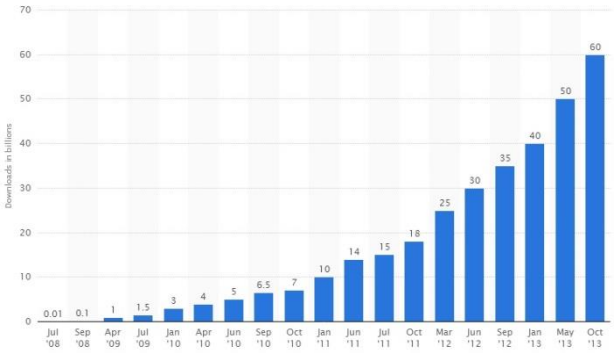
Καταλήγοντας στο Maya αναλύουμε τα χαρακτηριστικά του, τις δυνατότητές του και τις λειτουργίες που υποστηρίζει.

2.2 Στατιστικά περί εφαρμογών

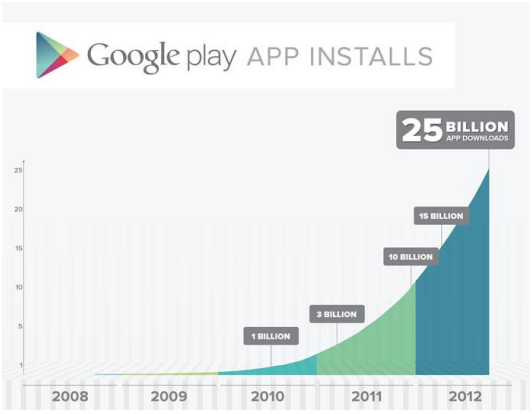
Τα δεδομένα των εφαρμογών και ο αριθμός τους αλλάζουν με πολύ έντονους ρυθμούς. Σύμφωνα με στοιχεία που βρήκαμε για το 2013 μπορούμε να διακρίνουμε το τεράστιο ενδιαφέρον της αγοράς προς αυτή την κατηγορία της τεχνολογίας.

Τον Οκτώβρη του 2013 η Apple ανακοίνωσε ότι στο App Store υπάρχουν περισσότερες από 1.000.000 εφαρμογές, ενώ μόλις τον Ιούνιο ήταν 900.000. Επίσης ανακοίνωσε ότι οι εφαρμογές που έχουν κατεβάσει οι χρήστες από το App Store ξεπερνούν τα 60 δισεκατομμύρια, ενώ τον Μάιο έφταναν τα 50 δισεκατομμύρια όπου υπήρχαν περισσότερες από 850.000 εφαρμογές για iPhone και 350.000 εφαρμογές για iPad. Αυτό αποτελεί μεγάλη πρόοδο καθώς μόλις το 2012 οι εφαρμογές που είχαν κατεβεί από χρήστες άγγιζαν τα 40 δισεκατομμύρια και οι διαθέσιμες εφαρμογές ήταν περίπου 775.000 και περίπου 300.000 για iPad. Στον πίνακα φαίνεται ο αριθμός των λήψεων των εφαρμογών από το App Store της Apple από τον Ιούλιο του 2008 έως τον Οκτώβριο του 2013.

Cumulative number of apps downloaded from the Apple App Store from June 2008 to October 2013 (in billions)



Τον Ιούλιο του 2013 η Google ανακοίνωσε ότι οι χρήστες Android έχουν κατεβάσει πάνω 50 δισεκατομμύρια εφαρμογές από το Google Play, και οι διαθέσιμες εφαρμογές είναι πάνω από 1.000.000. Αυτό αποτελεί μεγάλη άνοδο για την Google καθώς μόλις το 2012 οι εφαρμογές που είχαν ληφθεί από τους χρήστες άγγιζαν τα 25 δισεκατομμύρια διαθέτοντας 675.000 εφαρμογές.



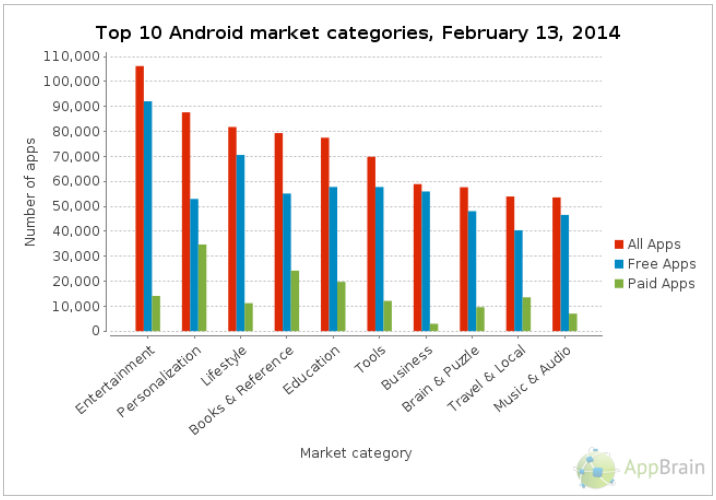
Σύμφωνα με στατιστικά του 2013, οι 9 στους 10 χρήστες smartphone χρησιμοποιούν το τηλέφωνό τους σε καθημερινή βάση. Η πιο δημοφιλής δραστηριότητα είναι τα γραπτά μηνυμάτα αποτελώντας το 92% της χρήσης του κινητού, ακολουθούμενη από την περιήγηση στο Internet με 84%, την ηλεκτρονική αλληλογραφία-emails με 76% και τη χρήση παιχνιδιών με 66%. Κατά μέσο όρο μηνιαίως ένας χρήστης iPhone κατεβάζει το μέγιστο 48 εφαρμογές από τις οποίες το 26% είναι παιχνίδια, ενώ ένας χρήστης Android κατεβάζει 35 εφαρμογές από τις οποίες το 20% είναι παιχνίδια. Όπως φαίνεται η κατηγορία των παιχνιδιών έχει μεγάλη ζήτηση και καταλαμβάνει σημαντικό κομμάτι στη χρήση των smartphones και στην δραστηριότητα των χρηστών, για αυτό και απασχολεί πολλούς προγραμματιστές.

2.3 Κατηγορίες παιχνιδιών

Τα παιχνίδια στο Google Play χωρίζονται στις εξής κατηγορίες :

- Arcade & Action
- Αγώνες (Racing)
- Αθλητικά παιχνίδια (Sports Games)
- Γραφικά στοιχεία (Widgets)
- Εύκολα (Casual)
- Ζωντανή ταπετσαρία (Live Wallpaper)
- Σπαζοκεφαλιές και κουίζ (Brain & Puzzle)
- Χαρτιά και καζίνο (Cards & Casino)

Σύμφωνα με τα στοιχεία της AppBrain οι πιο δημοφιλείς κατηγορίες εφαρμογών στην αγορά του Android είναι η κατηγορία της Διασκέδασης η οποία περιλαβάνει κατά κύριο λόγο τα παιχνίδια. Στον πίνακα αναγράφεται ο συνολικός αριθμός των εφαρμογών, ελεύθερων εφαρμογών και εφαρμογών επί πληρωμής για κάθε κατηγορία στην αγορά Android:



Σύμφωνα με στοιχεία του Φεβρουαρίσου 2014, ο επόμενος πίνακα δείχνει τα στοιχεία για όλες τις κατηγορίες των εφαρμογών παιχνιδιών Android, που περιλαμβάνουν τον συνολικό αριθμό εφαρμογών της συγκεκριμένης κατηγορίας, τη μέση βαθμολογία σε αστέρια, τον αριθμό και το ποσοστό των εφαρμογών της κατηγορίας που έχουν περισσότερες από 50,000 λήψεις, τον αριθμό και το ποσοστό των επί πληρωμής εφαρμογών σε αυτήν την κατηγορία, τη μέση τιμή των εφαρμογών επί πληρωμής σε αυτή την κατηγορία, και τον αριθμό και το ποσοστό των χαμηλής ποιότητας εφαρμογών σε αυτή την κατηγορία.

Android games

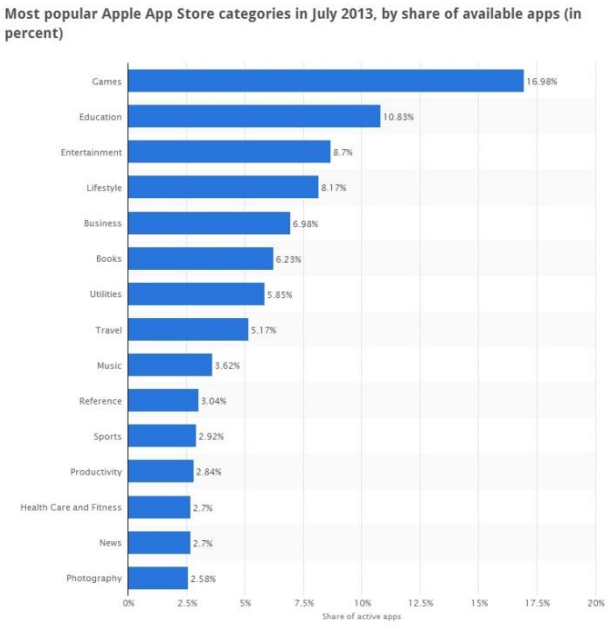
Category name	All apps	Average rating	Apps with >50K downloads	Paid apps	Average price	Low quality apps
Arcade & Action	36324	3.9	6776 (19 %)	6750 (19 %)	$ 1.83	3500 (10 %)
Brain & Puzzle	57652	4.0	6072 (11 %)	9591 (17 %)	$ 1.63	6254 (11 %)
Cards & Casino	12887	4.0	1759 (14 %)	2019 (16 %)	$ 2.56	3393 (26 %)
Casual	39061	3.9	7416 (19 %)	6679 (17 %)	$ 1.96	4223 (11 %)
Sports Games	6321	3.9	1380 (22 %)	1240 (20 %)	$ 1.92	917 (15 %)
Racing	5177	3.9	1732 (33 %)	751 (15 %)	$ 1.68	540 (10 %)

Όπως μπορούμε να δούμε η κατηγορία Arcade & Action είναι η τρίτη κατηγορία με τον μεγαλύτερο αριθμό εφαρμογών σε σχέση με τις υπόλοιπες κατηγορίες παιχνιδιών, και δεύτερη κατηγορία με τις περισσότερες λήψεις.

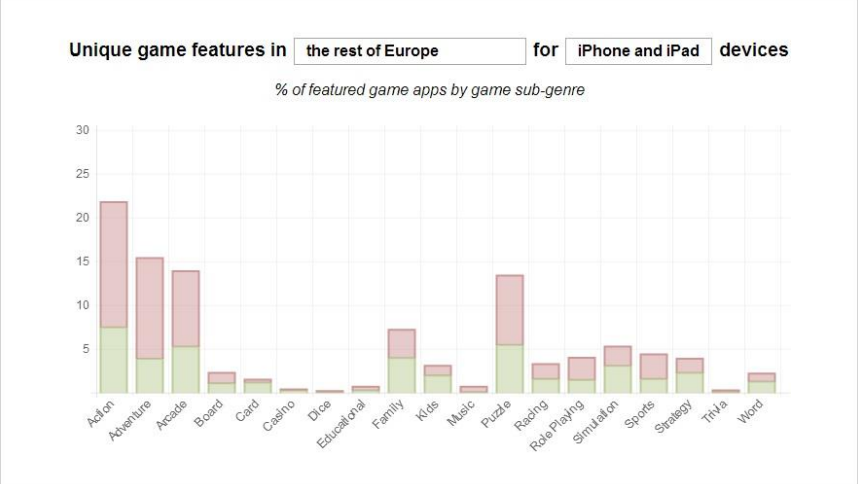
Τα παιχνίδια στο App Store χωρίζονται στις εξής κατηγορίες :

- Δράση (Action)
- Περιπέτεια (Adventure)
- Arcade
- Σπαζοκεφαλιές (Board)
- Χαρτιά (Card)
- Καζίνο (Casino)
- Ζάρια (Dice)
- Εκπαιδευτικά (Educational)
- Οικογενειακά (Family)
- Μουσικά (Music)
- Παζλ (Puzzle)
- Αγώνες (Racing)
- Παιχνίδια ρόλων (Role Playing)
- Προσομοίωση (Simulation)
- Αθλητικά (Sports)
- Στρατηγική (Strategy)
- Εύκολα (Trivia)
- Λέξεις (Word)

Σύμφωνα με τα στοιχεία της Statistics Portal η πιο δημοφιλής κατηγορία εφαρμογών της Apple App Store για το 2013 είναι τα παιχνίδια, με αρκετά μεγαλύτερο ποσοστό εφαρμογών. Αναλύοντας τα στοιχεία που ανακοινώθηκαν στο iTunes App Store το 2013 μπορούμε να έχουμε μια πιο αναλυτική εικόνα όσον αφορά το ρόλο που έχουν τα παιχνίδια σε σχέση με τις υπόλοιπες εφαρμογές και ποια κατηγορία παιχνιδιών κεντρίζει περισσότερο το ενδιαφέρον των χρηστών. Αν και μόνο το 16,98% είναι παιχνίδια, αποτελούν το ήμισυ των εφαρμογών που εμφανίζονται στην κεντρική σελίδα App Store.



Το επόμενο διαγράμματα δείχνει το ποσοστό των προτεινόμενων εφαρμογών σε κάθε κατηγορία, σε χώρες τις Ευρώπης συμπεριλαμβανομένης και της Ελλάδας. Με πράσινο χρώμα είναι οι δωρεάν εφαρμογές, ενώ με κόκκινο είναι οι εφαρμογές επί πληρωμής. Παρατηρούμε πως δεν υπάρχει μεγάλη διακύμανση από χώρα σε χώρα. Τα παιχνίδια Δράσης (Action) είναι τα πιο δημοφιλή, και ακολουθούν τα παιχνίδια Περιπέτειας, Arcade και Παζλ.



Το παιχνίδι που υλοποιήσαμε ανήκει στη δημοφιλέστερη κατηγορία παιχνιδιού, δηλαδή στην Arcade & Action στο Google Play, ενώ στο App Store ανήκει και στις τρεις κατηγορίες Action, Adventure και Arcade καθώς περιλαμβάνει τα περισσότερα χαρακτηριστικά τους και τα συνδυάζει, τα οποία αναλύουμε στη συνέχεια.

2.3.1 Arcade game

Τα παιχνίδια αυτής της κατηγορίας έχουν συχνά μικρά επίπεδα, απλό και διασθήτικο σύστημα ελέγχου, και ταχέως αυξανόμενη δυσκολία. Ο χρήστης μπορεί να παίζει όσο διάστημα παραμένει ζωντανός ο ήρωάς του. Συνήθως αυτά τα παιχνίδια έχουν σχεδιαστεί με Flash / Java / DHTML και να τρέχουν άμεσα σε web - browsers .

Έχουν μια απλοποιημένη μηχανή φυσικής και δεν απαιτούν πολύ χρόνο εκμάθησης προκειμένου να διατηρηθεί η συνιστώσα της δράσης τους. Οι AI αντίπαλοι μερικές φορές έχουν προγραμματιστεί ώστε να βρίσκονται πάντα κοντά στον ήρωα (rubberband effect).

Για το χειρισμό υπάρχει ένας ελεγκτής arcade controller, οι λειτουργίες του παιχνιδιού γίνονται με διαισθητικό έλεγχο, υπάρχει γρήγορα αυξανόμενη δυσκολία, και τα παιχνίδια κινητών με διαισθητικό έλεγχο είναι συχνά με μικρές πίστες.

2.3.2 Action game

Τα παιχνίδια δράσης δίνουν έμφαση σε φυσικές προκλήσεις, στο συντονισμό του χεριού με το μάτι και την αντίδραση στο χρόνο. Περιλαμβάνει παιχνίδια μάχης(fighting games), παιχνίδια με πυροβολισμούς (shooter games), και παιχνίδια πλατφόρμας (platform games), τα οποία θεωρούνται ευρέως ότι είναι τα πιο σημαντικά παιχνίδια δράσης, αν και κάποια παιχνίδια στρατηγικής πραγματικού χρόνου θεωρούνται ότι είναι παιχνίδια δράσης.

Σε ένα παιχνίδι δράσης ο παίκτης ελέγχει τυπικά το avatar του πρωταγωνιστή. Το avatar πρέπει να μετακινηθεί μέσα στο επίπεδο, να συλλέξει αντικείμενα, να αποφύγει τα εμπόδια, και να αντιμετωπίσει τους εχθρούς με διάφορες επιθέσεις. Οι επιθέσεις του εχθρού και τα εμπόδια καταστρέφουν την υγεία και τις ζωές του avatar, και το παιχνίδι τελειώνει ξεμείνει από ζωές. Εναλλακτικά, ο παίκτης κερδίζει το παιχνίδι με την ολοκλήρωση μιας σειράς επιπέδων. Στόχος του παίκτη είναι να μεγιστοποιεί το σκορ τους συλλέγοντας αντικείμενα νικώντας τους εχθρούς.

Καθορισμός στοιχείων

Αυτή η κατηγορία περιλαμβάνει οποιοδήποτε παιχνίδι όπου η πλειοψηφία των προκλήσεων είναι φυσικές δοκιμασίες ελέγχου των ικανοτήτων. Τα παιχνίδια δράσης μπορεί μερικές φορές να ενσωματώνουν προκλήσεις όπως αγώνες, μάχη, ή τη συλλογή αντικειμένων, αλλά δεν είναι κεντρικής σημασίας. Οι παίκτες μπορούν επίσης να εφαρμόσουν στρατηγική και να εξερευνήσουν τις προκλήσεις, αλλά αυτά τα παιχνίδια απαιτούν πρώτα και πάνω από

όλα υψηλή ταχύτητα αντίδρασης και καλό συντονισμό χεριού-ματιού. Ο παίκτης είναι συχνά κάτω από πίεση χρόνου, και δεν υπάρχει αρκετός χρόνος για πολύπλοκο στρατηγικό σχεδιασμό. Σε γενικές γραμμές, τα πιο γρήγορα παιχνίδια δράσης είναι πιο δύσκολα.

Σχεδιασμός του παιχνιδιού

Επίπεδα

Οι παίκτες πρέπει να ολοκληρώσουν μια σειρά από επίπεδα. Τα επίπεδα συχνά ομαδοποιούνται με βάση το θέμα, με παρόμοια γραφικά και εχθρούς. Κάθε επίπεδο περιλαμβάνει μια ποικιλία από προκλήσεις, πχ πυροβολάνε ή πετάνε αντικείμενα στον εχθρό σε ένα shooter παιχνίδι, τις οποίες ο παίκτης πρέπει να ξεπεράσει για να κερδίσει το παιχνίδι.

Τα παλαιότερα παιχνίδια ανάγκαζαν τους παίκτες να ξεκινήσουν το επίπεδο πάλι από την αρχή αφού πεθάνουν, αν και είχαν εξελιχθεί για να προσφέρουν την αποθήκευση του παιχνιδιού και των σημείων ελέγχου (checkpoints) ώστε να επιτρέψει στον παίκτη να κάνει μερική επανεκκίνηση (partway) μέσα μέσα από ένα επίπεδο. Τώρα τα περισσότερα παιχνίδια επιτρέπουν την « ανάσταση » ή « κλωνοποίηση » και τη δυνατότητα της ανάκτησης των χαμένων αντικειμένων μετά από το θάνατο για ένα ορισμένο ποσό νομισμάτων στο παιχνίδι, που συνήθως αυξάνεται εκθετικά όσο αυξάνονται οι φορές που πεθαίνει ο παίκτης. Τα εμπόδια και οι εχθροί σε ένα επίπεδο συνήθως δεν διαφέρουν μεταξύ των τμημάτων του παιχνιδιού, επιτρέποντας στους παίκτες να μάθουν από τη δοκιμή και τα λάθη τους. Ωστόσο, τα επίπεδα μερικές φορές προσθέτουν ένα τυχαίο στοιχείο, όπως έναν εχθρό που εμφανίζεται τυχαία ή ένα απρόβλεπτο μονοπάτι.

Για επίπεδα που απαιτούν διερεύνηση, ο παίκτης μπορεί να χρειαστεί να ψάξει για μια έξοδο από το επίπεδο που είναι κρυμμένη ή να φυλάσσονται από τους εχθρούς. Αυτά τα επίπεδα μπορεί επίσης να περιέχουν μυστικά, τα οποία μπορεί να είναι κρυμμένα ή δυσπρόσιτα αντικείμενα, ή χώρους που περιέχουν κάτι πολύτιμο. Το βραβείο μπορεί να είναι ένα μπόνους ή μια μη - τυπική έξοδος που επιτρέπει σε έναν παίκτη να αποκτήσει πρόσβαση σε ένα κρυφό επίπεδο.

Ικανότητες του χαρακτήρα

Στα περισσότερα παιχνίδια δράσης, ο παίκτης ελέγχει ένα avatar ως πρωταγωνιστή. Το avatar έχει τη δυνατότητα να πλοηγηθεί και να ελιχθεί, και συχνά συλλέγει ή χειρίζεται αντικείμενα. Έχει μια σειρά από άμυνες και επιθέσεις, καθώς μπορεί να πυροβολάει ή να γρονθοκοπάει.

Οι παίκτες μπορούν να βρουν μια δύναμη (power-up) μέσα στον κόσμο του παιχνιδιού που χορηγεί προσωρινές ή μόνιμες βελτιώσεις στις ικανότητές τους. Για παράδειγμα, το avatar μπορεί να αποκτήσει μια αύξηση στην ταχύτητα, πιο ισχυρές επιθέσεις, ή μια προσωρινή

ασπίδα από επιθέσεις. Μερικά παιχνίδια δράσης επιτρέπουν ακόμη στους παίκτες να ξοδέψουν τους αναβαθμισμένους βαθμούς σε δυνάμεις της επιλογής τους. Σε παιχνίδια δράσης, το μεγαλύτερο μέρος της ανάπτυξης του χαρακτήρα του avatar προέρχεται από power-ups και νέες κινήσεις, και οι νοητικές καταστάσεις συνήθως δεν αλλάζουν ή δεν προοδεύουν.

Εμπόδια και εχθροί

Σε παιχνίδια δράσης που αφορούν την πλοήγηση σε ένα χώρο, οι παίκτες θα συναντήσουν εμπόδια, παγίδες και εχθρούς. Οι εχθροί συνήθως ακολουθούν τα καθιερωμένα πρότυπα και επιτίθονται στον παίκτη, αν και νεότερα παιχνίδια δράσης μπορεί να κάνουν χρήση της πιο περίπλοκης τεχνητής νοημοσύνης για να καταδιώξουν τον παίκτη. Οι εχθροί εμφανίζονται μερικές φορές σε ομάδες ή σε κύματα, με τους εχθρούς να αυξάνονται σε δύναμη και αριθμό μέχρι το τέλος του επιπέδου.

Υγεία και ζωή

Σε πολλά παιχνίδια δράσης, το avatar έχει ένα ορισμένο αριθμό των χτυπημάτων που δέχεται (hitpoints) ή υγείας, ο οποίος εξαντλείται από τις επιθέσεις του εχθρού και από άλλους κινδύνους. Μερικές φορές, η υγεία μπορεί να αναπληρώνονται με τη συλλογή ένος αντικειμένου στο παιχνίδι. Όταν ο παίκτης εξαντλήσει την υγεία του, το avatar πεθαίνει. Στο avatar του παίκτη δίνεται συχνά ένας μικρός αριθμός από ευκαιρίες για να ξαναπροσπαθήσει μετά το θάνατο, συνήθως αναφέρεται ως ζωή. Κατά την έναρξη μιας νέας ζωής, ο παίκτης συνεχίζει το παιχνίδι, είτε από την ίδια θέση που έχασε τη ζωή του, ένα σημείο ελέγχου, ή από την αρχή του επιπέδου. Κατά την έναρξη μιας νέας ζωής, το avatar είναι συνήθως ανίκητος για μερικά δευτερόλεπτα για να επιτρέψει στον παίκτη να επαναπροσανατολιστεί. Οι παίκτες μπορούν να κερδίσουν επιπλέον ζωές φθάνοντας σε ένα συγκεκριμένο σκορ ή βρίσκοντας ένα αντικειμένο στο παιχνίδι. Τα Arcade παιχνίδια περιορίζουν τον αριθμό των ζωών των παικτών.

Γραφικά και διεπαφή

Τα παιχνίδια δράσης λαμβάνουν χώρα είτε σε 2D ή 3D από ποικίλες οπτικές γωνίες. Σε 3D παιχνίδια δράσης, η προοπτική είναι συνήθως συνδεδεμένη με τον avatar από μια προοπτική πρώτου προσώπου ή τρίτου προσώπου. Ωστόσο, ορισμένα 3D παιχνίδια προσφέρουν ένα πλαίσιο ευαίσθητης προοπτικής που ελέγχεται από μια κάμερα τεχνητής νοημοσύνης. Τα περισσότερα από αυτά που χρειάζεται να γνωρίζει ο παίκτης περιέχονται σε μία οθόνη, αν και τα παιχνίδια δράσης συχνά κάνουν χρήση του heads-up display που εμφανίζουν σημαντικές πληροφορίες όπως η υγεία ή πυρομαχικά.

Βαθμολόγηση και νίκη

Τα παιχνίδια δράσης έχουν την τάση να θέτουν απλούς στόχους , και η επίτευξή τους είναι προφανής. Ένας κοινός στόχος είναι η έρευση του σημείου τερματισμού που μπορεί να

περιλαμβάνει έναν θησαυρό. Αυτό συχνά παρουσιάζεται με τη μορφή μιας δομημένης ιστορίας , με αίσιο τέλος μετά τη νίκη του παιχνιδιού.

Πολλά παιχνίδια δράσης παρακολουθούν το σκορ του παίκτη. Πόντοι απονέμονται για την ολοκλήρωση ορισμένων προβλημάτων (challenges), ή τη νίκη ορισμένων εχθρών. Μερικές φορές τα παιχνίδια δράσης θα προσφέρουν μπόνους αντικείμενα που αυξάνουν το σκορ του παίκτη. Δεν υπάρχει καμία ποινή για τη μη συλλογή τους , αν και αυτά τα μπόνους αντικείμενα μπορεί να ξεκλειδώνουν κρυμμένα επίπεδα ή συγκεκριμένα γεγονότα. Σε πολλά παιχνίδια δράσης η επίτευξη μιας υψηλής βαθμολογίας είναι ο μόνος στόχος , και τα επίπεδα αυξάνουν σε δυσκολία μέχρι να χάσει ο παίκτης. Τα Arcade παιχνίδια μπορεί είτε να μην τερματίζουν ποτέ ή να έχουν διακριτές συνθήκες νίκης αναλόγως το θέμα τους.

2.3.3 Adventure game

Ένα παιχνίδι περιπέτειας είναι ένα παιχνίδι στο οποίο ο παίκτης αναλαμβάνει το ρόλο του πρωταγωνιστή σε μια διαδραστική ιστορία η οποία καθοδηγείται από την έρευνα και την επίλυση γρίφων. Σχεδόν όλα τα παιχνίδια περιπέτειας (κείμενο και γραφικά) έχουν σχεδιαστεί για έναν παίκτη, και δεδομένου της έμφασης στην ιστορία και τον χαρακτήρα καθιστά δύσκολο το σχεδιασμό για πολλούς παίκτες (multi-player). Το παιχνίδι περιπέτειας ορίζεται από το θέμα που πραγματεύεται, τη δραστηριότητα της περιπέτειας. Βασικά στοιχεία του είδους περιλαμβάνουν αφήγηση, εξερεύνηση και επίλυση γρίφων.

Σχέση με άλλα είδη

Οι προκλήσεις μάζης (combat) και δράσης είναι περιορισμένες ή απουσιάζουν από τα παιχνίδια περιπέτειας, ξεχωρίζοντας έτσι από τα παιχνίδια δράσης . Μειώνοντας την έμφαση στην καταπολέμηση των εχθρών, δεν σημαίνει ότι δεν υπάρχει καμία σύγκρουση στα παιχνίδια περιπέτειας, απλά η μάχη δεν είναι η κύρια δραστηριότητα. Μερικά παιχνίδια αναμειγνύουν τη δράση με τη περιπέτεια σε όλη την διάρκεια του παιχνιδιού. Αυτά τα υβριδικά παιχνίδια δράσης-περιπέτειας (action-adventure) περιλαμβάνουν περισσότερο σωματικές προκλήσεις από τα καθαρά παιχνίδια περιπέτειας , καθώς επίσης και ταχύτερο ρυθμό. Τα παιχνίδια τα οποία εμπεριέχουν αρκετές μη φυσικές προκλήσεις θεωρούνται παιχνίδια δράσης περιπέτειας, και δεν έχουν καμία δεξιότητα συστήματος , μάχης, ή έναν αντίπαλο που θα ηττηθεί μέσα από στρατηγική και τακτική, αλλά μπορεί για παράδειγμα να απαιτούν από τους παίκτες να περιηγηθούν σε λαβύρινθους .

Σχεδιασμός του παιχνιδού

Τα παιχνίδια περιπέτειας είναι για έναν παίκτη σε μεγάλο βαθμό με γνώμονα την ιστορία. Περισσότερο από κάθε άλλο είδος, τα παιχνίδια περιπέτειας εξαρτώνται από την ιστορία και ρυθμίζονται για να δημιουργήσουν μία συναρπαστική εμπειρία ενός παίκτη. Έχουν καθοριστεί τυπικά σε ένα καθηλωτικό περιβάλλον, συχνά έναν φανταστικό κόσμο, και

προσπαθούν να ποικίλλουν στις ρυθμίσεις από κεφάλαιο σε κεφάλαιο, για να προσθέτουν καινοτομία και ενδιαφέρον για την εμπειρία. Η κωμωδία είναι εμφανής, και τα παιχνίδια συχνά είναι σχεδιασμένα ώστε να έχουν κωμικές αντιδράσεις όταν οι παίκτες επιχειρούν πράξεις ή συνδυασμούς πράξεων που είναι γελοίες ή αδύνατες.

Ο πρωταρχικός στόχος στα παιχνίδια περιπέτειας είναι η ολοκλήρωση της ανατεθημένης αναζήτησης. Η υψηλή βαθμολογία παρέχει στον παίκτη ένα δευτερεύοντα στόχο, και χρησιμεύσει ως δείκτης της εξέλιξης. Η πρωταρχική προϋπόθεση αποτυχίας στα παιχνίδια περιπέτειας, που κληρονόμησε περισσότερο από τα παιχνίδια δράσης, είναι ο θάνατος του παίκτη.

2.4 Παιχνίδια κινητών

Ένα παιχνίδι κινητού είναι ένα βίντεοπαιχνίδι που παίζεται σε ένα τηλέφωνο, smartphone, PDA, υπολογιστή tablet, φορητή συσκευή αναπαραγωγής πολυμέσων ή αριθμομηχανή, αλλά δεν περιλαμβάνονται τα παιχνίδια που παίζονται σε ειδικά φορητά συστήματα βίντεοπαιχνιδιών, όπως το Nintendo 3DS ή PlayStation Vita.

Το πρώτο downloadable περιεχόμενο εισήχθη ήδη το 2000. Ωστόσο η διανομή των παιχνιδιών των κινητών από φορείς εκμετάλλευσης κινητής τηλεφωνίας παρέμεινε στη μορφή των τυχερών παιχνιδιών μέχρι που το ξεκίνησε η Apple από το App Store που παρουσιάστηκε το 2008. Το App Store, που ήταν το πρώτο κατάστημα που λειτουργούσε άμεσα από τον κάτοχο της κινητής πλατφόρμας, άλλαξε σημαντικά τη συμπεριφορά των καταναλωτών και γρήγορα διεύρυνε τις αγορές για τα κινητά παιχνίδια, καθώς σχεδόν κάθε ιδιοκτήτης smartphone άρχισε να κατεβάζει υλικό παιχνιδιών για κινητά τηλέφωνα. Τα παιχνίδια κινητών παίζονται στην ίδια τη συσκευή ή πάνω στο mobile cloud.

2.4.1 Ιστορία

Η έναρξη του App Store της Apple το 2008 άλλαξε ριζικά την αγορά. Πρώτα απ 'όλα διευρύνθηκαν οι δυνατότητες των καταναλωτών καθώς μπορούν να επιλέγουν από πού να κατεβάσουν τις εφαρμογές, από το κατάστημα εφαρμογών της συσκευής , το κατάστημα του φορέα ή το καταστήματα τρίτων μέσω του ανοιχτού διαδικτύου. Οι χρήστες της Apple όμως, μπορούν να χρησιμοποιήσουν μόνο το Apple App Store, δεδομένου ότι η Apple απαγορεύει τη διανομή των εφαρμογών μέσω οποιουδήποτε άλλου καναλιού διανομής. Δεύτερον, οι προγραμματιστές εφαρμογών κινητών μπορούν να ανεβάσουν απευθείας στο App Store χωρίς τις τυπικές μακρές διαπραγματεύσεις με τους εκδότες και τους φορείς, γεγονός που αύξησε το μερίδιο των εσόδων τους και έκανε την ανάπτυξη των παιχνιδιών κινητών πιο κερδοφόρα.

Τρίτον, η στενή ανάμειξη του App Store με την ίδια τη συσκευή οδήγησε πολλούς καταναλωτές να δοκιμάσουν τις εφαρμογές , και η αγορά παιχνιδιών έλαβε μια σημαντική ώθηση.

Κατά συνέπεια, τα πρώτα εμπορικά άκρως επιτυχημένα παιχνίδια κινητών εμφανίστηκαν λίγο μετά την έναρξη του App Store. Η αρχική έκδοση του Angry Birds αναπτύχθηκε από την Rovio Entertainment, κυκλοφόρησε στις iOS το Δεκέμβριο του 2009 .

Σήμερα τα smart phone και tablet παιχνίδια έχουν διανύσει πολύ δρόμο. Τα γραφικά τους είναι περίπου το ίδιο όπως θα περίμενε κανείς για ένα παιχνίδι κονσόλας 6ης ή 5ης γενιάς, η οποία δεν μπορεί να φαίνεται σαν μια πολύ μεγάλη βελτίωση αλλά θεωρείται βελτίωση επειδή το παιχνίδι παίζεται σε ένα κινητό τηλέφωνο.

Μετά την ολοκλήρωση των 3D APIs σε κινητές πλατφόρμες, το παγκόσμιο mobile gaming έδωσε το δικό του στίγμα στα παιχνίδια. Μετά την τεράστια επιτυχία του τουρνουά Arena Soccer 3D από την Mobilenter το οποίο είχε πάνω από 35 εκατομμύρια downloads μέσα σε μόλις μία εβδομάδα, η ανάπτυξη των παιχνιδιών 3D έγινε η κύρια περιοχή της ανάπτυξης των παιχνιδιών των κινητών και το mobile gaming έγινε μία από τις πιο σημαντικές πλατφόρμες παιχνιδιών .

2.4.2 Διαφορετικές πλατφόρμες

Τα παιχνίδια κινητών αναπτύσσονται με τη χρήση πλατφορμών και τεχνολογιών, όπως το Windows Mobile, Palm OS, Symbian, Adobe Flash Lite, DoJa της NTT DoCoMo, της Sun Java ME, BREW της Qualcomm, WIPI, το Apple iOS, Windows Phone 8 ή το Google Android.

Η Java είναι η πιο κοινή πλατφόρμα για κινητά παιχνίδια, ωστόσο τα όρια των επιδόσεων της οδηγούν στην έκδοση διαφόρων ιθαγενών δυαδικών μορφών για πιο εξελιγμένα παιχνίδια.

2.4.3 Κοινά όρια των κινητών παιχνίδια

Τα παιχνίδια κινητών τείνουν να είναι μικρά σε έκταση και συχνά εξαρτώνται από το καλό gameplay και όχι τα γραφικά , λόγω της έλλειψης της επεξεργαστικής ισχύος των συσκευών του πελάτη. Ένα σημαντικό πρόβλημα για τους προγραμματιστές και τους εκδότες των παιχνιδιών των κινητών είναι η περιγραφή ενός παιχνιδιού με τόση λεπτομέρεια που να δίνει στον πελάτη αρκετές πληροφορίες για να κάνει μια απόφαση αγοράς. Τα περισσότερα από τα παιχνίδια κινητών είναι χτισμένα γύρω από ένα συγκεκριμένο θέμα ή έχουν κάποια συγκεκριμένη γραμμή ιστορίας. Τα παιχνίδια κινητών πωλούνται κυρίως μέσω του Δικτύου Carriers (φορείς εκμετάλλευσης πυλών) και αυτό σημαίνει ότι υπάρχουν μόνο λίγες γραμμές κειμένου και ίσως ένα στιγμιότυπο οθόνης του παιχνιδιού για να προσελκύσει τον πελάτη. Για αυτό αναγράφεται η ποιότητα του παιχνιδιού και η κατηγορία στην οποία ανήκει.

Οι πρόσφατες καινοτομίες στον τομέα των παιχνιδιών των κινητών περιλαμβάνουν Singleplayer, Multiplayer και 3D γραφικά. Τα Multiplayer παιχνίδια βρίσκουν γρήγορα ένα ακροατήριο, καθώς οι προγραμματιστές επωφελούνται από τη δυνατότητα να μπορείς να παίξεις ενάντια σε άλλους, μια φυσική προέκταση της σύνδεσης του κινητού τηλεφώνου. Επίσης δημοφιλοί είναι τα τυχερά παιχνίδια και τα κουίζ.

2.4.4 Όρια για την ανάπτυξη του κλάδου της κινητής παιχνίδια

Τα κύρια εμπόδια στην ανάπτυξη του κλάδου των παιχνιδιών των κινητών στην Ευρώπη το 2013 είναι ότι οι προγραμματιστές παιχνιδιών πρέπει να ακολουθούν τα πρότυπα ανάπτυξης, να είναι καλά ενημερωμένοι σχετικά με το hardware και να ασκούν όσο το δυνατόν μεγαλύτερη επιρροή σε αυτό, καθώς και στην ανάπτυξη των λειτουργικών συστημάτων. Είναι απαραίτητη η αξιολόγηση των API, καθώς οι εννοποιημένες μορφές παραγωγής απαιτούν να συγχρονιστούν τα εργαλεία και τα APIs σε όλες τις πλατφόρμες. Τα Middleware και οι μηχανές παιχνιδιών φέρνουν τους κινδύνους lock-in και lock-out. Επίσης η ευαισθητοποίηση του καταναλωτή των διαθέσιμων παιχνιδιών του κινητού των είναι χαμηλή .

2.4.5 Κατανομή

Τα παιχνίδια κινητών μπορούν να διανεμηθούν σε έναν από τις τέσσερις τρόπους :

- Μέσω Air (OTA) ένα δυαδικό αρχείο παιχνιδιού (συνήθως BREW ή Java) παραδίδεται στην κινητή συσκευή μέσω ασύρματων δικτύων μεταφοράς .
- Μέσω παράλληλης φόρτωσης (Sideloaded), ένα δυαδικό αρχείο παιχνιδιού είναι φορτωμένο στο τηλέφωνο ενώ είναι συνδεδεμένο με έναν υπολογιστή, είτε μέσω καλωδίου USB ή Bluetooth .
- Μέσω προεγκατάστασης, ένα δυαδικό αρχείο παιχνιδιού είναι προεγκατεστημένο στη συσκευή από τον κατασκευαστή πρωτότυπου εξοπλισμού (OEM) .
- Μέσω κατεβασμένου προγράμματος περιήγησης στο κινητό, ένα αρχείο του παιχνιδιού (συνήθως Adobe Flash Lite) έχει ληφθεί άμεσα από μια ιστοσελίδα για κινητό.

Μετά την έναρξη του Apple App Store και των κινητών πλατφόρμων OS όπως η Apple iOS , το Google Android και Microsoft Windows Mobile 7, οι προγραμματιστές κινητών OS δημιούργησαν ψηφιακές βιτρίνες λήψης που μπορούν να τρέξουν σε συσκευές που χρησιμοποιούν το λειτουργικό σύστημα ή το λογισμικό που χρησιμοποιείται στα PCs . Αυτές οι βιτρίνες (όπως της Apple iOS App Store) ενεργούν ως κεντρική ψηφιακή λήψη υπηρεσιών από την οποία μπορούμε να κατεβάσουμε μια ποικιλία μέσων ψυχαγωγίας και λογισμικού, συμπεριλαμβανομένων των παιχνιδιών όπου η πλειοψηφία των παιχνιδιών διατίθεται μέσω αυτών.

Η δημοτικότητα των mobile games αυξήθηκε τη δεκαετία του 2000, καθώς το 2007 πουλήθηκαν παιχνίδια αξίας περίπου 3.000.000.000 δολαριών ΗΠΑ με προβλεπόμενη ετήσια

αύξηση άνω του 40%. Η κυριότητα ενός smartphone αυξάνει την πιθανότητα του καταναλωτή να παίξει παιχνίδια για κινητά τηλέφωνα. Πάνω από το 90 % των χρηστών smartphone παίζουν ένα παιχνίδι κινητού, τουλάχιστον μία φορά την εβδομάδα.

Πολλά παιχνίδια κινητών που διανέμονται δωρεάν για τον χρήστη, αλλά μεταφέρουν πληρωμένη διαφήμιση : παραδείγματα είναι Flappy Bird και Candy Crush Saga . Ακολουθεί το « freemium » μοντέλο, στο οποίο το βασικό παιχνίδι είναι δωρεάν, αλλά πρόσθετα στοιχεία για το παιχνίδι μπορούν να αγοραστούν χωριστά από το χρήστη.

2.5 Μηχανές παιχνιδιού

Μια μηχανή του παιχνιδιού είναι ένα σύστημα σχεδιασμένο για τη δημιουργία και την ανάπτυξη των video games. Οι κορυφαίες μηχανές παιχνιδιών παρέχουν ένα πλαίσιο λογισμικού που χρησιμοποιούν οι προγραμματιστές για να δημιουργήσουν παιχνίδια για κονσόλες παιχνιδιών, κινητά τηλέφωνα και προσωπικούς υπολογιστές. Η βασική λειτουργικότητα που τυπικά προέρχεται από μια μηχανή παιχνιδιού περιλαμβάνει ένα μηχανισμό απόδοσης ("renderer") για 2D ή 3D γραφικά, μια νέα μηχανή φυσικής, ανίχνευση σύγκρουσης, ήχο, scripting, animation, τεχνητή νοημοσύνη, δικτύωση, streaming, μνήμη διαχείρισης, threading, υποστήριξη τοπικοποίησης, και ένα γράφημα σκηνής. Η διαδικασία της ανάπτυξης παιχνιδιού συχνά γίνεται με την επαναχρησιμοποίηση/προσαρμογή της ίδιας μηχανής παιχνιδιού για τη δημιουργία διαφορετικών παιχνιδιών, για να είναι πιο εύκολη η υποστήριξη παιχνιδιών σε πολλαπλές πλατφόρμες.

2.5.1 Σκοπός

Σε πολλές περιπτώσεις, οι μηχανές παιχνιδιών παρέχουν μια σειρά από οπτικά εργαλεία ανάπτυξης παράλληλα με επαναχρησιμοποιήσιμα στοιχεία λογισμικού. Αυτά τα εργαλεία παρέχονται γενικά σε ένα ολοκληρωμένο περιβάλλον ανάπτυξης, ώστε να απλοποιηθεί η ταχεία ανάπτυξη των παιχνιδιών, με τρόπο που βασίζεται σε δεδομένα. Οι προγραμματιστές των μηχανών του παιχνιδιού προσπαθούν να «προ - εφεύρουν τον τροχό " με την ανάπτυξη ισχυρών λογισμικών που περιλαμβάνουν πολλά στοιχεία που μπορεί να χρειαστεί ένας προγραμματιστής παιχνιδιού για να φτιάξει ένα παιχνίδι. Οι περισσότερες πλατφόρμες μηχανών παιχνιδιού παρέχουν διεργασίες, που διευκολύνουν την ανάπτυξη, όπως τα γραφικά, τον ήχο, τη φυσική και AI λειτουργίες. Αυτές οι μηχανές παιχνιδιών μερικές φορές ονομάζονται « middleware », επειδή παρέχουν μια ευέλικτη και επαναχρησιμοποιήσιμη πλατφόρμα λογισμικού η οποία παρέχει όλες τις βασικές λειτουργίες που απαιτούνται για την ανάπτυξη μιας εφαρμογής παιχνιδιού, μειώνοντας ταυτόχρονα το κόστος, τη πολυπλοκότητα και το χρόνο διάθεσης στην αγορά, δηλαδή όλοι οι κρίσιμοι παράγοντες στην άκρως

ανταγωνιστική βιομηχανία βιντεοπαιχνιδιών. Οι μηχανές Gamebryo, JMonkey και RenderWare είναι ευρέως χρησιμοποιούμενα προγράμματα middleware.

Όπως και άλλες λύσεις middleware, οι μηχανές παιχνιδιών παρέχουν συνήθως πλατφόρμα άντλησης, επιτρέποντας το ίδιο παιχνίδι να τρέχει σε διάφορες πλατφόρμες, συμπεριλαμβανομένων κονσόλες παιχνιδιών και προσωπικούς υπολογιστές με λίγες αλλαγές στον πηγαίο κώδικα του παιχνιδιού, εάν υπάρχουν. Συχνά οι μηχανές παιχνιδιού σχεδιάζονται με μια component-based αρχιτεκτονική που επιτρέπει σε συγκεκριμένα συστήματα της μηχανής να αντικατασταθούν ή να παραταθούν με πιο εξειδικευμένα (και συχνά πιο ακριβά) συστατικά middleware παιχνιδιού όπως το Havok για τη Φυσική, το σύστημα ήχου Miles για τον ήχο, ή το Bink για το βίντεο. Μερικές μηχανές παιχνιδιών, όπως η RenderWare, έχουν σχεδιαστεί ως μια σειρά από χαλαρά συνδεδεμένα συστατικά middleware παιχνιδιού που μπορούν να συνδυαστούν επιλεκτικά για να δημιουργήσουν μια προσαρμοσμένη μηχανή, αντί της πιο κοινής προσέγγισης της επέκτασης ή της προσαρμογής μιας ευέλικτης ολοκληρωμένης λύσης. Η επεκτασιμότητα αποτελεί υψηλή προτεραιότητας για τις μηχανές παιχνιδιού λόγω της μεγάλης ποικιλίας των χρήσεων για τις οποίες εφαρμόζονται. Ο μηχανές παιχνιδιών συχνά χρησιμοποιούνται για άλλα είδη διαδραστικών εφαρμογών με γραφικές ανάγκες πραγματικού χρόνου, όπως demos μάρκετινγκ, αρχιτεκτονικές απεικονίσεις , προσομοιώσεις εκπαίδευσης, και περιβάλλοντα μοντελοποίησης.

Μερικές μηχανές παιχνιδιών παρέχουν μόνο δυνατότητες real-time 3D rendering αντί του ευρέος φάσματος των λειτουργιών που απαιτούνται από τα παιχνίδια. Οι μηχανές αυτές βασίζονται στους προγραμματιστές για να αναπτύξουν το υπόλοιπο της λειτουργικότητας ή να να συγκεντρώσουν άλλα συστατικά middleware παιχνιδιού. Αυτοί οι τύποι μηχανών γενικά αναφέρονται ως " μηχανές γραφικών ", " μηχανές rendering , " ή " 3D μηχανές " αντί του πιο γενικού όρου " μηχανές παιχνιδιού ". Αυτή η ορολογία χρησιμοποιείται σε πολλές μηχανές 3D παιχνιδιών με πλήρεις δυνατότητες, οι οποίες αναφέρονται απλώς ως " μηχανές 3D ". Μερικά παραδείγματα των μηχανών γραφικών είναι : Crystal Space , Genesis3D , Irrlicht , OGRE , Realmforge , Truevision3D και Vision Engine. Σύγχρονες μηχανές παιχνιδιού ή γραφικών παρέχουν γενικά ένα γράφημα σκηνής, το οποίο είναι μια αντικειμενοστραφής αναπαράσταση του 3D κόσμου του παιχνιδιού που συχνά απλοποιεί το σχεδιασμό του παιχνιδιού και μπορεί να χρησιμοποιηθεί για πιο αποτελεσματική παροχή των τεράστιων εικονικών κόσμων.

2.5.2 Hardware abstraction

Τις περισσότερες φορές, οι μηχανές 3D ή τα συστήματα απόδοσης στις μηχανές παιχνιδιών βασίζονται σε ένα περιβάλλον προγραμματισμού εφαρμογών γραφικών (API - Application Programming Interface), όπως το Direct3D ή το OpenGL τα οποία παρέχουν μια αφαίρεση του λογισμικού της μονάδας επεξεργασίας γραφικών (GPU) ή της κάρτας βίντεο. Βιβλιοθήκες χαμηλού επιπέδου, όπως το DirectX, το Simple Layer DirectMedia (SDL), και το

OpenAL, επίσης χρησιμοποιούνται ευρέως στα παιχνίδια καθώς παρέχουν ανεξάρτητη πρόσβαση του hardware σε άλλο hardware του υπολογιστή, όπως σε συσκευές εισόδου (ποντίκι, πληκτρολόγιο και joystick), κάρτες δικτύου, και κάρτες ήχου. Πριν την ανάπτυξη του hardware σε 3D γραφικά , είχαν χρησιμοποιηθεί renderers λογισμικού. Το λογισμικό rendering εξακολουθεί να χρησιμοποιείται σε ορισμένα εργαλεία μοντελοποίησης ή rendered εικόνες όταν η οπτική ακρίβεια ετκιμάται σε επιδόσεις πραγματικού χρόνου (καρέ ανά δευτερόλεπτο) ή όταν το υλικό του υπολογιστή δεν ικανοποιεί τις ανάγκες, όπως την υποστήριξη shader .

Με την έλευση του hardware accelerated στην επεξεργασία φυσικής, διάφορα APIs φυσικής, όπως το PAL και τις προεκτάσεις της φυσικής COLLADA, έγιναν διαθέσιμα για να παρέχουν μια αφαίρεση του λογισμικού της μονάδας επεξεργασίας φυσικής των διαφορετικών παρόχων middleware και των πλατφόρμων της κονσόλας. Οι μηχανές παιχνιδιών μπορούν να γραφτούν σε οποιαδήποτε γλώσσα προγραμματισμού όπως η C + + , C ή Java .

2.5.3 Πρόσφατες τάσεις

Καθώς η τεχνολογία της μηχανής του παιχνιδιού ωριμάζει και γίνεται πιο φιλική προς το χρήστη, η εφαρμογή των μηχανών παιχνιδιών έχει διευρυνθεί. Τώρα χρησιμοποιούνται για σοβαρά παιχνίδια: οπτικοποίηση, εκπαίδευση, ιατρική, και στρατιωτικές εφαρμογές προσομοίωσης. Για να διευκολυνθεί αυτή η προσβασιμότητα, νέες πλατφόρμες hardware είναι πλέον στο στόχαστρο από τις μηχανές παιχνιδιών, συμπεριλαμβανομένων των κινητών τηλεφώνων (π.χ. κινητά Android, iPhone) και web browsers (π.χ. WebGL, Shockwave, Flash, WebVision Trinigy, το Silverlight, Unity Web Player, O3D και καθαρό DHTML).

Επιπλέον, οι περισσότερες μηχανές παιχνιδιού χτίζονται σε υψηλότερου επιπέδου γλώσσες, όπως η Java και η C#/.NET (π.χ. TorqueX και Visual3D.NET) ή Python (Panda3D). Δεδομένου ότι τα περισσότερα εμπλουτισμένα 3D παιχνίδια είναι πλέον ως επί το πλείστον GPU περιορισμένα, δηλαδή περιορίζονται από την ισχύ της κάρτας γραφικών, η πιθανή επιβράδυνση λόγω της μετάφρασης των γενικών εξόδων της υψηλότερου επιπέδου γλώσσας καθίσταται αμελητέα, ενώ τα κέρδη παραγωγικότητας που προσφέρονται από αυτές τις γλώσσες λειτουργούν προς όφελος των προγραμματιστών της μηχανής του παιχνιδιού. Αυτές οι πρόσφατες τάσεις προωθούνται από εταιρείες όπως η Microsoft για την υποστήριξη Indie ανάπτυξη παιχνιδιού. Η Microsoft ανέπτυξε το XNA ως το SDK της επιλογής για όλα τα παιχνίδια βίντεο που κυκλοφόρησαν για το Xbox και τα συναφή προϊόντα. Καθίσταται ευκολότερη και φθηνότερη από ποτέ η αναπτύξη μηχανών παιχνιδιών για πλατφόρμες που υποστηρίζουν τη διαχείριση πλαίσιων-frameworks.

2.5.4 Middleware παιχνιδιού

Με την ευρύτερη έννοια του όρου, οι μηχανές παιχνιδιού μπορούν να περιγραφούν ως middleware. Στο πλαίσιο των βιντεοπαιχνιδιών όμως, ο όρος "middleware" συχνά χρησιμοποιείται για να δηλώσει τα υποσυστήματα της λειτουργικότητας εντός μιας μηχανής παιχνιδιού. Μερικά middleware παιχνίδια κάνουν μόνο ένα πράγμα, αλλά το κάνουν πιο πειστικά ή πιο αποτελεσματικά από ό, τι κάνει γενικά ένα middleware.

Τα τέσσερα πιο ευρέως χρησιμοποιούμενα προγράμματα middleware που παρέχουν τα υποσυστήματα της λειτουργικότητας περιλαμβάνουν RAD Game Tools' Bink, Firelight FMOD , Havok και Scaleform GFX. Τα RAD Game Tools ανέπτυξαν το Bink για τη βασική απόδοση του βίντεο, μαζί με τον Miles ήχου, και το Granny 3D rendering. Το Firelight FMOD είναι μια χαμηλού κόστους ισχυρή ακουστική βιβλιοθήκη και ένα σύνολο εργαλείων. Το Havok παρέχει ένα ισχυρό σύστημα προσομοίωσης της φυσικής, μαζί με μια σειρά από κινούμενα σχέδια και τρόπους συμπεριφορά. Το Scaleform παρέχει GFX για υψηλή απόδοση Flash UI, μαζί με ένα τρόπο υψηλής ποιότητας αναπαραγωγής βίντεο, και ένα Input Method Editor (IME) add-on για την υποστήριξη συνομιλίας- chat της Ασίας στο παιχνίδι.

Μερικά middleware περιέχουν πλήρη πηγαίο κώδικα, και άλλα παρέχουν μόνο μια αναφορά API από τη μεταγλωττισμένη δυαδική βιβλιοθήκη. Μερικά προγράμματα middleware μπορούν να λάβουν άδεια, για να έχουν πλήρη τον πηγαίο κώδικα .

2.5.5 First-person shooter engines

Ένα πολύ γνωστό υποσύνολο των μηχανών παιχνιδιών είναι οι 3D first-person shooter (FPS) μηχανές παιχνιδιών. Η πρωτοποριακή ανάπτυξη από την άποψη της ποιότητας της εικόνας γίνεται σε FPS παιχνίδια με την ανθρώπινη κλίμακα. Ενώ τα παιχνίδια της προσομοίωσης πτήσης και οδήγησης και στρατηγικής σε πραγματικό χρόνο (RTS) παρέχουν ρεαλισμό σε μεγάλη κλίμακα, οι first-person shooters είναι στην πρώτη γραμμή των γραφικών του υπολογιστή σε μικρότερες κλίμακες. Η ανάπτυξη των γραφικών των FPS μηχανών που εμφανίζονται στα παιχνίδια μπορεί να χαρακτηρίζοναι από μια σταθερή αύξηση στον τομέα των τεχνολογιών, με κάποιες σημαντικές ανακαλύψεις.

2.6 Graphics pipeline (Αγωγός γραφικών)

Ο αγωγός γραφικών ή ο rendering αγωγός αναφέρεται στην αλληλουχία των σταδίων που χρησιμοποιούνται για τη δημιουργία μιας 2D raster αναπαράστασης μιας 3D σκηνής. Αφού δημιουργήσουμε ένα 3D μοντέλο, για παράδειγμα σε ένα βίντεοπαιχνίδι, ή οποιοδήποτε άλλα 3d κινούμενα σχέδια υπολογιστή, ο αγωγός γραφικών είναι η διαδικασία της μετατροπής

αυτού του 3D μοντέλου σε αυτό που φαίνεται στην οθόνη του υπολογιστή. Στις αρχές τα 3D γραφικά υπολογιστών hardware χρησιμοποιήθηκαν για να επιταχύνουν τα βήματα του αγωγού μέσω ενός αγωγού σταθερής λειτουργίας, αλλά το hardware εξελίχθηκε και έγινε γενικότερου σκοπού, επιτρέποντας μεγαλύτερη ευελιξία στην απόδοση των γραφικών, καθώς το πιο γενικευμένο hardware επιτρέπει το ίδιο hardware να εκτελέσει όχι μόνο διαφορετικά στάδια του αγωγού, σε αντίθεση με το σταθερό σκοπό του hardware, αλλά ακόμη και περιορισμένες μορφές γενικού σκοπού computing. Καθώς το hardware εξελίσσεται, το ίδιο κάνουν και οι αγωγοί γραφικών, η OpenGL και η DirectX αγωγοί, αλλά η γενική ιδέα του αγωγού παραμένει η ίδια.

2.6.1 Έννοια

Ο 3d αγωγός αναφέρεται συνήθως στην πιο κοινή μορφή του 3d υπολογιστή rendering, Η 3D απόδοση πολυγώνου είναι διακριτή από raytracing και raycasting. Ειδικότερα το 3D rendering πολυγώνου είναι παρόμοιο με raycasting. Στο raycasting μια ακτίνα προέρχεται από το σημείο όπου βρίσκεται η κάμερα, εάν η ακτίνα χτυπά μια επιφάνεια, τότε υπολογίζεται το χρώμα και ο φωτισμός του σημείου επί της επιφανείας όπου χτύπησαν οι ακτίνες. Στο 3d rendering πολυγώνου θα συμβεί το αντίστροφο, η περιοχή που είναι κατά την άποψη της κάμερας υπολογίζεται, και στη συνέχεια οι ακτίνες δημιουργούνται από κάθε μέρος της κάθε επιφάνειας προβολής της κάμερας και εντοπίζονται πίσω από την κάμερα.

2.6.2 Προγραμματιζόμενος αγωγός γραφικών

Οι υπολογιστές άρχισαν να δέχονται σημαντικές αλλαγές τα τελευταία χρόνια με την εισαγωγή μιας ξεχωριστής κάρτας βίντεο και την άνοδο του hardware accelerated γραφικών. Αυτό έχει οδηγήσει στην ανάγκη για ένα προγραμματιζόμενο αγωγό γραφικών που μπορεί να χειραγωγηθεί από Shaders. Με την εισαγωγή του προγραμματιζόμενου αγωγού γραφικών εφαρμογών αγωγού πιο σταθερές λειτουργίες έχουν καταστεί άνευ αντικειμένου, όπως η άμεση λειτουργία της OpenGL ή Direct3D ενσωματωμένη στο hardware Transform, clipping, και ο φωτισμός.

2.6.3 Στάδια του αγωγού γραφικών

Η σκηνή έχει δημιουργηθεί από τρισιδάστατα γεωμετρικά αρχέτυπα. Συνήθως αυτό γίνεται με τη χρήση τριγώνων, τα οποία είναι κατάλληλα καθώς αυτά ορίζονται πάντοτε σε ένα επίπεδο. Η μοντελοποίση και η μετατροπή είναι ο μετασχηματισμός από το τοπικό σύστημα συντεταγμένων στο 3d παγκόσμιο σύστημα συντεταγμένων. Ο μετασχηματισμός κάμερας είναι ο μετασχηματισμός του συστήματος συντεταγμένων του 3d κόσμου σε 3d κάμερα σύστηματος συντεταγμένων, με τη φωτογραφική μηχανή, όπως την αρχική. Φωτίζουμε σύμφωνα με το φωτισμό και την ανάκλαση, και υπολογίζονται η επίδραση του φωτισμού και των αντανακλάσεων. Ο μετασχηματισμός προβολής γίνεται μεταμορφώνοντας τις συντεταγμένες του 3d κόσμου στη 2d θέα της φωτογραφικής μηχανής. Στην περίπτωση μιας

Perspective προβολής, τα αντικείμενα που είναι απομακρυσμένα από την κάμερα γίνονται μικρότερα. Αυτό επιτυγχάνεται διαιρώντας τις συντεταγμένες X και Y της κάθε κορυφής του κάθε primitive από τις Z συντεταγμένες του, οι οποίες αντιπροσωπεύει την απόστασή του από την κάμερα. Σε μια orthographic προβολή, τα αντικείμενα διατηρούν το αρχικό τους μέγεθος ανεξάρτητα από την απόσταση από την κάμερα. Κατά τη διαδικασία Clipping τα γεωμετρικά αντικείμενα που πέφτουν εντελώς έξω από το viewing frustum δεν θα είναι ορατά και απορρίπτονται.

Rasterization ή σάρωση μετατροπής είναι η διαδικασία με την οποία η εικόνα 2D, η οποία αναπαραστάται στο χώρο της σκηνής, μετατρέπεται σε μορφή raster, καθορίζοντας τις σωστές τιμές pixel που προκύπτον. Από τώρα και στο εξής, οι εργασίες θα πρέπει να πραγματοποιούνται σε κάθε pixel. Αυτό το στάδιο είναι μάλλον πολύπλοκο, καθώς περιλαμβάνει πολλαπλά βήματα που συχνά αναφέρονται ως μια ομάδα με το όνομα του αγωγού εικονοστοιχείων (pixel pipeline).

Κατά την ύφανση (Texturing) τα επιμέρους θραυσμάτα(fragments) του αγωγού (ή προ-pixel) έχουν ένα χρώμα που βασίζεται σε τιμές που παρεμβάλλονται από τις κορυφές κατά τη διάρκεια του rasterization, από την υφή στη μνήμη, ή από ένα πρόγραμμα shader.

2.6.4 Ο αγωγός γραφικών σε hardware

Ο rendering pipeline χαρτογραφείται πάνω σε τρέχον hardware επιτάχυνσης γραφικών, έτσι ώστε η είσοδος στο GPU να είναι υπό τη μορφή των κορυφών. Αυτές οι κορυφές στη συνέχεια υφίστανται μετασχηματισμό και φωτισμό ανά κορυφή. Σε αυτό το σημείο στους σύγχρονους αγωγούς GPU ένα προσαρμοσμένο πρόγραμμα shader κορυφών μπορεί να χρησιμοποιηθεί για να χειραγωγηθούν οι 3D κορυφές πριν την ραστεροποίηση(rasterization) . Μόλις μετατραπούν και ανάψουν, οι κορυφές δέχονται κούρεμα και rasterization έχοντας ως αποτέλεσμα τα θραύσματα. Ένα δεύτερο προσαρμοσμένο πρόγραμμα shader μπορεί στη συνέχεια να τρέξει σε κάθε θραύσμα, πριν οι τελικές τιμές των πίξελ να είναι η έξοδος στο ρυθμιστικό πλαίσιο για την οθόνη .

Ο αγωγός γραφικών είναι κατάλληλος για τη διαδικασία απόδοσης επειδή επιτρέπει στη GPU να λειτουργεί σαν επεξεργαστής ρεύματος αφού όλες οι κορυφές και τα θραύσματα μπορούν να θεωρηθούν ως ανεξάρτητα. Αυτό επιτρέπει σε όλα τα στάδια του αγωγού να χρησιμοποιούνται ταυτόχρονα για διαφορετικές κορυφές ή θραύσματα καθώς επεξεργάζονται το δρόμο τους μέσα από το σωλήνα. Εκτός από τις κορυφές διασωλήνωσης και τα θραύσματα , η ανεξαρτησία των επεξεργαστών γραφικών επιτρέπει να χρησιμοποιούν παράλληλες μονάδες επεξεργασίας για την επεξεργασία πολλαπλών κορυφών ή θραυσμάτων σε ένα μόνο στάδιο του αγωγού την ίδια στιγμή.

2.7 Διεπαφές γραφικών (3D APIs)

Όπως αναφέραμε παραπάνω, τις περισσότερες φορές οι μηχανές 3D ή τα συστήματα απόδοσης στις μηχανές παιχνιδιού βασίζονται σε ένα περιβάλλον προγραμματισμού εφαρμογών γραφικών (API), όπως το Direct3D ή το OpenGL τα οποία παρέχουν μια αφαίρεση του λογισμικού της μονάδας επεξεργασίας γραφικών (GPU) ή της κάρτας βίντεο. Αυτά τα αναλύουμε παρακάτω, καθώς και η Unity χρησιμοποιεί ως μηχανή γραφικών τη Direct3D (για τα Windows, Xbox 360), την OpenGL (για τα Mac, Windows, Linux), την OpenGL ES (για τα Android, iOS), και την API (για τις κονσόλες).

2.7.1 Διεπαφή προγραμματισμού εφαρμογών (Application programming interface - API)

Μια διεπαφή προγραμματισμού εφαρμογών (API) καθορίζει πώς ορισμένα στοιχεία του λογισμικού θα πρέπει να αλληλεπιδρούν μεταξύ τους . Εκτός από την πρόσβαση σε βάσεις δεδομένων ή υλικού υπολογιστών, όπως σκληρούς δίσκους ή κάρτες γραφικών, μια API μπορεί να χρησιμοποιηθεί για να διευκολύνει το έργο των στοιχείων του περιβάλλοντος της εργασίας του χρήστη με τον προγραμματισμό των γραφικών. Στην πράξη πολλές φορές μια API έρχεται με τη μορφή μιας βιβλιοθήκης που περιλαμβάνει προδιαγραφές για ρουτίνες, δομές δεδομένων, κλάσεις αντικειμένων, και μεταβλητές. Σε ορισμένες άλλες περιπτώσεις, ιδίως για SOAP και REST υπηρεσίες , μια API έρχεται απλά ως μια προδιαγραφή απομακρυσμένων κλήσεων που εκτίθονται στους καταναλωτές API. Μια API διαφέρει από μια εφαρμογή δυαδικής διεπαφής (ABI) κατά το ότι μια API είναι πηγαίος κώδικας με βάση ενώ ένα ABI είναι ένα δυαδικό περιβάλλον.

2.7.2 Direct3D

Η Direct3D είναι μέρος της διεπαφής προγραμματισμού εφαρμογών DirectX της Microsoft (API). Είναι η βάση για τους φορείς γραφικών API για τις κονσόλες συστημάτων Xbox και Xbox 360. Η Direct3D χρησιμοποιείται για την απόδοση τρισδιάστατων γραφικών σε εφαρμογές όπου η απόδοση είναι σημαντική, όπως τα παιχνίδια. Επιτρέπει επίσης στις εφαρμογές να τρέχουν σε πλήρη οθόνη αντί σε ένα ενσωματωμένο παράθυρο, αν και μπορεί να τρέξει και σε ένα παράθυρο. Η Direct3D χρησιμοποιεί επιτάχυνση υλικού, εάν είναι διαθέσιμη στην κάρτα γραφικών, επιτρέποντας την επιτάχυνση υλικού του συνόλου του αγωγού 3D rendering ή ακόμη και μερική μόνο επιτάχυνση. Η Direct3D εκθέτει τις προηγμένες δυνατότητες γραφικών των 3D γραφικών hardware, συμπεριλαμβανομένων των z - buffering , χωροταξικού anti-aliasing, alpha blending, mipmapping, ατμοσφαιρικών επιδράσεων και προοπτικής σωστής απεικόνισης υφής. Η ενοποίηση με άλλες τεχνολογίες DirectX επιτρέπει στην Direct3D να αποδώσει τα παραπάνω χαρακτηριστικά, όπως η χαρτογράφηση βίντεο, το

3D rendering υλικού σε 2D επίπεδα επικάλυψης, ακόμη και sprites, παρέχοντας τη χρήση των 2D και 3D γραφικών σε διαδραστικές σχέσεις των μέσων ενημέρωσης .

Η Direct3D είναι μια 3D API, δηλαδή περιέχει πολλές εντολές για 3D rendering. Προσφέρει πλήρη εξομοίωση λογισμικού κορυφής, αλλά όχι εξομοίωση λογισμικού pixel γιατί τα χαρακτηριστικά δεν είναι διαθέσιμα σε hardware. Η API δεν περιλαμβάνει Rasterizer αναφοράς (REF ή συσκευή) , το οποίο μιμείται μια γενική κάρτα γραφικών στο λογισμικό, αν και είναι πολύ αργή για τις περισσότερες εφαρμογές 3D σε πραγματικό χρόνο και συνήθως χρησιμοποιούνται μόνο για debugging. Μια νέα rasterizer λογισμικού σε πραγματικό χρόνο, WARP, μιμείται το πλήρες σύνολο των δυνατοτήτων της Direct3D 10.1, της οποίας οι επιδόσεις του ότι είναι στο ίδιο επίπεδο με τις lower-end 3D κάρτες σε multi - core επεξεργαστές. Ο κύριος ανταγωνιστής της Direct3D είναι η OpenGL.

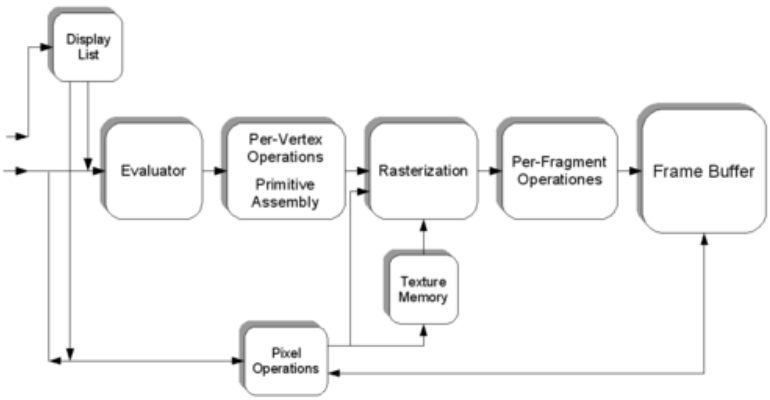
2.7.3 OpenGL

Η OpenGL (Open Graphics Library) είναι μια cross-γλώσσα, multi-platform διεπαφή προγραμματισμού εφαρμογών (API) για την απόδοση 2D και 3D διανυσματικών γραφικών. Η API συνήθως χρησιμοποιείται για να αλληλεπιδρούν με μια μονάδα επεξεργασίας γραφικών (GPU), για να επιτευχθεί επιτάχυνση του υλικού απόδοσης. Η OpenGL χρησιμοποιείται ευρέως σε CAD, εικονική πραγματικότητα, επιστημονική απεικόνιση, οπτικοποίηση πληροφοριών, προσομοίωση πτήσης, και βιντεοπαιχνίδια.

Η προδιαγραφή της OpenGL περιγράφει μια αφηρημένη API για την κατάρτιση 2D και 3D γραφικών, είναι σχεδιασμένη να εφαρμοστεί ως επί το πλείστον ή εξ ολοκλήρου στο hardware.

Η API ορίζεται ως μια σειρά από λειτουργίες που μπορούν να κληθούν από το πρόγραμμα client, μαζί με έναν αριθμό από τους σταθερούς ακεραίους αριθμούς, ανεξάρτητα της γλώσσας. Ως εκ τούτου η OpenGL έχει πολλές γλώσσες bindings, μερικές από τις πιο αξιοσημείωτες είναι η JavaScript δεσμευτική της WebGL (API, με βάση την OpenGL ES 2.0 για 3D rendering μέσα από ένα web browser), οι δεσμοί C WGL, GLX και ΚΕΓ, η C binding παρέχεται από το iOS, και τα Java και C bindings που προβλέπονται από το Android .

Η OpenGL είναι ανεξάρτητη γλώσσα και ανεξάρτητη από την πλατφόρμα, και ασχολείται με την απόδοση, παρέχοντας APIs που δεν σχετίζονται με την είσοδο , τον ήχο , ή τα παραθύρα.



Η OpenGL είναι μια εξελισσόμενη API. Εκτός από τα χαρακτηριστικά που απαιτούνται από τον πυρήνα API, οι πωλητές GPU μπορούν να παρέχουν πρόσθετη λειτουργικότητα με τη μορφή των επεκτάσεων. Οι επεκτάσεις μπορούν να εισαγάγουν νέες λειτουργίες και νέες σταθερές, και μπορούν να χαλαρώσουν ή να καταργήσουν τους περιορισμούς στις υπάρχουσες λειτουργίες OpenGL, το οποίο αυξάνει σημαντικά την ευελιξία της OpenGL. Όλες οι επεκτάσεις συλλέγονται και ορίζονται από το OpenGL Registry. Τα χαρακτηριστικά που εισάγονται από κάθε νέα έκδοση του OpenGL συνήθως σχηματίζονται από το συνδυασμό των διάφορων χαρακτηριστικών που εφαρμόζονται ευρέως σε επεκτάσεις, ιδιαίτερα οι επεκτάσεισ της έγκρισης τύπου ARB ή EXT .

2.7.4 OpengGL ES

Η OpenGL για ενσωματωμένα συστήματα (OpenGL ES ή GLES) είναι ένα υποσύνολο των γραφικών του υπολογιστή OpenGL που αποδίδει τη διεπαφή εφαρμογών προγραμματισμού (API) για την απόδοση 2D και 3D γραφικών του υπολογιστή, όπως αυτά που χρησιμοποιούνται από τα βιντεοπαιχνίδια, συνήθως με επιτάχυνση του hardware χρησιμοποιώντας μια μονάδα επεξεργασίας γραφικών (GPU). Είναι σχεδιασμένο για τα ενσωματωμένα συστήματα, όπως smartphones, ταμπλέτες υπολογιστών, κονσόλες βιντεοπαιχνιδιών και PDAs. Η API είναι διαγλωσσικό και multi-platform. Δεν υπάρχει καμία ισοδύναμη με την OpenGL βιβλιοθηκή, όπως η GLUT ή GLU για OpenGL ES.

2.7.5 Σύγκριση των OpenGL και Direct3D

Η Direct3D και η OpenGL είναι ανταγωνιστικές διεπαφές προγραμματισμού εφαρμογών (APIs) που μπορούν να χρησιμοποιηθούν σε εφαρμογές για να καταστήσουν 2D και 3D γραφικά υπολογιστών, με κοινό στόχο την επιτάχυνση του hardware. Οι GPUs που υποστηρίζουν πιο πρόσφατες εκδόσεις των προτύπων είναι συμβατές με τις εφαρμογές που χρησιμοποιούν τα παλαιότερα πρότυπα, για παράδειγμα μπορεί κανείς να τρέξει τα μεγαλύτερα DirectX 9 παιχνίδια σε μια πιο πρόσφατη DirectX 11 πιστοποιημένη-GPU.

Η Direct3D ανάπτυξη εφαρμογών στοχεύει συνήθως την πλατφόρμα Microsoft Windows. Η OpenGL API είναι ένα ανοιχτό πρότυπο, και υπάρχουν υλοποιήσεις για μια ευρεία ποικιλία από πλατφόρμες. Η Direct3D είναι μια ιδιόκτητο API που παρέχει λειτουργίες για να καταστήσει τρισδιάστατα γραφικά, και χρησιμοποιεί επιτάχυνση hardware, εάν είναι διαθέσιμη στην κάρτα γραφικών. Η OpenGL είναι μια ανοικτή πρότυπη API που παρέχει μια σειρά από λειτουργίες για να καταστήσει 2D και 3D γραφικά, και είναι διαθέσιμο στα περισσότερα σύγχρονα λειτουργικά συστήματα συμπεριλαμβανομένων, αλλά χωρίς να περιορίζονται, των Windows, Mac OS X και Linux. Μια σημαντική διαφορά όμως είναι ότι η Direct3D υλοποιεί την API σε μια κοινή runtime, η οποία με τη σειρά της συνομιλεί σε μια χαμηλού επιπέδου Device Driver Interface (DDI). Με την OpenGL κάθε προμηθευτής

εφαρμόζει το σύνολο API στο ίδιο το πρόγραμμα εμφάνισης. Αυτό σημαίνει ότι κάποιες λειτουργίες API μπορεί να έχουν ελαφρώς διαφορετική συμπεριφορά από το ένα στο άλλο.

Η ιδιόκτητη Direct3D είναι διαθέσιμη μόνο σε λειτουργικά συστημάτα της Microsoft, συμπεριλαμβανομένων των ενσωματωμένων εκδόσεων που χρησιμοποιούνται σε βιντεοπαιχνιδιών κονσόλων του Xbox και σε Dreamcast της Sega. Η OpenGL έχει εφαρμογές που διατίθονται σε πολλές πλατφόρμες, συμπεριλαμβανομένων των Microsoft Windows, συστήματα Unix-based, όπως το Mac OS X , GNU/Linux . Ένα υποσύνολο της OpenGL επιλέχθηκε ως η κύρια βιβλιοθήκη γραφικών για το Android, BlackBerry, iOS και Symbian με τη μορφή OpenGL ES .

2.7.6 Μονάδα επεξεργασίας γραφικών (Graphics processing unit - GPU)

Μια μονάδα επεξεργασίας γραφικών (GPU), που επίσης μερικές φορές ονομάζεται μονάδα οπτικής επεξεργασίας (VPU), είναι ένα εξειδικευμένο ηλεκτρονικό κύκλωμα σχεδιασμένο για να χειριστεί γρήγορα και να αλλάξει τη μνήμη για να επιταχύνει τη δημιουργία των εικόνων σε ένα ρυθμιστικό πλαίσιο που προορίζεται για την παραγωγή σε μια οθόνη. Οι GPUs χρησιμοποιούνται στα ενσωματωμένα συστήματα, κινητά τηλέφωνα, προσωπικούς υπολογιστές, σταθμούς εργασίας, και κονσόλες παιχνιδιών . Οι σύγχρονες GPUs είναι πολύ αποτελεσματικές στο χειρισμό γραφικών του υπολογιστή, και ιδιαίτερα η παράλληλη δομή τους τις καθιστά πιο αποτελεσματικές από ό,τι οι επεξεργαστές γενικής χρήσης για τους αλγόριθμους εφόσον η επεξεργασία των μεγάλων μπλοκ δεδομένων γίνεται παράλληλα. Σε έναν προσωπικό υπολογιστή, μια GPU μπορεί να είναι παρόν σε μια κάρτα βίντεο, ή μπορεί να είναι στη μητρική πλακέτα ή σε ορισμένους επεξεργαστές επί της CPU.

Τα σύγχρονα GPUs χρησιμοποιούν τα περισσότερα από τα τρανζίστορ τους για να κάνουν υπολογισμούς που σχετίζονται με 3D γραφικά υπολογιστών. Χρησιμοποιήθηκαν αρχικά για να επιταχύνουν τις εργασίες της χαρτογράφησης υφής και την απόδοση πολυγώνων που απαιτούν πολλή μνήμη, αργότερα για την επιτάχυνση γεωμετρικών υπολογισμών , όπως η περιστροφή και η μετάφραση των κορυφών σε διαφορετικά συστήματα συντεταγμένων. Οι πρόσφατες εξελίξεις σε GPUs περιλαμβάνουν υποστήριξη για προγραμματιζόμενα shaders που μπορούν να χειριστούν κορυφές και υφές με πολλές από τις ίδιες λειτουργίες που υποστηρίζονται από CPUs , υπερδειγματοληψία και παρεμβολή τεχνικών για τη μείωση aliasing, και πολύ υψηλής ακρίβειας χώρου χρώματος. Οι GPUs περιλαμβάνουν 3D hardware και τις βασικές 2D επιταχύνσεις και δυνατότητες framebuffer.

2.8 Όροι και έννοιες

2.8.1 Μηχανή γραφικών

Μια μηχανή γραφικών είναι ένα είδος προγράμματος υπολογιστή υπεύθυνο για την κατάρτιση των γραφικών ηλεκτρονικών υπολογιστών. Ο όρος μπορεί να αναφέρεται σε μηχανή του παιχνιδιού, ένα σύστημα σχεδιασμένο για τη δημιουργία και την ανάπτυξη των video games, όπως μια μηχανή γραφικών shooter πρώτου προσώπου, που είναι μια μηχανή παιχνιδιών βίντεο που ειδικεύεται στην προσομοίωση 3D περιβάλλοντων για χρήση σε ένα first-person shooter βίντεοπαιχνίδι. Επίσης μπορεί να αναφέρεται σε μια μηχανή γραφικών (hardware), μια εξειδικευμένη συσκευή υλικού του υπολογιστή για την εκτέλεση υπολογισμών γραφικών ανεξάρτητα από τον κύριο επεξεργαστή ενός υπολογιστή.

Τα 3D γραφικά έχουν γίνει τόσο δημοφιλοί, ιδιαίτερα στα βιντεοπαιχνίδια, που έχουν εξειδικευμένες APIs (διεπαφές προγραμματισμού εφαρμογών) που δημιουργήθηκαν για να διευκολύνουν τις διαδικασίες σε όλα τα στάδια της παραγωγής γραφικών ηλεκτρονικών υπολογιστών, και παρέχουν έναν τρόπο για τους προγραμματιστές να έχουν πρόσβαση στο hardware με έναν αφηρημένο τρόπο, ενώ εξακολουθούν να επωφελούνται από το ειδικό hardware κάθε συγκεκριμένης κάρτας γραφικών.

Από τις APIs για 3D γραφικά υπολογιστών χαμηλού επιπέδου είναι ιδιαίτερα δημοφιλοί η OpenGL, η OpenGL ES 3D API για φορητές συσκευές, και η Direct3D (ένα υποσύνολο του DirectX). Υπάρχουν και μηχανές που βασίζονται σε JavaScript. Το WebGL είναι μια διεπαφή Javascript για OpenGL ES API, που επιτρέπει στις εφαρμογές να χρησιμοποιούν τα μητρικά γραφικά. Υπάρχουν επίσης, υψηλότερου επίπεδου 3D APIs γραφικής παράστασης σκηνή που παρέχουν πρόσθετη λειτουργικότητα από το χαμηλότερου επιπέδου απόδοσης API. Επίσης υπάρχουν Flash-based engines, μηχανές 3D που έχουν αναπτυχθεί στην πλατφόρμα Adobe Flash για να τρέξουν σε web browsers.

2.8.2 Texture (υφές)

Η χαρτογράφηση υφής (Texture mapping) είναι μια μέθοδος για την προσθήκη λεπτομέρειας, επιφάνειας υφής (bitmap ή ράστερ εικόνας), ή χρώματος σε ένα γραφικό ή 3D μοντέλο υπολογιστή που δημιουργείται.

Ένας χάρτης υφής εφαρμόζεται και αντιστοιχίζεται προς την επιφάνεια ενός σχήματος ή ενός πολυγώνου. Σε κάθε κορυφή ενός πολυγώνου έχει εκχωρηθεί μια υφή συντεταγμένων, οι οποίες στην περίπτωση του 2d είναι επίσης γνωστές ως UV συντεταγμένων, είτε με ρητή εκχώρηση ή με διαδικαστικό ορισμό. Στη συνέχεια οι θέσεις δειγματοληψίας της εικόνας, με παρεμβολή σε όλη την επιφάνεια ενός πολυγώνου, παράγουν ένα οπτικό αποτέλεσμα που φαίνεται να έχει περισσότερο πλούτο από ό,τι θα μπορούσε διαφορετικά να επιτευχθεί με

έναν περιορισμένο αριθμό πολυγώνων. Multitexturing είναι η χρήση περισσότερων από μία υφές το χρόνο σε ένα πολύγωνο, όπως για να φωτίσει μια επιφάνεια ως εναλλακτική λύση για τον εκ νέου υπολογισμό του φωτισμού της επιφάνειας που χρειάζεται κάθε φορά. Μια άλλη τεχνική multitexture είναι η bump mapping, η οποία επιτρέπει μια υφή να ελέγχει άμεσα την επιφάνεια που έχει κατεύθυνση προς αυτήν για τον υπολογισμό του φωτισμού. Μπορεί να δώσει μια πολύ καλή εμφάνιση από μια σύνθετη επιφάνεια, που παίρνει από τη λεπτομέρεια του φωτισμού εκτός από το συνήθες λεπτομερή χρωματισμό. Το Bump mapping έγινε δημοφιλές στα τελευταία βίντεοπαιχνίδια ως hardware γραφικών και έχει γίνει αρκετά ισχυρό για να το δουλεύει σε πραγματικό χρόνο .

2.8.3 Φωτισμός

Ο φωτισμός των γραφικών των υπολογιστών αναφέρεται στην προσομοίωση του φωτός σε γραφικά υπολογιστών. Η προσομοίωση μπορεί να είναι εξαιρετικά ακριβής, ή μπορεί απλά εμπνευστεί από το φως της φυσικής, αλλά και στις δύο περιπτώσεις χρησιμοποιείται ένα μοντέλο σκίασης (shading model) για να περιγράψει τον τρόπο που οι επιφάνειες αντιδρούν στο φως. Ο φωτισμός γραφικών του υπολογιστή μπορεί επίσης να αναφέρεται στην εργασία του φωτισμού στο computer animation.

2.8.4 Lightmap

Ένα lightmap είναι μια δομή δεδομένων η οποία περιέχει τη φωτεινότητα των επιφανειών σε 3d εφαρμογές γραφικών, όπως τα βιντεοπαιχνίδια. Τα Lightmaps προ-υπολογίζονται και χρησιμοποιούνται κανονικά μόνο για στατικά αντικείμενα. Είναι ιδιαίτερα κατάλληλα για αστικούς και εσωτερικούς χώρους με μεγάλες επίπεδες επιφάνειες.

Κατά τη δημιουργία των lightmaps, οποιοδήποτε μοντέλο φωτισμού μπορεί να χρησιμοποιηθεί, επειδή ο φωτισμός είναι απολύτα προϋπολογισμένος και οι επιδόσεις σε πραγματικό χρόνο δεν είναι πάντα αναγκαίες. Τα σύγχρονα 3d πακέτα περιλαμβάνουν ειδικά plugins για την εφαρμογή χάρτη φωτός UV συντεταγμένων, χάρτη πολλαπλών επιφανειών σε μεμονωμένα φύλλα υφών, rendering των χαρτών και προσαρμοσμένα lightmap εργαλεία δημιουργίας. Η απαλότητα των σκιών καθορίζεται από το πόσο η μηχανή παρεμβάλλει τα δεδομένα Lumel σε μια επιφάνεια, και μπορεί να οδηγήσει σε μια pixelated ματιά εάν τα lumels είναι πολύ μεγάλα. Τα Lightmaps μπορούν επίσης να υπολογιστούν σε πραγματικό χρόνο για την καλή ποιότητα χρώματος εφέ φωτισμού που δεν είναι επιρρεπείς στις ατέλειες και η δημιουργία σκιάς πρέπει να γίνει με χρήση άλλης μεθόδου.

2.8.5 Σκιαστής (Shader)

Στον τομέα των γραφικών του υπολογιστή ένας shader είναι ένα πρόγραμμα υπολογιστή που χρησιμοποιείται για να κάνει σκίαση, δηλαδή παραγωγή των κατάλληλων

επιπέδων του φωτός και του χρώματος μέσα σε μια εικόνα, να δημιουργήσει ειδικά εφέ και να τα μετεπεξεργαστεί.

Οι Shaders υπολογίζουν τις επιπτώσεις του rendering στο hardware γραφικών με υψηλό βαθμό ευελιξίας. Οι περισσότεροι shaders κωδικοποιούνται για μια μονάδα επεξεργασίας γραφικών (GPU). Οι γλώσσες σκίασης συνήθως χρησιμοποιούνται για τον προγραμματισμό του αγωγού απόδοσης της GPU. Με τους shaders μπορούν να χρησιμοποιηθούν προσαρμοσμένα αποτελέσματα. Η θέση, η απόχρωση, ο κορεσμός, η φωτεινότητα, η ανίχνευση ακμών και κίνησης, και η αντίθεση όλων των pixels, των κορυφών, των υφών που χρησιμοποιούνται για την κατασκευή μιας τελικής εικόνας μπορούν να μεταβληθούν, χρησιμοποιώντας αλγόριθμους που ορίζονται στο shader, και μπορούν να τροποποιηθούν με εξωτερικές μεταβλητές ή υφές που εισάγει το πρόγραμμα καλώντας τον shader .

2.8.6 Computer animation

Animation υπολογιστή ή CGI animation είναι η διαδικασία που χρησιμοποιείται για τη δημιουργία κινούμενων εικόνων με τη χρήση γραφικών ηλεκτρονικών υπολογιστών. Ο πιο γενικός όρος περιλαμβάνει στατικές σκηνές και δυναμικές εικόνες, ενώ το animation υπολογιστών αναφέρεται μόνο σε κινούμενες εικόνες. Το σύγχρονο animation υπολογιστών συνήθως χρησιμοποιεί 3D γραφικά υπολογιστών, χαμηλό εύρος ζώνης, και ταχύτερες απεικονίσεις σε πραγματικό χρόνο. Μερικές φορές ο στόχος του animation είναι ο ίδιος ο υπολογιστής, αλλά μερικές φορές ο στόχος είναι ένα άλλο μέσο, όπως φιλμ.

Το Animation υπολογιστών είναι ουσιαστικά ένας ψηφιακός διάδοχος με τεχνικές στάσεις κίνησης που χρησιμοποιούνται στο παραδοσιακό animation με 3D μοντέλα και καρέ καρέ animation με 2D εικόνες. Τα τρισδιάστατα κινούμενα σχέδια είναι περισσότερο ελεγχόμενα από άλλες διαδικασίες με πιο φυσική προσέγγιση, επειδή επιτρέπει τη δημιουργία των εικόνων που δεν θα ήταν εφικτές με οποιαδήποτε άλλη τεχνολογία. Για να δημιουργήσουμε την ψευδαίσθηση της κίνησης, μια εικόνα εμφανίζεται στην οθόνη του υπολογιστή και επανειλημμένα αντικαταστάται από μια νέα εικόνα που είναι παρόμοια με αυτή, αλλά με ελαφριά πρόοδο του χρόνου (συνήθως σε ποσοστό 24 ή 30 καρέ/δευτερόλεπτο). Αυτή η τεχνική είναι πανομοιότυπη με το πώς η ψευδαίσθηση της κίνησης επιτυγχάνεται με την τηλεόραση και την κίνηση εικόνων .

Για 3D animations, τα αντικείμενα (μοντέλα) δημιουργούνται στο monitor του υπολογιστή (πρότυπο) και οι 3D φιγούρες στήνονται (rigged) με ένα εικονικό σκελετό. Για 2D σχήματα κινουμένων σχεδίων, τα ξεχωριστά αντικείμενα (εικόνες) και ξεχωριστά διαφανή στρώματα χρησιμοποιούνται με ή χωρίς ένα εικονικό σκελετό. Στη συνέχεια τα άκρα, τα μάτια, το στόμα, τα ρούχα, κλπ. του σχήματος κινούνται από τον γραφίστα σε βασικά καρέ. Οι διαφορές στην εμφάνιση μεταξύ των βασικών πλαισίων υπολογίζεται αυτόματα από τον

υπολογιστή σε μια διαδικασία γνωστή ως tweening ή μορφοποίηση. Στο τέλος αποδίδεται η κινούμενη εικόνα.

Για 3D animations όλα τα πλαίσια πρέπει να αποδίδονται μετά την πλήρη μοντελοποίηση. Τα πλαίσια μπορούν επίσης να εμφανίζονται σε πραγματικό χρόνο, όπως αυτά παρουσιάζονται στο κοινό του τελικού χρήστη. Τα animations χαμηλού εύρους ζώνης που μεταδίδονται μέσω του διαδικτύου (π.χ. 2D Flash , X3D) χρησιμοποιούν συχνά το λογισμικό στον υπολογιστή των τελικών χρηστών για να αποδωθούν σε πραγματικό χρόνο.

2.8.7 Meshes

Ένα πλέγμα πολυγώνου είναι μια συλλογή από κορυφές, ακμές και επιφάνειες που καθορίζουν το σχήμα ενός πολυεδρικού αντικειμένου σε 3D γραφικά υπολογιστών και στερεά μοντελοποίηση. Τα πρόσωπα συνήθως αποτελούνται από τρίγωνα, τετράπλευρα ή άλλα απλά κυρτά πολυγώνα, καθώς έτσι απλοποιείται η απόδοση, αλλά μπορεί να αποτελείται από περισσότερα κοίλα πολύγωνα ή πολύγωνα με τρύπες.

Η μελέτη των πολυγωνικών πλεγμάτων είναι ένας μεγάλος υπο-τομέας των computer graphics και της γεωμετρικής μοντελοποίησης. Διαφορετικές αναπαραστάσεις πολυγωνικών πλεγμάτων χρησιμοποιούνται για διαφορετικές εφαρμογές και στόχους. Τα ογκομετρικά meshes είναι διαφορετικά από τα meshes πολυγώνων σε ότι αντιπροσωπεύουν ρητά τόσο την επιφάνεια και τον όγκο μιας δομής, ενώ το meshes πολυγώνου μόνο απεικονίζει την επιφάνεια. Δεδομένου ότι τα πολυγωνικά πλέγματα χρησιμοποιούνται ευρέως σε γραφικά ηλεκτρονικών υπολογιστών, υπάρχουν επίσης αλγόριθμοι για ray tracing, ανίχνευση σύγκρουσης, και πολυγωνικά πλέγματα δυναμικού άκαμπτου σώματος.

2.8.8 Φυσική παιχνιδιού

Η φυσική του computer animation ή η φυσική του παιχνιδιού περιλαμβάνει την εισαγωγή των νόμων της φυσικής σε μια μηχανή προσομοίωσης ή στη μηχανή του παιχνιδιού, ιδιαίτερα σε 3D γραφικά υπολογιστών, με σκοπό να καταστούν οι επιπτώσεις ώστε να φαίνονται πιο αληθινές στον παρατηρητή. Συνήθως η φυσική προσομοίωση είναι μόνο μια στενή προσέγγιση της πραγματικής φυσικής, και ο υπολογισμός γίνεται με τη χρήση διακριτών τιμών.

Υπάρχουν διάφορα στοιχεία που αποτελούν συστατικά της φυσικής προσομοίωσης, συμπεριλαμβανομένης της μηχανής φυσικής, το κώδικα του προγράμματος που χρησιμοποιείται για την προσομοίωση της νευτώνειας φυσικής στο περιβάλλον, και την ανίχνευση σύγκρουσης που χρησιμοποιείται για να λύσει το πρόβλημα του καθορισμού όταν υπάρχουν δύο ή περισσότερα φυσικά αντικείμενα στο περιβάλλον που συναντά το ένα το άλλο.

2.9 Κατηγορίες των μηχανών παιχνιδιών

Πολλά εργαλεία που ονομάζονται μηχανές παιχνιδιών είναι διαθέσιμα για τους σχεδιαστές παιχνιδιών για να κωδικοποιήσουν ένα παιχνίδι γρήγορα και εύκολα, χωρίς να χρειάζεται να το φτιάξουν από το μηδέν. Υπάρχουν πολλά διαθέσιμα εργαλεία που χρησιμοποιούνται για την ανάπτυξη παιχνίδιού πάνω σε διάφορες γλώσσες προγραμματισμού, και είναι σημαντικό να επιλεχθεί η κατάλληλη πλατφόρμα που θα το υποστηρίζει. Είναι προτιμότερο η εφαρμογή να είναι συμβατή με όσες περισσότερες πλατφόρμες το δυνατόν. Για το λόγο αυτόν εξετάζονται παρακάτω οι σημαντικότερες πλατφόρμες δημιουργίας παιχνιδιών ως προς τα χαρακτηριστικά τους, τις ικανότητές και τις διαφορές τους, ώστε να επιλεγεί η καταλληλότερη. Παρακάτω συγκρίνουμε μόνο μηχανές παιχνιδιού που υποστηρίζουν την ανάπτυξη τρισδιάστατου παιχνιδιού σε λειτουργικά κινητών συσκευών, καθώς αυτές μας ενδιαφέρουν στην περίπτωσή μας.

2.9.1 Μηχανές παιχνιδιού χωρίς χρήση προγραμματισμού

Υπάρχουν πλατφόρμες για τη δημιουργία παιχνιδιών σε κινητά χωρίς τη χρήση προγραμματισμού. Το αρνητικό όμως είναι ότι μειονεκτούν στην ευελιξία και απαιτείται η πληρωμή συνδρομής. Το σημαντικότερο όμως είναι ότι λόγω έλλεψης προγραμματισμού περιορίζονται οι δυνατότητες του παιχνιδιού, και για αυτό δεν επιλέχθηκε καμία από τις παρακάτω πλατφόρμες. Όμως επειδή είναι σημαντικές στον τομέα της δημιουργίας παιχνιδιών σε κινητά τις αναφέρουμε, παρόλο που οι περισσότερες δεν απαιτούν καμία γνώση προγραμματισμού.

- Construct 2 – για δημιουργία παιχνιδιού σε κινητά για τα Windows που επιτρέπει τη δημιουργία iOS, Android και Facebook παιχνίδια χωρίς γνώσεις προγραμματισμού. Χρησιμοποιεί απλό ‘event-based’ interface.
- Gamemaker - πλατφόρμα που χρησιμοποιεί μια drag and drop οπτική διεπαφή και επιτρέπει τη δημιουργία παιχνίδιών για Android και iOS. Χρησιμοποιείται για την κατασκευή iOS gamese Froad και Grave Maker.
- Stencyl - πλατφόρμα δημιουργίας παιχνιδιού για iOS και Flash και μελλοντική υποστήριξη από Android και HTML5. Διαθέτει οπτικό drag and drop interface.
- Multimedia Fusion 2 - επιτρέπει τη δημιουργία και παιχνιδιών και εφαρμογών για iOS, Android, Java και XNA χωρίς κανένα προγραμματισμό. Υποστηρίζεται ότι τα βασικά μαθαίνονται μέσα σε μια ώρα.
- GameSalad – δημιουργία παιχνιδιού με Drag and drop πλατφόρμα που επιτρέπει την ανάπτυξη και τη δημοσίευση σε Windows Phone, iOS, Android και HTML5.

- Gideros Mobile - Χρησιμοποιεί Flash-συναρτήσεις για τη δημιουργία παιχνιδιών και εφαρμογών. Δεν είναι τόσο απλό όσο μερικές από τις παραπάνω πλατφόρμες των παιχνιδιών, αλλά δεν χρειάζεται υψηλού επιπέδου γλώσσες προγραμματισμού. Είναι δωρεάν για iOS και Android , χρησιμοποιεί τη γλώσσα προγραμματισμού LUA και έχει το δικό του IDE.
- LiveCode - επιτρέπει τη χρήση ενός "έξυπνου" γραφικού περιβάλλοντος και «αγγλικής γλώσσας προγραμματισμού" για τη γρήγορη δημιουργία παιχνιδιών, τα οποία μπορούν να μεταφέρονται μεταξύ iOS και Android.
- Game Editor - πλατφόρμα δημιουργίας παιχνιδιού Open source που είναι ελεύθερο να χρησιμοποιηθεί (εφ 'όσον το παιχνίδι είναι open source). Υποστηρίζει iOS και Windows Mobile.
- Mominis – είναι πλατφόρμα δημιουργίας παιχνιδιού που βασίζεται στο drag and drop. Είναι ελεύθερη, εύκολη στη χρήση και κατάλληλη για όλα τα ήδη λειτουργικών συστημάτων καθώς στο κάθε παιχνίδι που δημιουργείται παρέχεται υποστήριξη για την κάθε πλατφόρμα.
- YoyoGames Gamemaker – πλατφόρμα μηχανής παιχνιδιών για κινητά για iOS και Android που χρησιμοποιεί Drag and drop.

2.9.2 Μηχανές παιχνιδιού με χρήση προγραμματισμού

Για τη δημιουργία παιχνιδιών σε iOS και Android χρησιμοποιούνται ισχυρές μηχανές, οι οποίες είναι πλαίσια σε λογισμικό που παρέχουν τα βασικά στοιχεία του παιχνιδιού, όπως 2D ή 3D γραφική απόδοση, φυσική, ήχο, AI και animation. Οι μηχανές αυτές απαιτούν γερές γνώσεις προγραμματισμού με αντάλλαγμα το μεγάλο μέγεθος των δυνατοτήτων και την ευελιξία που προσφέρουν. Αυτός είναι ο λόγος που οι μηχανές αυτές προσελκύουν το ενδιαφέρον μας. Παρακάτω παρουσιάζεται ένα ευρύ φάσμα των μηχανών παιχνιδιού, οι οποίες απαιτούν προγραμματισμό.

Μηχανές παιχνιδιού για iOS και για Android

Πολυπλατφόρμες-Cross platform (iOS/Android):

- Edgelib – μία από τις πιο ισχυρές και ευέλικτες 2D και 3D middleware μηχανές για cross-platform ανάπτυξη παιχνιδιού. Επιτρέπει τη δημιουργία ανώτερων εφαρμογών και παιχνιδιών σε μια τεράστια ποικιλία από πλατφόρμες κινητών. Δημιουργεί multi-platform εφαρμογές με περιβάλλον εργασίας για Apple iOS, το Google Android, Symbian, Windows Mobile, επιφάνεια εργασίας (Linux/Windows/OS X) και πολλές περισσότερες πλατφόρμες.

- Unity3D - είναι ένα εργαλείο ανάπτυξης παιχνιδιών που έχει σχεδιαστεί για να αφήνει τον προγραμματιστή να επικεντρωθεί στη δημιουργία καταπληκτικών παιχνιδιών. Είναι μια cross πλατφόρμα εφαρμογών, εύκολη στη διαχείριση στοιχείων-assets, βασίζεται στη φυσική και είναι πολύ ισχυρή για 3D παιχνίδια. Μπορεί όμως να χρησιμοποιηθεί και για 2D παιχνίδια, τα οποία θα είναι 3D αλλά επεξεργασμένα κατάλληλα ώστε να φαίνονται και να ελέγχονται σαν να είναι 2D. Το Unity δεν εμποδίζει τον προγραμματιστή και του επιτρέπει να επικεντρωθεί απλώς στη δημιουργία του παιχνιδιού. Είναι κατάλληλο για ανάπτυξη στο διαδίκτυο, τα κινητά, ή την κονσόλα. Υπάρχει επίσης, η δυνατότητα της αγοράς 3D στοιχείων και animations και της εύκολης εισαγωγής τους στην εφαρμογή με σκοπό τη διευκόλυνση του προγραμματιστή. Είναι εύκολο στον προγραμματισμο καθώς χρησιμοποιεί Javascript και C#.
- Unity Mobile - Mobile έκδοση της δημοφιλούς μηχανής 3D Unity. Υποστηρίζει iOS και Android.
- Emo – framework παιχνιδιού σε κινητά. Χρησιμοποιεί Squirrel scripting γλώσσα και βασίζεται σε OpenGL ES και OpenAL / OpenSL. Το παιχνίδι μπορεί να τρέξει τόσο σε iOS όσο και σε Android με μία μόνο υλοποίηση.
- Unreal Development Kit - Δωρεάν έκδοση της βιομηχανίας Unreal Engine III, που μας επιτρέπει να δημιουργήσουμε τα παιχνίδια χρησιμοποιώντας UE3 χωρίς άδεια Unreal Engine 3. Το UDK χρησιμοποιείται για να δημιουργήσει παιχνίδια, εφαρμογές και προηγμένες 3D προσομοιώσεις. Υποστηρίζει iOS και Android.
- JMonkey Engine - Δωρεάν λογισμικό ανοιχτού κώδικα Java OpenGL μηχανή που επιτρέπει τον προγραμματισμό σε Java και εφαρμόζεται σε οποιοδήποτε OpenGL 2 συμβατή συσκευή.
- ShiVa3D - είναι το εργαλείο που επιλέγουν οι προγραμματιστές γιατην εύκολη δημιουργία εκπληκτικών 3D εφαρμογών και παιχνιδιών σε πραγματικό χρόνο για Windows Mobile, Mac OS, Linux, iPhone- iOS, Android, Palm, το Wii , το iPad και BlackBerry OS. Η Shiva 3D ισχυρίζεται ότι είναι ισχυρή και «η πιο πολλαπλή πλατφόρμα» μηχανής παιχνιδιού 3D, 3D WYSIWYG Editor και MMO Server. Μερικά από τα χαρακτηριστικά της είναι τα εξής: Android, Palm και το Wii μηχανή, Ενιαίο Εργαλείο Συγγραφής, Μηχανή Plug-ins & Επεκτάσεις, Native C++ Complication, Mesh Modification API, Shiva Editor PLE Εξαγωγή, Compound Dynamic Body.
- Corona SDK - δημοφιλής πλατφόρμα ανάπτυξης εφαρμογών app που επιτρέπει τη δημιουργία παιχνιδιών, καθώς και άλλες εφαρμογές για iOS και Android. Η Corona είναι μια 2D μηχανή πολύ απλή στη χρήση και έχει την ευκολότερη και συντομότερη σύνταξη LUA. Είναι αυστηρά για 2D παιχνίδια και είναι περισσότερο SDK παρά μηχανή/εφαρμογή όπως είναι το Unity 3D. Υποστηρίζεται ότι χρησιμοποιείται από περισσότερους από 150.000 προγραμματιστές. Το πλαίσιο της Corona έχει αυξήσει δραματικά την παραγωγικότητα. Εργασίες όπως το animating αντικειμένων σε OpenGL

ή δημιουργία διεπαφής χρήστη widgets χρειάζεται μόνο μία γραμμή κώδικα, και οι αλλαγές είναι άμεσα ορατές στο Corona Simulator. Ο έλεγχος είναι άμεσος και γρήγορος χωρίς χρονοβόρες διαδικασίες όμως μετά τη δημιουργία του παιχνιδιού πρέπει ο κώδικας να σταλεί στην Corona για τη σύνταξη. Η Corona είναι η μόνη ολοκληρωμένη λύση για την ανάπτυξη σε όλες τις πλατφόρμες, εκδόσεις OS, και τα μεγέθη οθόνης. Με την Corona μπορεί η εγγραφή και η δημιουργία του παιχνιδιού να γίνει μόνο μία φορά στο iOS ή στο Android και η Corona αυτόματα κλιμακώνει το περιεχόμενό στις συσκευές από τα κινητά τηλέφωνα έως τα tablets.

- Libdx - Android, HTML5 και Java 3D/2D πλαίσιο ανάπτυξης παιχνιδιών. Δωρεάν για χρήση, επιτρέπει την ανάπτυξη σε πολλαπλές πλατφόρμες με μία μόνο υλοποίηση.
- Papaya Social Game Engine - 2D μηχανή παιχνιδιού από το σχεδιασμό της Papaya για να βοηθήσει τους προγραμματιστές να δημιουργήσουν την κοινωνία των παιχνιδιών των κινητών ευκολότερη. Υποστηρίζει iOS και Android.
- MoSync – Ελαφρύ εργαλείο για HTML5/JavaScript προγραμματιστές για να φτιάξουν εφαρμογές για iOS, Android και Windows Phone.
- NME - Δωρεάν ανοικτός κώδικας που επιτρέπει την ανάπτυξη των Android, iOS, BlackBerry και Windows εφαρμογών του τηλεφώνου από μία ενιαία codebase. Δεν απαιτούνται δεξιότητες σε C ή C++.
- SIO2 – είναι ένα OpenGLES με βάση τη 2D και 3D μηχανή παιχνιδιού για iOS και Android, MacOS και τα Windows, το οποίο παρέχει όλες τις σύγχρονες λειτουργίες μηχανής παιχνιδιού ενσωματωμενές με τιμή πολύ χαμηλότερη συγκριτικά με άλλες μηχανές στην αγορά με τον ίδιο τύπο ποιότητας. Η μηχανή επιτρέπει επίσης τη μεταφορά του παιχνιδιού στην αγορά Mac και Windows. Βασικό χαρακτηριστικό του είναι ότι είναι εξαιρετικά γρήγορο και ευέλικτο, επιτρέπει τη δημιουργία AAA παιχνιδιών για πλατφόρμα κινητών και δεν επιβάλλει τη χρήση κάποιου συγκεκριμένου IDE για να κωδικοποιηθούν ή να χτιστούν τα παιχνίδια. Χρησιμοποιεί σύνταξη cross-platform C / C++.
- Marmalade - αναπτυξιακό εργαλείο παιχνιδιού που επιτρέπει την ανάπτυξη cross-platform παιχνιδιών και εφαρμογών υποστηρίζοντας C / C++ , πρότυπα βιβλιοθηκών, STL και OpenGL ES . Είναι ιδιαίτερα συμβατή με τα πρότυπα πλατφόρμας και επιτρέπει την ανάπτυξη τόσο φορητών όσο και επιτραπέζιων υπολογιστών-desktop. Βοηθάει τους προγραμματιστές να δημιουργήσουν και να διανείμουν cross-platform εφαρμογές γρήγορα, εύκολα και χωρίς συμβιβασμούς. Υποστηρίζει iOS, Android και BlackBerry.
- BattryTech - αναπτυξιακό πλαίσιο παιχνιδιού που επιτρέπει την εγγραφή C++ κώδικα για Android και iPhone πλατφόρμες.
- App Game Kit - Χρησιμοποιεί μια γλώσσα BASIC script και επιτρέπει να αναπτυχθεί ο κώδικας των προγραμματιστών σε πολλαπλές πλατφόρμες όπως το iOS, Android και BlackBerry OS.

- Antiryad Gx - Cross-πλατφόρμα multi-core 3D και 2D μηχανή παιχνιδιού για iOS, Android και WP. Υποστηρίζει C, C++ και Gel.
- PhoneGap - είναι μια πλατφόρμα HTML5 app που επιτρέπει τη δημιουργία εφαρμογών με τεχνολογίες web και την απόκτηση πρόσβασης σε APIs και στα καταστήματα app. Το PhoneGap αξιοποιεί τις τεχνολογίες web που οι προγραμματιστές ήδη γνωρίζουν καλύτερα: HTML και JavaScript. Το PhoneGap είναι μια εφαρμογή ανοιχτού κώδικα των ανοικτών προτύπων. Αυτό σημαίνει ότι οι προγραμματιστές και οι εταιρείες μπορούν να χρησιμοποιήσουν το PhoneGap για εφαρμογές κινητών που είναι δωρεάν, εμπορικά, ανοιχτού κώδικα, ή οποιοδήποτε συνδυασμό αυτών. Το PhoneGap θα παραμείνει πάντα δωρεάν και θα έχει ανοικτό κώδικα με άδεια από το MIT.
- Titanium Mobile - μεταφράζει τις γνώσεις του προγραμματιστή πάνω στο web σε εφαρμογές που εκτελούνται και μοιάζουν ακριβώς σαν να γράφτηκαν σε Objective-C [iPhone και iPad] ή Java [Android]. Με πάνω από 300 APIs και με μια ακμάζουσα κοινότητα προγραμματιστών που παρέχουν την απαραίτητη υποστήριξη, είναι δυνατή η δημιουργία εφαρμογών που μπορούν να χαρακτηριστούν ως κοινωνικές, τοπικές, μέτρια ακριβές, διαδραστικές, και επεκτάσιμες.
- Cocos2d-x - είναι μια δωρεάν μηχανή παιχνιδιού 2D για κινητά ανοιχτού κώδικα που διατίθεται βάσει της άδειας MIT. Χρησιμοποιεί C++ ως μητρική γλώσσα προγραμματισμού. Είναι είναι μια πολυ-πλατφόρμα, ελαφρυά, και φιλική στον προγραμματιστή . Προέρχεται από το "cocos2d ", που κάνει επίσης Android εφαρμογές, κρατώντας τα κύρια χαρακτηριστικά συγχρονισμένα με αυτό. Μπορεί να φτιάξει εφαρμογές για το iPhone, iPod και iPad, καθώς και για Android συσκευές. Για τη σωστή χρήση αυτής της πλατφόρμας είναι απαραίτητη η γνώση σε βασική Objective C και C++, διαχείριση μνήμης και στον τρόπο με τον οποίο ο υπολογιστής αποθηκεύει και μετακινεί τα δεδομένα.Αν το παιχνίδι είναι 2D τότε το Cocos2d είναι κατάλληλο εργαλείο, καθώς είναι συγκεκριμένα για 2d και παρέχει έτοιμα παιχνίδια και εφαρμογές που μπορούν να χρησιμοποιηθούν δωρεάν.
- Moai - Δωρεάν και ανοιχτού κώδικα. Το Moai SDK είναι μια πολλαπλή πλατφόρμα ανάπτυξης παιχνιδιών για κινητά, που μπορεί να εφαρμοστεί σχεδόν παντού ,όμως έχει πολύ κακή τεκμηρίωση. Χρησιμοποιεί τη γλώσσα προγραμματισμού Lua σε φορητές συσκευές στο cloud. Η Moai SDK μπορεί να χειριστεί γραφικά, animation, input, φυσική, συγκρούσεις, και πολλά άλλα. Το Moai Cloud φιλοξενεί τη λογική του παιχνιδιού, τις βάσεις δεδομένων και επιπλέον το περιεχόμενο του παιχνιδιού, καθώς και βασικές υπηρεσίες που χρειάζονται για τη δημιουργία και διαχείριση μεγάλων παιχνιδιών. Το Moai είναι σχεδιασμένο για έμπειρους προγραμματιστές παιχνιδιών που επιθυμούν να χρησιμοποιήσουν το Lua για την ανάπτυξη κινητών και cloud. Η χρήση του Lua σε πολλαπλές πλατφόρμες ανάπτυξης σημαίνει ότι ο προγραμματιστής

μπορεί να επικεντρωθεί στην ανάπτυξη μεγάλων παιχνιδιών και όχι στην εναλλαγή μεταξύ πολλαπλών γλωσσών.

- Simple DirectMedia Layer - είναι μια cross-platform βιβλιοθήκη πολυμέσων σχεδιασμένη για να παρέχει πρόσβαση ήχου χαμηλού επιπέδου, πληκτρολογίου, ποντικιού, joystick, 3D hardware μέσω του OpenGL, και 2D βίντεο framebuffer. Χρησιμοποιείται από το λογισμικό αναπαραγωγής MPEG, emulators, και πολλά δημοφιλή παιχνίδια. Το SDL είναι γραμμένο σε C, αλλά λειτουργεί εγγενώς με C++, και έχει συνδέσεις με διάφορες άλλες γλώσσες, συμπεριλαμβανομένων των Ada, C#, D, Eiffel, Erlang, Euphoria, Go, Guile, Haskell, Java, Lisp, Lua, ML, Objective C, Pascal, Perl, PHP, Pike, Pliant, Python, Ruby, Smalltalk, και Tcl.
- Unreal engine 3 – πλατφόρμα μηχανής παιχνιδιού για κινητά που υποστηρίζει Android και iOS. Χρησιμοποιεί UnrealScript και Unreal Kismet (advanced visual scripting solution C++).

Μηχανές παιχνιδιού για iOS

Συγκερκιμένες μηχανές παιχνιδιού για iOS :

- iTorque - 2D game editor για την ανάπτυξη iPad, iPhone και iPod Touch..
- Cocos2D - πλαίσιο για τη δημιουργία 2D παιχνιδιών για το iPod Touch, iPhone και iPad. Υποστηρίζεται ότι χρησιμοποιείται από περισσότερα από 2500 παιχνίδια στο App Store.
- Sparrow - μηχανή παιχνιδιού για iOS, δωρεάν στη χρήση ανοικτού κώδικα. Χρησιμοποιεί καθαρή Objective C και χτίστηκε εξ'ολοκλήρου για το iPhone και το iPad.
- Oolong - Δωρεάν στη χρήση της μηχανής παιχνιδιού γραμμένη σε C++ που επιτρέπει τη δημιουργία νέων παιχνιδιών iOS και εφαρμόζεται σε υπάρχοντα παιχνίδια των iOS συσκευών.
- Bork3D - Χαμηλού κόστους 3D μηχανή παιχνιδιού για iPhone, iPad και iPod Touch. Κατάλληλο μόνο για προγραμματιστές με εμπειρία στον προγραμματισμό.
- NinevehGL - 3D μηχανή που κατασκευάστηκε με Objective C, διαθέτει την ικανότητα να εισάγει 3D μοντέλα απευθείας από 3D λογισμικά, πολλαπλά shaders σε ένα ενιαίο αντικείμενο και πολλά ειδικά εφέ.
- Newton – μηχανή φυσικής πραγματικού χρόνου ανοιχτού κώδικα με άδεια zlib που έχει σχεδιαστεί για να χρησιμοποιείται με βασικές γνώσεις των αρχών της Φυσικής. Ελεύθερο στη χρήση.
- Kobold2D - Το Kobold2D είναι μια εκτεταμένη και βελτιωμένη έκδοση της δημοφιλής μηχανής παιχνιδιού Cocos2D for iPhone. Ό, τι ισχύει για Cocos2D μπορεί ακόμα να εφαρμοστεί. Το Kobold2D είναι πιο βολικό στη χρήση, πιο εύκολο, πιο ισχυρό και πιο ευέλικτο από το Cocos2D με όλα τα έγγραφα που είναι διαθέσιμα online και offline .

Είναι ελεύθερο και open source με επιπλέον περιεχόμενο για την υποστήριξη της ανάπτυξης του καθώς διαθέτει περισσότερα και καλύτερα δείγματα projects ,καλύτερη τεκμηρίωση, και είναι προεγκατεστημένο με πολλά πρότυπα για διάφορα είδη παιχνιδιών. Με αυτό τον τρόπο γίνεται ευκολότερο για κάποιον να αρχίσει τον προγραμματισμό παιχνιδιών iOS. Χρησιμοποιείται για την ανάπτυξη παιχνιδιών σε iPhone, iPod touch, iPad και Mac OS X στο Apple Stores

- DragonFire - 2D iPhone και iPad εργαλείο ανάπτυξης για τα Windows.

Μηχανές παιχνιδιού για Android

Συγκεκριμένες μηχανές παιχνιδιού για Android:

- Candroid - μηχανή παιχνιδιού για το Android δωρεάν στη χρήση(εκτιμάται δωρεά).
- AndEngine - μηχανή παιχνιδιού ελεύθερο στη χρήση Android 2D OpenGL.
- jPCT AE - Δωρεάν 3D μηχανή παιχνιδιού που είναι μια εφαρμογή της jPCT για το Android. Υποστηρίζει OpenGL ES 1.x και 2.0.
- Android Box2D - Βραβευμένη 2D μηχανή φυσικής άκαμπτου σώματος γραμμένη σε C++ για τους προγραμματιστές Android.
- Catcake - Δωρεάν στη χρήση 3D μηχανή γραφικών για το Android (καθώς και το Linux), που προορίζεται να είναι εύκολο στη χρήση και υψηλής απόδοσης.
- Cocos2D για το Android - πλαίσιο για τη δημιουργία 2D παιχνιδιών για το Android. Βασίζεται στο πλαίσιο Cocos2D για το iPhone. Ελεύθερο στη χρήση (εκτιμάται δωρεά).

2.10 Επιλογή της πλατφόρμας Unity 3D για τη δημιουργία του παιχνιδιού

Μας ενδιαφέρει η πλατφόρμα να χρησιμοποιεί προγραμματισμό και να υποστηρίζει τα λειτουργικά συστήματα Android και iOS καθώς και τη δημιουργία τρισδιάστατου παιχνιδιού. Θεωρητικά πιο ικανοποιητικές για τα 3D περιβάλλοντα φαίνονται οι μηχανές παιχνιδιού Unity 3D, Unreal Development Kit και Shiva 3D.

2.10.1 Unreal Development Kit

Η Unreal Development Kit είναι μηχανή παιχνιδιού που παρέχει ανάπτυξη για DirectX υπολογιστές, Xbox 360, PlayStation 3, και Oses (Mac OS X, iOS) που βασίζεται σε OpenGL.

Η Unreal Development Kit (UDK) είναι μια δωρεάν έκδοση του SDK UE3, η οποία είναι διαθέσιμη στο ευρύ κοινό, αλλά δεν συμπεριλαμβάνει τον πηγαίο κώδικα. Η UDK παρέχει υποστήριξη για όλα τα UDK-based παιχνίδια σε πλατφόρμες iOS, OS X και Windows.

Η Unreal Engine είναι μια μηχανή παιχνιδιού που αναπτύχθηκε από την Epic Games, και είναι η πρώτη που έφτιαξε first-person shooter παιχνίδι. Παρά το γεγονός ότι έχει αναπτυχθεί κυρίως για first-person shooters, έχει χρησιμοποιηθεί με επιτυχία σε μια ποικιλία από άλλα είδη. Με τον κώδικα της γραμμένο σε C++ ή σε Unrealscript, η Unreal Engine διαθέτει υψηλού βαθμού δυνατότητα μεταφοράς και είναι ένα εργαλείο που χρησιμοποιείται από πολλούς προγραμματιστές παιχνιδιών σήμερα.

Η τρέχουσα έκδοση είναι η Unreal Engine 4, και έχει σχεδιαστεί για της Microsoft τη DirectX 9 (για Windows και Xbox 360), τη DirectX 10 (για Windows Vista) και DirectX 11 (για τα Windows 7 και αργότερα), OpenGL (για το OS X, Linux, PlayStation 3, Wii U, και iOS), Android, Στάδιο 3D (για το Adobe Flash Player 11 και μετά) και JavaScript/WebGL (για HTML5).

Η Unreal Engine είναι κορυφαία μηχανή παιχνιδιών και είναι μία από τις καλύτερες που υπάρχουν. Υποστηρίζει πολύ υψηλού επιπέδου γραφικά και αναλύσεις, παρέχει πολλές δυνατότητες, υποστηρίζοντας πολλές πλατφόρμες. Παρόλα αυτά δεν την επιλέξαμε καθώς η δωρεάν έκδοση που προσφέρει δεν συμπεριλαμβάνει τον πηγαίο κώδικα, γεγονός το οποίο μας περιορίζει πάρα πολύ καθώς πρέπει να γραφτούν τα πάντα εξαρχής. Επιπλέον η μηχανή αυτή έχει μεγάλες απαιτήσεις σε πόρους του συστήματος ώστε να υποστηρίξει τα γραφικά υψηλού επιπέδου, γεγονός που μας έκανε να μην την προτιμήσουμε καθώς δεν θέλουμε να επιβαρύνουμε το σύστημά μας αλλά να είναι όσο το δυνατόν πιο ελαφρύ καθώς σκοπός μας είναι να είναι συμβατό με τις περισσότερες φορητές συσκευές, οι οποίες δεν έχουν όλες ισχυρό επεξεργαστή ή πολύ καλή ανάλυση. Άλλος ένας λόγος, επίσης πολύ σημαντικός, ήταν ότι δεν υπήρχαν αρκετά έτοιμα παραδείγματα ή μεθοδολογίες για την ανάπτυξη του παιχνιδιού ώστε να κατανοήσουμε πως λειτουργεί η μηχανή αυτή εις βάθος και λεπτομερώς.

2.10.2 Shiva 3D

Η ShiVa3D είναι μια 3D μηχανή παιχνιδιού με ένα πρόγραμμα επεξεργασίας γραφικών που έχει σχεδιαστεί για τη δημιουργία εφαρμογών και βιντεοπαιχνιδιών για το Web, τις κονσόλες και τις φορητές συσκευές. Μπορεί να παράγει παιχνίδια και 3D προσομοιώσεις γραφικών για Windows, Mac, Linux, iPhone, iPad, BlackBerry Tablet OS/10 BlackBerry, Android, Palm OS, το Wii και το WebOS, αυτόνομα ή ενσωματωμένα σε web browsers.

Η μηχανή του παιχνιδιού χρησιμοποιεί OpenGL, OpenGL ES ή DirectX, και μπορεί επίσης να τρέξει σε λειτουργία του λογισμικού. Η ShiVa3D υποστηρίζει plug-ins, όπως την NVIDIA PhysX, την F-Mod βιβλιοθήκη ήχων και το ARToolKit.

Περισσότερες από 8.000 εφαρμογές έχουν δημιουργηθεί με τη χρήση του λογισμικού. Στις 19 Μαρτίου του 2010 η εταιρεία κυκλοφόρησε μια beta έκδοση του Android SDK της, καθιστώντας την το πρώτο 3D development kit για την πλατφόρμα Android.

Αν και η Shiva παρείχε πολλές δυνατότητες δεν την επιλλέξαμε καθώς δεν υπήρχε τόσο μεγάλη υποστήριξη από την κοινότητά της, υπήρχαν λίγα έτοιμα παραδείγματα παιχνιδιού που να εξηγούν τη διαδικασία ανάπτυξής του, οπότε αυτό μας δυσκόλεψε στην εκμάθηση της μηχανής εις βάθος.

2.10.3 Επιλογή της Unity 3D

Η Unity είναι και αυτή μία από τις κορυφαίες μηχανές παιχνιδιού. Ανταποκρίνεται και η Unity σε πάρα πολλές διαφορετικές πλατφόρμες, δηλαδή Android, IOS, PC, MAC, Linux, Web Player, Flash, PS3, Xbox 360, Wii U.

Δεν μας περιορίζει ως προς τη γλώσσα προγραμματισμού καθώς υποστηρίζει τρεις διαφορετικές γλώσσες προγραμματισμού, τη JavaScript, τη C#, και τη Python, από τις οποίες μας ενδιαφέρει η γλώσσα JavaScript. Επίσης, η Unity παρέχει δωρέαν τον πηγαίο κώδικα, ο οποίος αποτελεί τη βάση για την ανάπτυξη του παιχνιδιού καθώς μπορεί να χρησιμοποιηθεί σε πάρα πολλές περιπτώσεις αυτούσιος ή τροποποιημένος.

Απαιτεί υψηλού επιπέδου 3D μοντέλα και animations καθώς υποστηρίζει το DirectX11, δίνοντας έτσι την ευκαιρία να αναπτυχθεί το γραφικό κομμάτι της διπλωματικής εργασίας, χωρίς όμως να επιβαρύνουμε πολύ το παιχνίδι έτσι ώστε να είναι αρκετά ελαφρύ για τα κινητά. Επιπλέον, είναι δυνατή η υλοποίηση 3D/2D (ή αλλιώς 2,5D) από το ίδιο εργαλείο. Η φωτογραφία που ακολουθεί δείχνει ένα παράδειγμα του τι είναι δυνατόν να γίνει με το DirectX11:

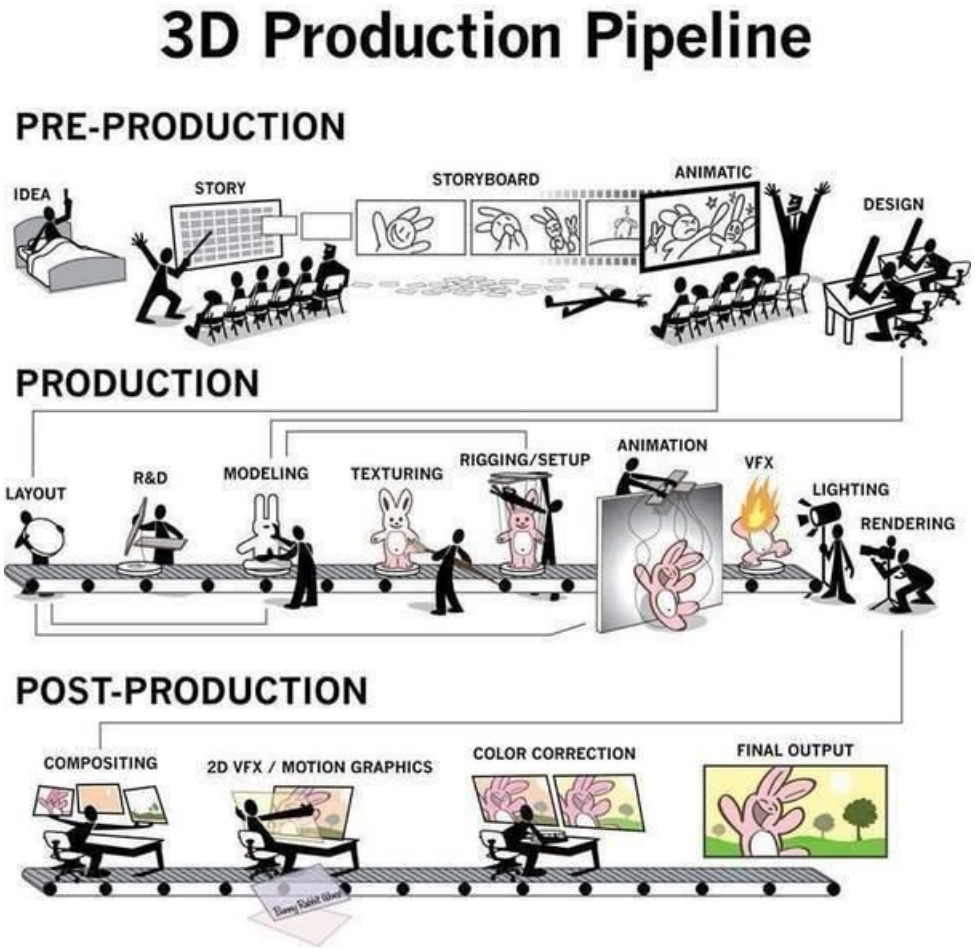
Συνεργάζεται με πολλά εργαλεία τα οποία βοηθούν στην γρηγορότερη υλοποίηση του project, όπως το Photoshop και το Flash Player, που μπορούν να χρησιμοποιηθούν στο σχεδιασμό. Διαθέτει έτοιμα αντικείμενα-assets, ήρωες και περιβάλλοντα που μπορούν να ενσωματωθούν αυτούσια ή επεξεργασμένα στο project χωρίς να χρειάζεται να δημιουργηθούν από το μηδέν.

Είναι μια πολύ ισχυρή μηχανή ανάπτυξης παιχνιδιών για κινητά που συγκεντρώνει μια ισχυρή μηχανή απόδοσης, ένα συνεχώς αναβαθμιζόμενο εργαλείο ανάπτυξης που περιλαμβάνει σκιές σε πραγματικό χρόνο και δυναμικά περιβάλλοντα, προσφέρει τεκμηρίωση σε βάθος, χιλιάδες έτοιμα assets-στοιχεία, και πολύτιμες συμβουλές από την συνεχώς διευρυνόμενη κοινότητα των προγραμματιστών της Unity iOS και Android.

Τέλος, υπάρχει άφθονο υλικό εκμάθησης και παραδειγμάτων όσον αφορά τη χρήση της Unity3D, όπως είναι το Unity3DStudents, tutorials, εγχειρίδια, βίντεο με παραδείγματα, και forums από άλλους developers, τα οποία βοήθησαν πάρα πολύ στην εκμάθηση της μηχανής και την ανάπτυξη του παιχνιδιού.

2.11 3D computer graphics

Τα 3D γραφικά υπολογιστών είναι τα γραφικά που χρησιμοποιούν μια τρισδιάστατη αναπαράσταση των γεωμετρικών στοιχείων που είναι αποθηκευμένα στον υπολογιστή για τους σκοπούς της εκτέλεσης υπολογισμών και απόδοσης 2D εικόνων. Αυτές οι εικόνες μπορούν να αποθηκεύονται για να προβληθούν αργότερα ή εμφανίζονται σε πραγματικό χρόνο.



Τα 3D γραφικά υπολογιστών συχνά αναφέρεται ως 3D μοντέλα. Εκτός από τα rendered γραφικά, το μοντέλο περιέχεται μέσα στο γραφικό αρχείο δεδομένων. Ένα 3D μοντέλο είναι η μαθηματική αναπαράσταση του τρισδιάστατου αντικειμένου. Ένα μοντέλο δεν είναι τεχνικά γραφικό μέχρι να εμφανιστεί. Λόγω της 3D εκτύπωσης, τα 3D μοντέλα δεν περιορίζονται στον εικονικό χώρο. Ένα μοντέλο μπορεί να εμφανίζεται οπτικά ως μια δισδιάστατη εικόνα, μέσω

μιας διαδικασίας που ονομάζεται 3D rendering ή χρησιμοποιείται σε μη γραφικές προσομοιώσεις ηλεκτρονικού υπολογιστή και υπολογισμών .

Η δημιουργία 3D γραφικών του υπολογιστή χωρίζεται σε τρία βασικά στάδια, το 3D modeling που είναι η διαδικασία σχηματισμού ενός μοντέλου του υπολογιστή δίνοντας το σχήμα ενός αντικειμένου, το Layout και animation που είναι η κίνηση και η τοποθέτηση αντικειμένων μέσα σε μια σκηνή, και το 3D rendering που περιλαμβάνει τους υπολογισμούς που γίνονται με βάση τη τοποθέτηση του φως, τους τύπους των επιφανειών, καθώς και άλλων ιδιοτήτων, για τη δημιουργία της εικόνας μέσω υφής.

2.11.1 Μοντελοποίηση

Το μοντέλο περιγράφει τη διαδικασία σχηματισμού του σχήματος ενός αντικειμένου. Δύο είναι οι πιο κοινές πηγές 3D μοντέλων, εκείνες με τις οποίες ένας καλλιτέχνης ή μηχανικός δημιουργεί τα μοντέλα στον υπολογιστή με κάποιο είδος εργαλείου 3D modeling, και εκείνες με τις οποίες τα μοντέλα σαρώνονται σε έναν υπολογιστή από αντικείμενα του πραγματικού κόσμου. Τα μοντέλα μπορούν επίσης να παραχθούν διαδικαστικά ή μέσω φυσικής προσομοίωσης. Ένα 3D μοντέλο σχηματίζεται από σημεία που ονομάζονται κορυφές (vertices ή vertexes) που ορίζουν το σχήμα και τη μορφή των πολυγώνων. Ένα πολύγωνο είναι μια περιοχή που αποτελείται από τουλάχιστον τρεις κορυφές (τρίγωνο). Ένα πολύγωνο τεσσάρων σημείων είναι ένα quad, και ένα πολύγωνο πάνω από τέσσερα σημεία είναι ένα n-γωνο. Η συνολική ακεραιότητα του μοντέλου και η καταλληλότητά του για χρήση σε κινούμενα σχέδια εξαρτώνται από τη δομή των πολυγώνων.

2.11.2 Layout and animation

Πριν από το rendering σε μια εικόνα, τα αντικείμενα πρέπει να τοποθετηθούν (laid out) σε μια σκηνή. Αυτό ορίζει τις χωρικές σχέσεις μεταξύ των αντικειμένων, συμπεριλαμβανομένης της θέσης και του μεγέθους. Το Animation αναφέρεται στην χρονική περιγραφή ενός αντικειμένου, δηλαδή στο πώς κινείται και παραμορφώνεται την πάροδο του χρόνου. Οι δημοφιλείς μέθοδοι περιλαμβάνουν βασικά καρέ, αντίστροφη κινηματική και καταγραφή της κίνησης (keyframing, inverse kinematics, και motion capture). Αυτές οι τεχνικές χρησιμοποιούνται συχνά σε συνδυασμούς. Όπως και με τη μοντελοποίηση, η φυσική προσομοίωση καθορίζει επίσης την κίνηση.

2.11.3 Rendering

Το Rendering μετατρέπει ένα μοντέλο σε μια εικόνα είτε με τη προσομοίωση του φωτός μεταφοράς για τη λήψη φωτορεαλιστικών εικόνων, είτε με την εφαρμογή κάποιου είδους στυλ, όπως μη-φωτορεαλισμού. Οι δύο βασικές λειτουργίες σε ρεαλιστική απόδοση είναι οι μεταφορές (πόσο φως δέχεται από το ένα μέρος στο άλλο) και η σκέδαση (πώς αλληλεπιδρούν οι επιφάνειες με το φως). Αυτό το βήμα γίνεται συνήθως με τη χρήση λογισμικού 3D γραφικών

υπολογιστή ή 3D γραφικά API. Η αλλαγή της σκηνής σε μια κατάλληλη μορφή απόδοσης περιλαμβάνει επίσης 3D προβολή, η οποία εμφανίζει μια τρισδιάστατη εικόνα σε δύο διαστάσεις.

2.11.4 Κοινοτήτες

Υπάρχει μια πληθώρα από ιστοσελίδες που έχουν σχεδιαστεί για να βοηθήσουν, να εκπαιδεύσουν και να υποστηρίξουν τους 3D γραφίστες. Ορισμένες διευθύνονται από προγραμματιστές λογισμικού και τους παρόχους του περιεχομένου, αλλά υπάρχουν επίσης και standalone sites. Αυτές οι κοινότητες επιτρέπουν στα μέλη να ζητούν συμβουλές, να εκδίδουν tutorials, να παρέχουν κριτικές για το προϊόν ή να δημοσιεύουν παραδείγματα της δουλειάς τους.

2.11.5 Διάκριση από φωτορεαλιστικά γραφικά 2D

Τα 3D γραφικά υπολογιστών βασίζονται σε πολλούς από τους ίδιους αλγορίθμους των 2D vector γραφικών υπολογιστών στο μοντέλο wire-frame και 2D γραφικών υπολογιστή raster στο τελικό rendered display. Σε λογισμικό γραφικών υπολογιστή, η διάκριση μεταξύ 2D και 3D είναι μερικές φορές θολή, οι 2D εφαρμογές μπορούν να χρησιμοποιήσουν 3D τεχνικές για την επίτευξη των αποτελεσμάτων όπως ο φωτισμός, και οι 3D μπορούν να χρησιμοποιούν 2D τεχνικές rendering. Δεν βασίζονται όλα τα 3D γραφικά υπολογιστών που εμφανίζονται σε ένα μοντέλο σε wireframe. Τα 2D γραφικά υπολογιστών με 3D φωτορεαλιστικά αποτελέσματα συχνά επιτυγχάνονται χωρίς wireframe μοντελοποίηση και μερικές φορές είναι δυσδιάκριτα στην τελική τους μορφή. Μερικά γραφικά λογισμικά τέχνης περιλαμβάνουν φίλτρα που μπορούν να εφαρμοστούν σε 2D διανυσματικά γραφικά ή 2D γραφικά raster σε διαφανή στρώματα. Οι εικαστικοί καλλιτέχνες μπορούν επίσης να αντιγράψουν ή να απεικονίσουν 3D εφέ και να χειριστούν φωτορεαλιστικά εφέ χωρίς τη χρήση φίλτρων.

2.12 Χαρακτηριστικά των σχεδιαστικών προγραμμάτων τρισδιάστατων χαρακτήρων που υποστηρίζει η Unity 3D

Όπως είδαμε η Unity υποστηρίζει πολλά διαφορετικά είδη αρχείων τρισδιάστατων στοιχείων από διαφορετικά προγράμματα τα οποία μπορεί να αναγνωρίσει αυτούσια ή μετατρέποντάς τα σε κάποια άλλη μορφή αρχείου. Για τη δημιουργία των χαρακτήρων και των στοιχείων του παιχνιδιού (assets), μας ενδιέφερε να χρησιμοποιήσουμε πρόγραμμα το οποίο υποστηρίζει πλήρως η Unity, αναγνωρίζοντας ολόκληρο το περιεχομένο του συμπεριλαμβανομένου του υλικού, των υφών, των κόκαλων, και των κινούμενων σχεδίων. Η

Unity υποστηρίζει αυτούσια τα αρχεία εφτά σχεδιαστικών προγραμμάτων, του Maya, του Cinema 4D, του 3ds Max, του Cheetah3D, του Modo, του Lightwave, και του Blender.

2.12.1 Maya

Η Autodesk Maya, που συνήθως αποκαλείται απλά Maya, είναι λογισμικό 3D γραφικών υπολογιστών που τρέχει σε Windows, Mac OS και Linux, η οποία ανήκει και αναπτύχθηκε από την Autodesk, Inc. Χρησιμοποιείται για τη δημιουργία διαδραστικών 3D εφαρμογών, συμπεριλαμβανομένων των βιντεοπαιχνιδιών, ταινιών, κινουμένων σχεδίων, τηλεοπτικών σειρών, ή οπτικών εφέ. Το προϊόν πήρε το όνομά του από τη σανσκριτική λέξη Μάγια (Maya माया), την ινδουιστική έννοια της ψευδαίσθησης.

2.12.2 3ds Max

Ο Autodesk 3ds Max, πρώην 3D Studio Max, είναι ένα πρόγραμμα 3D γραφικών υπολογιστών για την παραγωγή 3D animations, μοντέλων, και εικόνων. Έχει αναπτυχθεί και παράγεται από την Autodesk Media and Entertainment. Έχει ικανότητες μοντελοποίησης, μια ευέλικτη αρχιτεκτονική plugin και μπορεί να χρησιμοποιηθεί για την πλατφόρμα Microsoft Windows. Συχνά χρησιμοποιείται από τους σχεδιαστές παιχνιδιών βίντεο, πολλά εμπορικά τηλεοπτικά στούντιο και αρχιτεκτονικά στούντιο οπτικοποίησης. Επίσης χρησιμοποιείται για τα εφέ σε ταινίες και την προ-οπτικοποίηση της ταινίας. Διαθέτει εργαλεία μοντελοποίησης και animation, shaders, δυναμική προσομοίωση, συστήματα σωματιδίων, radiosity, κανονική δημιουργία χάρτη και rendering, παγκόσμιο φωτισμό, ένα προσαρμόσιμο περιβάλλον εργασίας χρήστη, και τη δική του γλώσσα scripting.

2.12.3 Cinema 4D

Το CINEMA 4D είναι ένα πρόγραμμα για 3D modeling, animation και rendering που αναπτύχθηκε από τη MAXON Computer GmbH. Είναι ικανό για το σχεδιασμό διαδικαστικής και πολυγωνικής/Subd μοντελοποίησης, animating, φωτισμού, υφής, rendering, και κοινών διεργασιών που εκτελούνται από προγράμματα τρισδιάστατου σχεδιασμού.

2.12.4 Cheetah3D

Το Cheetah3d είναι ένα πρόγραμμα γραφικών για 3D modeling, animation και rendering. Είναι γραμμένο σε Cocoa για Mac OS X. Το πρόγραμμα έχει ως στόχο αρχάριους και ερασιτεχνικούς 3D καλλιτέχνες. Προσφέρει μια σειρά από μεσαία και high-end χαρακτηριστικά, σε συνδυασμό με ένα σχετικά απλό user interface. Η απλότητα του είναι που το κάνει να ξεχωρίζει από άλλα προγράμματα. Είναι διαθέσιμο για Mac OS X. Μια απλή άδεια χρήσης κοστίζει 99 δολάρια. Μια δωρεάν δοκιμαστική έκδοση είναι διαθέσιμη, χωρίς τη δυνατότητα της αποθηκεύσης ή εξαγωγής μοντέλων.

2.12.5 Modo

Το Modo είναι ένα πρόγραμμα για πολύγωνη και υποδιαιρεμένη μοντελοποίηση επιφάνειας, γλυπτικής, ζωγραφική 3D, animation και rendering που αναπτύχθηκε από Luxology, LLC. Το πρόγραμμα ενσωματώνει χαρακτηριστικά όπως n-gons και edge weighting, και τρέχει σε Microsoft Windows, Linux και πλατφόρμες Mac OS X.

2.12.6 Lightwave

Το LightWave 3D είναι ένα πλήρες λογισμικό υψηλής απόδοσης 3D γραφικών υπολογιστή που αναπτύχθηκε από τη NewTek. Έχει χρησιμοποιηθεί σε ορισμένες μεγάλες blockbuster ταινίες, τηλεοπτικές εκπομπές, κινούμενα γραφικά, ψηφιακό matte painting, οπτικά εφέ, ανάπτυξη βιντεοπαιχνιδιών, τηλεοπτικές διαφημίσεις, σχεδιασμό προϊόντος, αρχιτεκτονική απεικονίσεων, εικονική παραγωγή, μουσικά βίντεο, προ-απεικονίσεις, και διαφήμιση. Το Lightwave έχει ένα πολύ γρήγορο φωτορεαλιστικό renderer και θεωρείται ως ένα εξαιρετικό εργαλείο για την ταχεία ανάπτυξη 3D περιεχομένου και χρησιμοποιείται σε ταινίες χαμηλού προϋπολογισμού.

2.12.7 Blender

Το Blender είναι ένα δωρεάν και open-source προϊόν λογισμικού για υπολογιστή 3D γραφικών που χρησιμοποιείται για τη δημιουργία ταινιών κινουμένων σχεδίων, οπτικών εφέ, τέχνης, τυπωμένων 3D μοντέλων, διαδραστικών εφαρμογών 3D και βιντεοπαιχνιδιών. Τα χαρακτηριστικά του Blender περιλαμβάνουν 3D modeling, UV ξετύλιγμα, ύφανση, rigging και skinning, προσομοίωση υγρών και καπνού, προσομοίωση των σωματιδίων, προσομοίωση του μαλακού σώματος, γλυπτική, animating, κίνηση, εντοπισμό κάμερας, rendering, επεξεργασία βίντεο και compositing. Διαθέτει επίσης ενσωματωμένη μηχανή παιχνιδιού.

2.12.8 Επιλογή του Maya

Ενδιαφερόμαστε για την ολοκληρωμένη δημιουργία χαρακτήρων ώστε να μπορέσουν να χρησιμοποιηθούν στο παιχνίδι μας, η οποία προσφέρεται και από τα εφτά σχεδιαστικά προγράμματα. Για να καταλήξουμε στο καταλληλότερο για το παιχνίδι μας στη Unity, συγκρίναμε τα προγράμματα αυτά μεταξύ τους, ως προς τις δυνατότητές τους και τις υπηρεσίες που προσφέρουν. Δεν χρειάστηκε να συγκρίνουμε τις μορφές αρχείων εισαγωγής και εξαγωγής που προσφέρουν, καθώς η Unity υποστηρίζει τις μορφές αρχείων των προγραμμάτων Maya, Cinema 4D, 3ds Max, Cheetah3D, Modo, Lightwave, και Blender, οπότε δεν μας απασχολεί η μορφή των αρχείων που θα έχουμε.

Το πρόγραμμα Cheetah 3D είναι διαθέσιμο μόνο σε πλατφόρμα OSX, ενώ εμείς δουλεύουμε σε πλατφόρμα Windows, οπότε αμέσως αποκλείσαμε το συγκεκριμένο πρόγραμμα.

Εκτός από το Blender που διατίθεται δωρεάν, τα προγράμματα 3ds Max, Maya και Cinema 4D διατίθονται δωρεάν σε φοιτητές χωρίς να υπάρχει περιορισμός στις δυνατότητες που παρέχουν. Έτσι κάναμε αίτηση με τα στοιχεία από τη σχολή μας και μια φωτογραφία του πάσο, αποκτώντας έτσι τους κωδικούς για να χρησιμοποιήσουμε τα τρία αυτά προγράμματα μόνο για μη εμπορικούς σκοπούς.

Επίσης βρήκαμε δωρεάν εκδόσεις και των υπόλοιπων προγραμμάτων οι οποίες παρείχαν μόνο τα βασικά από τις συνολικές δυνατότητες των προγραμμάτων, ή ήταν διαθέσιμα δωρεάν για λίγες μόνο μέρες ως δοκιμή. Αυτό όμως δεν είναι αρκετό, οπότε τα αποκλείσαμε από την επιλογή μας και μείναμε ανάμεσα στα Blender, 3ds Max, Maya και Cinema 4D.

Καθώς ο σκοπός μας είναι η ολοκληρωμένη δημιουργία τρισδιάστατων χαρακτήρων, συγκρίναμε τις δυνατότητές τους ως προς τις υφές και το χρωματισμό. Από τα προγράμματα που έχουμε στη διάθεσή μας δωρεάν, κατάλληλα για αυτές τις λειτουργίες είναι τα 3ds Max, Blender, Cinema 4D και Maya. Συγκρίνοντας επίσης ως προς το σχεδιασμό κινουμένων σχεδίων και την προσομοίωση, βλέπουμε πάλι ότι πιο ισχυρά με περισσότερες δυνατότητες είναι τα 3ds Max, Blender, Cinema 4D και Maya.

Καταλλήγοντας σε αυτά τα τέσσερα προγράμματα τα συγκρίναμε ως προς την απόδοση- rendering, την διασύνδεση και τη διεπαφή, χωρίς όμως να βρούμε ουσιαστικές διαφορές στα χαρακτηριστικά τους που να μας κάνει να αποκλείσουμε κάποιο από αυτά. Για τον λόγο αυτό εγκαταστήσαμε και τα τέσσερα από αυτά προγράμματα δοκιμάζοντάς τα ώστε να βρούμε το καταλληλότερο. Όπως διαπιστώσαμε, και τα τέσσερα προγράμματα είναι πολύ ισχυρά και μας καλύπτουν απόλυτα στις λειτουργίες που θέλουμε.

Τελικά επιλέξαμε να ασχοληθούμε με το Maya καθώς βρήκαμε πολύ μεγάλη βοήθεια στο διαδίκτυο από άλλους χρήστες αυτού του προγράμματος, αλλά και λεπτομερείς οδηγίες οι οποίες επεξηγούν αναλυτικά τη διαδικασία δημιουργίας ενός ήρωα και την εισαγωγή του στη Unity.

ΚΕΦΑΛΑΙΟ 3

ΤΕΧΝΟΛΟΓΙΚΗ ΒΑΣΗ

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα αναπτύξουμε το τεχνικό κομμάτι της Unity και του Maya, όσον αφορά την αρχιτεκτονική τους και τις μονάδες που τις αποτελούν, καθώς και τα χαρακτηριστικά τους.

3.2 Βασικά τεχνικά χαρακτηριστικά της Unity 3D

Η Unity είναι και αυτή μία από τις κορυφαίες μηχανές παιχνιδιού. Είναι μια μηχανή παιχνιδιού που υποστηρίζει πολλές διαφορετικές πλατφόρμες και έχει ενσωματωμένο ένα IDE (Integrated development environment), ολοκληρωμένο περιβάλλον ανάπτυξης, το οποίο αναπτύχθηκε από τη Unity Technologies. Χρησιμοποιείται για την ανάπτυξη παιχνιδιών στο διαδίκτυο, σε υπολογιστές, σε κονσόλες και σε φορητές συσκευές. Υποστηρίζει την ανάπτυξη για iOS, Android, Windows, BlackBerry 10, OS X, PC, MAC, Linux, web browsers, Flash, PlayStation 3, PlayStation Vita, Xbox 360, Windows Phone 8, and Wii U. Δύο εκδόσεις της μηχανής του παιχνιδιού είναι διαθέσιμες, η Unity και η Unity Pro.

Navigation and Pathfinding

Σε πολλά παιχνίδια ένας χαρακτήρας πρέπει να είναι σε θέση να ταξιδέψει αυτόματα από την τρέχουσα θέση του σε έναν επιθυμητό προορισμό. Για παράδειγμα μπορούμε να ελέγχουμε έναν χαρακτήρα κάνοντας κλικ σε ένα σημείο-στόχο στον κόσμο του παιχνιδιού, ή ίσως ένας NPC εχθρός μπορεί να χρειαστεί να βρεί μια διαδρομή για να παρακολουθήσει τον παίκτη. Σε ορισμένες περιπτώσεις αυτό είναι ένα απλό θέμα κίνησης σε μια ευθεία γραμμή ή κατά μήκος μιας προκαθορισμένης διαδρομής, αλλά υπάρχουν επίσης κόσμοι παιχνιδιού που βασίζονται γύρω από κτίρια, δάση ή άλλες ρυθμίσεις, όπου η διαδρομή προς το στόχο δεν είναι τόσο άμεση. Σε παιχνίδια όπως αυτά, η λογική ελέγχου είναι απαραίτητη για να επιτρέψει στο χαρακτήρα να λαμβάνει αποφάσεις σε κάθε σημείο κατά μήκος της διαδρομής του για να επιλέξει το επόμενο βήμα του. Η διαδικασία της επιλογής των κινήσεων ενός χαρακτήρα σε όλο τον κόσμο το παιχνιδιού σαν αυτό είναι γνωστή ως πλοήγηση (navigation).

Υπάρχουν διάφορες τεχνικές που μπορούν να χρησιμοποιηθούν για την πλοήγηση ενός χαρακτήρα στον προορισμό του, μερικές από τις οποίες περιλαμβάνουν ένα στοιχείο της δοκιμής και του λάθους. Συχνά όμως η επιθυμητή συμπεριφορά είναι να κάνουμε το χαρακτήρα να φαίνεται να σχεδιάζει την πορεία του με έξυπνο τρόπο, λαμβάνοντας την πιο άμεση και συμφέρουσα διαδρομή. Αυτό απαιτεί ένα κομμάτι της τεχνητής νοημοσύνης (AI) για να καθορίσει τον καλύτερο τρόπο για να πάει από το σημείο εκκίνησης στο σημείο προορισμού. Αυτό το έργο στο παιχνίδι AI είναι γνωστό ως pathfinding και παρόλο που μπορεί να είναι αρκετά τεχνικό, η Unity κάνει κάποιες κοινές περιπτώσεις να είναι πολύ εύκολες χρησιμοποιώντας το ενσωματωμένο σύστημα pathfinding.

Built-in Pathfinding

Η Unity μας βοηθά να φέρουμε γρήγορα τη σκηνή μας στη ζωή με το αυτόματο πλέγμα πλοήγησης (NavMesh). Τα NavMeshes περιγράφουν τα όρια του κάθε χώρου πλοήγησης στο παιχνίδι μας και χρησιμοποιούνται κατά το χρόνο εκτέλεσης για την διερεύνηση πορείας. Ορίζουμε (bake) τα δεδομένα πλοήγησης στον επεξεργαστή, και η Unity αναλαμβάνει την εύρεση της υψηλής απόδοσης μονοπατιού και την προσομοίωση πλήθους κατά το χρόνο εκτέλεσης. Τα εμπόδια αναγνωρίζονται από τη Unity και ο ήρωας είναι σε θέση να αντιδράσει δυναμικά σε μεταβαλλόμενα περιβάλλοντα.

Βασικά πλεονεκτήματα της Unity

Η Unity διαφέρει όσον αφορά την ευκολία χρήσης και το εύρος της στήριξης των εργαλείων της βιομηχανίας. Η εισαγωγή μοντέλων, υφών, ήχου, κώδικα, καλλιτεχνικών και άλλων στοιχεία στο project που υλοποιούμε στη Unity είναι αποτελεσματική και εύκολη, καθώς χρειάζεται μόνο να τα αποθηκεύσουμε στο φάκελο του project και εισάγονται αυτόματα. Μπορούμε να τροποποιήσουμε τα στοιχεία ανά πάσα στιγμή, και οι αλλαγές στο παιχνίδι είναι άμεσες. Η επέκταση API του επεξεργαστή της Unity δίνει τον πλήρη έλεγχο για το πώς θέλουμε να εισάγονται τα στοιχεία μας.

Η Unity μπορεί να εισάγει 3D μοντέλα, οστά (bones), και κινούμενα σχέδια (animations) από σχεδόν οποιαδήποτε 3D εφαρμογή. Για κάθε αποθηκευμένη αλλαγή στο Maya, 3ds Max, Modo, Cinema 4D, Cheetah3D ή στο Blender, η Unity αμέσως επανεισάγει το ανανεωμένο asset-στοιχείο και εφαρμόζει αλλαγές σε ολόκληρο το project.

Χειρίζεται τέλεια ανά pixel την εμφάνιση των γραμματοσειρών Unicode TrueType. Μπορούμε να εισάγουμε οποιαδήποτε γραμματοσειρά TTF και να εμφανίζεται εξαιρετικά το κειμένο. Υποστηρίζονται οι συμβολοσειρές Unicode και η είσοδος IME.

Είναι αποτελεσματική στο χειρισμό των υφών. Μετατρέπει έναν Height-map σε Normal-map: κάθε υφή αυτόματα μετατρέπεται σε έναν κανονικό χάρτη, ακόμα και αν

αλλάξουμε τα αρχεία της εικόνας. Υποστηρίζει αρκετές διαφορετικές μεθόδους παραγωγής mipmap και υψηλής ποιότητας: Λεπτομερή Fade, Φίλτρα Kaiser, και Διόρθωση Γάμμα.

Αποθηκεύοντας τα πολλαπλών στρώσεων αρχεία μας στο Photoshop, η Unity συμπιέζει αυτόματα τις εικόνες με υψηλή ποιότητα. Περιλαμβάνει 5 διαφορετικές προεπιλογές για να μας αφήσει να ρυθμίσουμε γρήγορα τις υφές μας. Μπορούμε να παρακάμψουμε το μέγεθος και τις ρυθμίσεις συμπίεσης ανά πλατφόρμα, ώστε να μπορούμε να κρατήσουμε ένα αρχείο πηγή.

Μπορεί να εισάγει οποιαδήποτε μορφή ήχου που υποστηρίζεται από FMOD. Το FMOD επιτρέπει να έχουμε εύκολο και σταθερό ήχο μεταξύ όλων των υποστηριζόμενων πλατφόρμων της Unity. Ο ήχος μπορεί να μετατραπεί εσωτερικά και να κατανεμηθεί ως Ogg Vorbis, για να μειωθεί το μέγεθος του αρχείου του δημοσιευμένου παιχνιδιού μας.

Υποστηρίζει Allegorithmic ουσίες, οι οποίες είναι assets-στοιχεία που μπορούν να έχουν πολλαπλές εξόδους για να δημιουργήσουν ολοκληρωμένες υφές με βάση το ίδιο σύνολο παραμέτρων, τα οποία μπορούμε να τα βρούμε στο Asset Store. Μπορούμε να τοποθετήσουμε τα αρχεία κατ 'ευθείαν στο project μας ρυθμίζοντας και προσαρμόζοντας τις παραμέτρους τους στο εσωτερικό του Editor της Unity. Έχουμε άμεση πρόσβαση στα εισαγόμενα στοιχεία, ώστε να μπορούμε να τα τροποποιήσουμε μέσω κώδικα στο Unity. Η αναζήτηση και η διαχείριση των στοιχείων είναι γρήγορη και εύκολη μέσα από τη Unity, και είναι δυνατή η προεπισκόπηση σε μεγάλα και σύνθετα έργα με το πρόγραμμα περιήγησης.

Η Unity αναγνωρίζει τρισδιάστατους χαρακτήρες από πολλά διαφορετικά σχεδιαστικά προγράμματα, τα οποία είτε ανανγωρίζει αμέσως είτεται μετατρέπει σε αρχεία .fbx ή .collada. Τα προγράμματα τα οποία υποστηρίζει είναι τα Maya (.mb & .ma), 3D Studio Max (.max), Cheetah 3D (.jas), Cinema 4D (.c4d), Blender (.blend), modo (.lxo), Autodesk FBX, COLLADA, Carrara, Lightwave, XSI 5 (.x), SketchUp Pro, Wings 3D, 3D Studio (.3ds), Wavefront (.obj), και Drawing Interchange Files (.dxf), και αναγνωρίζει από αυτά το πλέγμα (Meshes), τις υφές (Textures), τα κινούμε σχέδια (Anims), και τα κόκαλα (Bones).

3D Package Support

	Meshes	Textures	Anims	Bones
Maya .mb & .ma[1]	✓	✓	✓	✓
3D Studio Max .max[1]	✓	✓	✓	✓
Cheetah 3D .jas[1]	✓	✓	✓	✓
Cinema 4D .c4d[1] [3]	✓	✓	✓	✓
Blender .blend[1]	✓	✓	✓	✓
modo .lxo[2]	✓	✓	✓	
Autodesk FBX	✓	✓	✓	✓
COLLADA	✓	✓	✓	✓
Carrara[1]	✓	✓	✓	✓
Lightwave[1]	✓	✓	✓	✓
XSI 5.x[1]	✓	✓	✓	✓
SketchUp Pro[1]	✓	✓		
Wings 3D[1]	✓	✓		
3D Studio .3ds	✓			
Wavefront .obj	✓			
Drawing Interchange Files .dxf	✓			

[1] Import uses the application's FBX exporter. Unity then reads the FBX file.
[2] Import uses the application's COLLADA exporter. Unity then reads the COLLADA file.
[3] Cinema4D 10 has a buggy FBX exporter. Please see here for workarounds.

Οι μορφές εικόνων που υποστηρίζονται ποικίλουν. Τα αρχεία .psd και .tiff που με στρώματα στο Photoshop εισάγονται αυτόματα ως ένα στρώμα στη Unity. Αρχεία σε μορφή

.JPEG, .PNG, .GIF, .BMP, .TGA, .IFF, .PICT και πολλές άλλες μορφές εικόνων υποστηρίζονται επίσης από τη Unity.

Οι μορφές βίντεο και ήχων που υποστηρίζονται είναι επίσης πολλών μορφών. Τα αρχεία MP3 και Ogg Vorbis .ogg υποστηρίζονται εγγενώς, ανάλογα με την πλατφόρμα. Σε κινητές πλατφόρμες ο ήχος μετατρέπεται σε MP3 για να έχουμε πλήρη η αποσυμπίεση hardware. Τα αρχεία της μορφής AIFF, WAV και πολλές άλλες μορφές ήχου υποστηρίζονται, και είναι ιδανικά για ηχητικά εφέ, των οποίων η συμπίεση είναι προαιρετική και μπορεί να ρυθμιστεί στη Unity. Τα αρχεία MOD, IT, S3M, και XM tracker υποστηρίζονται πλήρως, ενώ τα Ogg Theora βίντεο υποστηρίζονται εγγενώς. Τα βίντεο MOV, AVI, ASF, MPG, MPEG και MP4VIDEO μετατρέπονται από τη Unity με ένα ρυθμιζόμενο bitrate.

Άλλα είδη αρχείων που υποστηρίζονται είναι μορφής XML και αρχεία κειμένου με .xml και .txt προεκτάσεις τα οποία μπορούν να χρησιμοποιηθούν σε χρόνο εκτέλεσης. Άλλες μορφές αρχείων, όπως RTF και DOC, μπορούν να χρησιμοποιηθούν για σημειώσεις και λίστες στο project.

3D formats

Η εισαγωγή τρισδιάστατων αρχείων στη Unity μπορεί να επιτευχθει από δύο κύρια είδη αρχείων: από εξαγόμενα αρχεία 3D μορφής όπως .FBX ή .OBJ, ή από αναγνωρίσιμα 3D αρχεία των εφαρμογών σε αυτούσια μορφή, όπως οι μορφές αρχείων .MAX και .Blend από το 3D Studio Max ή Blender. Και με τους δύο τρόπους μπορούμε να πάρουμε το περιεχόμενο των αρχείων στη Unity, αλλά υπάρχουν περιορισμοί οι οποίοι πρέπει να ληφθούν υπόψιν ως προς τον τρόπο που θα επιλέξουμε.

Η Unity μπορεί να διαβάσει εξαγόμενα αρχεία της μορφής .FBX, .dae (Collada), .3DS, .dxf και .obj. Οι εξαγωγείς FBX και Collada μπορούν να υποστηρίζουν πολλές εφαρμογές. Τα πλεονεκτήματα των εξαγόμενων αρχείων είναι ότι εξάγουμε μόνο τα δεδομένα που χρειαζόμαστε, χωρίς περιττές πληροφορίες, είναι επαληθεύσιμα καθώς είναι διαθέσιμη η επανεισαγωγή τους σε 3D προγράμματα, είναι σε γενικές γραμμές μικρότερα αρχεία, ενθαρρύνεται η ατομική προσέγγιση, και υποστηρίζονται 3D προγράμματα των οποίων η μορφή των αρχείων δεν υποστηρίζεται άμεσα. Τα μειονεκτήματα όμως είναι ότι μπορεί να είναι πιο αργό το pipeline, και είναι ευκολότερο να χαθούν τα ίχνη των εκδόσεων μεταξύ της πηγής και τα δεδομένα του παιχνιδιού (που εξάγονται από τον FBX).

Η Unity μπορεί επίσης να εισάγει αυτούσια τα αρχεία των 3D εφαρμογών Max, Maya, Blender, Cinema4D, Modo, Lightwave και Cheetah3D, τα οποία έχουν αντίστοιχα τη μορφή .MAX, .MB, .MA κλπ. Τα πλεονεκτήματα της εισαγωγής αυτής της μορφής αρχείων είναι ότι έχουμε απλότητα και γρήγορη διαδικασία επανάληψης, αποθηκεύουμε το αρχείο προέλευσης και η Unity το επαναεισάγει. Τα μειονεκτήματα είναι ότι πρέπει να εγκατασταθεί ένα νόμιμο

αντίγραφο του λογισμικού που χρησιμοποιείται σε όλους τους υπολογιστές που χρησιμοποιούν το Unity, τα αρχεία μπορεί να γεμίσουν με περιττά στοιχεία, τα μεγάλα αρχεία μπορεί να επιβραδύνουν τις ενημερώσεις της Unity, και λιγότερη επικύρωση σημαίνει και δυσκολότερη αντιμετώπιση προβλημάτων.

3.2.1 Απόδοση

Η μηχανή γραφικών χρησιμοποιεί Direct3D (Windows, Xbox 360), OpenGL (Mac, Windows, Linux), OpenGL ES (Android, iOS), και ιδιόκτητες APIs (κονσόλες).

Υποστηρίζεται η χαρτογράφηση χτυπήματος (bump mapping), η χαρτογράφηση αντανάκλασης (reflection mapping), η παράλληλη χαρτογράφηση (parallax mapping), οι δυναμικές σκιές χρησιμοποιώνας χάρτες σκιών, και επεξεργάζοναι κατάλληλα οι υφές ωστε να αποδίδουν σε πλήρης οθόνη.

Η Unity υποστηρίζει γραφικά μοντέλα (art assets) και μορφές αρχείων από τα προγράμματα 3ds Max, Maya, Softimage, Blender, modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks and Allegorithmic Substance. Τα μοντέλα αυτών τον προγραμμάτων μπορούν να προστεθούν στο παιχνίδι και να διαχειριστούν μέσα από τη Unity.

Εμείς χρησιμοποιήσαμε το Maya για να φτιάξουμε τον πιγκουίνο, τα νομίσματα, το ιγκλού, το μαγικό σμαράγδι, τα ψάρια και το θησαυρό. Ο βίκινγκ, το κανόνι, η φάλαινα και οι βράχοι δημιουργήθηκαν με το 3ds Max αλλά τα πήραμε έτοιμα από το Asset Store της Unity.

Στο Adobe Photoshop φτιάξαμε όλες τις υφές (textures) των μοντέλων, τα μενού, τα κουμπιά, τις ενδείξεις με το σκορ και τις ζωές, τις καρτέλες με τις επιλογές και τα μηνύματα που παρουσιάζονται στο παιχνίδι και τα φόντο.

Η γλώσσα ShaderLab χρησιμοποιείται για σκιές, υποστηρίζοντας προγραμματισμό γραμμένο σε GLSL ή Cg. Μια σκιά μπορεί να περιλαμβάνει πολλαπλές παραλλαγές αλλά η Unity ανιχνεύει την καλύτερη παραλλαγή για την τρέχουσα κάρτα γραφικών και αν καμία δεν είναι συμβατή βρίσκει μια εναλλακτική σκιά θυσιάζοντας ίσως τα χαρακτηριστικά για την απόδοση.

3.2.2 Κώδικας

Η μηχανή είναι χτισμένη πάνω σε Mono, μια υλοποίηση του ανοιχτού κώδικα .NET Framework. Οι προγραμματιστές μπορούν να χρησιμοποιήσουν UnityScript (μια γλώσσα της Unity εμπνευσμένη από τη σύνταξη της ECMAScript, η οποία αναφέρεται ως JavaScript από το λογισμικό), C#, ή Boo (η οποία έχει εμπνευστεί από τη σύνταξη της Python). Η Unity διαθέτει τη προσαρμοσμένη έκδοση του MonoDevelop για την εκσφαλμάτωση του κώδικα. Εμείς χρησιμοποιήσαμε τη JavaScript για την ανάπτυξη του παιχνιδού. Η JavaScript (JS) είναι διερμηνευμένη γλώσσα προγραμματισμού για ηλεκτρονικούς υπολογιστές.

Η JavaScript είναι μια γλώσσα σεναρίων που βασίζεται στα πρωτότυπα, είναι δυναμική, και έχει πρώτης τάξης συναρτήσεις. Η σύνταξή της είναι επηρεασμένη από τη C. Η JavaScript αντιγράφει πολλά ονόματα και συμβάσεις ονοματοδοσίας από τη Java, αλλά γενικά οι δύο αυτές γλώσσες δε σχετίζονται και έχουν πολύ διαφορετική σημασιολογία. Οι βασικές αρχές σχεδιασμού της JavaScript προέρχονται από τις γλώσσες προγραμματισμού Self και Scheme. Είναι γλώσσα βασισμένη σε διαφορετικά προγραμματιστικά παραδείγματα, υποστηρίζοντας αντικειμενοστρεφή, προστακτικό και συναρτησιακό στυλ προγραμματισμού. Το πρότυπο της γλώσσας κατά τον οργανισμό τυποποίησης ECMA ονομάζεται ECMAscript. Η JavaScript επισημοποιήθηκε με την τυπική γλώσσα ECMAScript και χρησιμοποιείται κυρίως ως μέρος ενός προγράμματος περιήγησης στο web (client-side JavaScript).

3.2.3 Εντοπισμός ήρωα

Η Unity περιλαμβάνει το Unity Asset Server, στο οποίο είναι διαθέσιμα και έτοιμα παιχνίδια, χαρακτήρες, κώδικες και ό,τι μπορεί να χρειαστεί ένας προγραμματιστής στο παιχνίδι του. Από εκεί βρήκαμε και εμείς τον βίκινγκ, το κανόνι, τη φάλαινα, τους βράχους και τα δέντρα.

Χρησιμοποιεί PostgreSQL ως backend, ένα σύστημα ήχου που είναι ενσωματωμένο στη βιβλιοθήκη FMOD (με δυνατότητα αναπαραγωγής Ogg Vorbits συμπιεσμένου ήχου), το Theora codec για την αναπαραγωγή βίντεο, το έδαφος (terrain) και τη βλάστηση της μηχανής (η οποία υποστηρίζει δέντρα billboarding), ενσωματωμένο lightmapping και global φωτισμό με Beast, multiplayer μέσω δικτύωσης χρησιμοποιώντας RakNet, και ενσωματωμένο πλέγμα πλοήγησης pathfinding.

3.2.4 Πλατφόρμες

Η Unity υποστηρίζει την ανάπτυξη σε πολλαπλές πλατφόρμες. Μέσα σε ένα έργο, οι προγραμματιστές έχουν τον έλεγχο παράδοσης σε κινητές συσκευές, προγράμματα περιήγησης στο Web, υπολογιστές και κονσόλες. Επιτρέπει επίσης τη συμπίεση υφής και τις ρυθμίσεις ανάλυσης για κάθε πλατφόρμα που υποστηρίζει το παιχνίδι.

Σήμερα οι υποστηριζόμενες πλατφόρμες περιλαμβάνουν Xbox One, BlackBerry 10, τα Windows 8, Windows Phone 8, Windows, Mac, Linux, Android, iOS, Unity Web Player, Adobe Flash, PlayStation 3, Xbox 360, Wii U και το Wii. Αν και δεν έχει επιβεβαιωθεί επίσημα, Unity υποστηρίζει επίσης το PlayStation Vita. Οι προσεχώς πλατφόρμες περιλαμβάνουν PlayStation 4 και Xbox One.

3.2.5 Φυσική

Η Unity έχει ενσωματωμένη τη μηχανή φυσικής PhysX της Nvidia (πρώην Ageia) με πρόσθετη υποστήριξη για την προσομοίωση ρούχων σε πραγματικό χρόνο, για skinned

meshes, thick ray casts, και collision layers. Διαθέτει ενσωματωμένη υποστήριξη για τη μηχανή φυσικής Box2D για 2D παιχνίδια.

3.2.6 Έκδοση

Εμείς χρησιμοποιήσαμε την έκδοση Unity 4, η οποία κυκλοφόρησε στα τέλη του 2012 και περιλαμβάνει αρκετές νέες προσθήκες. Περιλαμβάνει πολλές ενημερώσεις με επιπλέον χαρακτηριστικά, όπως το νέο Retained GUI. Τα νέα χαρακτηριστικά περιλαμβάνουν την υποστήριξη της DirectX 11 και του Mecanim animation. Οι Mobile βελτιώσεις γραφικών περιλαμβάνουν σκιές σε πραγματικό χρόνο, skinned mesh instancing, η δυνατότητα χρήσης κανονικών χαρτών (normal maps) κατά τη δημιουργία των lightmaps και GPU profiler. Η ανάπτυξη του Adobe Flash add-on έχει απελευθερωθεί επίσης με την Unity 4.0.

Περιλαμβάνει επίσης μια νέα επιλογή εγκατάστασης για να δημοσιεύσει τα παιχνίδια σε υπολογιστές Linux. Ενώ η ανάπτυξη add-on μπορεί δυνητικά να λειτουργήσει με διάφορες μορφές του Linux, η ανάπτυξη εστιάζεται κυρίως στο Ubuntu για την πρωτοβάθμια απελευθέρωσή του. Αυτή η επιλογή εγκατάστασης παρέχεται σε όλους τους χρήστες της Unity 4 χωρίς επιπλέον κόστος. Από την έκδοση 4 η Unity λειτουργεί σε συνεργασία με το Facebook για να ξεκινήσει μια κοινωνική πλατφόρμα μέσω του Unity Web Player .

Η Unity 4 εκτός από τις αναβαθμίσεις, εμπεριέχει όλα τα στοιχεία της προηγούμενης έκδοσής της, όπως το Shuriken συστήμα σωματιδίων, το navmesh για pathfinding και την αποφυγή εμποδίων, το γραμμικό χώρο, το φωτισμό, το HDR rendering, το multi-threaded rendering, τους ανιχνευτές φωτός, την ανάπτυξη Google Native Client, το Adobe Flash Player add-on preview, το GPU profiler, και τον κατευθυνόμενο φωτεισμό (directional lightmaps).

3.2.7 Mecanim

Το Mecanim είναι η τεχνολογία animation της Unity, με την οποία δίνεται ροή και φυσική κίνηση στους χαρακτήρες. Περιλαμβάνει εργαλεία για την δημιουργία των state machines, blend trees, IK rigging, και την αυτόματη αναδιάταξη των animations μέσα από τον Unity editor. Επιπλέον μια σειρά από κινούμενα σχέδια retargetable είναι διαθέσιμα στο Asset Store. Πολλά από αυτά τα αρχεία animation χρησιμοποιούν καταγραφή της κίνησης και παρέχονται χωρίς κόστος από τη Unity Technologies.

3.2.8 Άλλες βελτιώσεις

- Shuriken σύστημα σωματιδίων που υποστηρίζει εξωτερικές δυνάμεις, bent normals και automatic culling
- Υποστήριξη 3D υφής
- Πλοήγηση: δυναμικά εμπόδια και προτεραιότητα αποφυγής
- Σημαντικές βελτιστοποιήσεις στην απόδοση της UnityGUI και τη χρήση της μνήμης
- Δυναμικές γραμματοσειρές σε όλες τις πλατφόρμες με HTML-όπως η σήμανση

- Απομακρυσμένος εντοπισμός σφαλμάτων Unity Web Player
- Παράθυρο νέων ροών εργασιών
- Επαναληπτική δημιουργία lightmap
- Ροές εργασίας που βασίζονται σε components
- Extensible επιθεωρητές για προσαρμοσμένες κλάσεις
- Βελτιωμένος Cubemap αγωγός εισαγωγής
- Βελτιώσεις δεδομένων Γεωμετρίας για την τεράστια μνήμη και την απόδοση εξοικονόμησης
- Τα πλέγματα μπορούν να κατασκευαστούν από μη τριγωνική γεωμετρία, έτσι ώστε να καθιστούν τα σημεία και τις γραμμές αποτελεσματικά
- Αναζήτηση, ζωντανή προεπισκόπηση και αγορά περιουσιακών στοιχείων από το Asset Store από το παράθυρο του project.

3.2.9 Κοινότητα

Η κοινότητα της Unity είναι πολύ ενεργή, καθώς έχει 2 εκατομμύρια εγγεγραμμένους προγραμματιστές, οι οποίοι βοηθάνε πολύ στην ανάπτυξη ενός παιχνιδιού και στη γνωριμία με την μηχανή Unity λύνοντας τυχόν απορίες, δίνοντας ιδέες και δείχνοντας παραδείγματα από δικά τους παιχνίδια για να καθοδηγήσουν νέους χρήστες. Επιπλέον η Unity παρέχει δωρεάν εκπαίδευση σε απευθείας σύνδεση στην ιστοσελίδα της. Υπάρχουν επίσης και αναλυτικές οδηγίες για τις κλάσεις της μηχανής του παιχνιδιού και τις διασυνδέσεις, οι οποίες είναι διαθέσιμες στο διαδίκτυο.

3.3 Η δομή της Unity

Το κάθε παιχνίδι της Unity είναι ένα project που περιλαμβάνει μία ή περισσότερες σκηνές (scenes) στις οποίες οργανώνεται. Η κάθε σκηνή αποτελείται από ένα σύνολο αντικειμένων, και μπορεί να περιλαμβάνει την πίστα του παιχνιδιού, γραφικές διεπαφές, ή το συνδυασμό τους. Το κάθε αντικείμενο που βρίσκεται στην σκηνή αποτελεί ένα GameObject. Οτιδήποτε περιέχει το παιχνίδι είναι GameObjects τα οποία αποτελούνται από Components (συστατικά). Η Unity χρησιμοποιεί μια component-based αρχιτεκτονική, όπου ένας αυθαίρετος αριθμός components συνδέεται με κάθε GameObject για να καθορίσει τη συμπεριφορά του αντικειμένου και τις ιδιότητές του. Χωρίς αυτά, το αντικείμενο είναι μόνο ένα γραφικό στοιχείο και δεν μπορεί να κάνει τίποτα ούτε να αλληλεπιδράσει με τους χαρακτήρες.

3.3.1 GameObject

Τα GameObjects είναι τα πιο σημαντικά αντικείμενα της Unity. Είναι πολύ σημαντικό να κατανοήσουμε τι είναι ένα GameObject και πώς μπορεί να χρησιμοποιηθεί. Κάθε αντικείμενο στο παιχνίδι είναι ένα GameObject. Ωστόσο τα GameObjects δεν κάνουν τίποτα από μόνα τους. Χρειάζονται ειδικές ιδιότητες πριν γίνουν ένας χαρακτήρας, ένα περιβάλλον, ή ένα ειδικό εφέ. Όμως κάθε ένα από αυτά τα αντικείμενα κάνει πολλά διαφορετικά πράγματα, και για να τα ξεχωρίζουμε έχουμε τα Components. Τα GameObjects είναι αντικείμενα που περιέχουν components. Όλα τα αντικείμενα στο παιχνίδι είναι GameObjects που περιέχουν διαφορετικά συστατικά. Ανάλογα με το είδος του αντικειμένου που θέλουμε να δημιουργήσουμε προσθέτουμε διαφορετικούς συνδυασμούς των στοιχείων στο GameObject. Μπορούμε επίσης να φτιάξουμε το δικό μας χρησιμοποιώντας Components Scripts. Ένα άδειο GameObject εξακολουθεί να περιέχει ένα όνομα, μια ετικέτα, και ένα στρώμα. Κάθε GameObject περιέχει επίσης ένα Component Transform που καθορίζει τη θέση, την περιστροφή, την κλίμακα, και τον γονέα. Μπορεί να δημιουργηθεί ένα Component χωρίς GameObject, αλλά δεν θα είναι σε θέση να χρησιμοποιηθεί μέχρι να εφαρμοστεί σε ένα GameObject.

3.3.2 Prefabs

Ένα Prefab είναι ένα είδος του περιουσιακού στοιχείου–asset, ένα επαναχρησιμοποιήσιμο GameObject αποθηκευμένο στο Project View. Τα Prefabs μπορούν να εισαχθούν σε οποιοδήποτε αριθμό σκηνής, πολλές φορές ανά σκηνή. Όταν προσθέσουμε ένα prefab σε μια σκηνή, δημιουργούμε μια παρουσία του. Όλες οι παρουσίες των Prefabs συνδέονται με το αρχικό Prefab και είναι ουσιαστικά κλώνοι του. Ανεξάρτητα από τον αριθμό των κλώνων που υπάρχουν στο παιχνίδι, όταν κάνουμε αλλαγές στο Prefab θα δούμε την αλλαγή να εφαρμόζεται σε όλους τους κλώνους.

Κληρονομικότητα σημαίνει ότι κάθε φορά που η πηγή Prefab αλλάζει, οι αλλαγές αυτές εφαρμόζονται σε όλες τα συνδεδεμένα GameObjects. Ωστόσο είναι δυνατόν να αλλάξουμε τις ιδιότητες ενός μόνο GameObject διατηρώντας ανέπαφο το σύνδεσμο, πανωγράφοντας τις τιμές του οι οποίες στη συνέχεια δεν θα επηρεαστούν από τις αλλαγές της πηγής. Επίσης είναι δυνατή η διακοπή του συνδέσμου διατηρώντας το αντικείμενο.

3.3.3 Interface

Το κύριο παράθυρο του επεξεργαστή αποτελείται από πολλές καρτέλες παραθύρων, που ονομάζονται Προβολές. Υπάρχουν διάφοροι τύποι των Προβολών.

Στο παράθυρο Scene βλέπουμε τη σκηνή του παιχνιδιού όπου μπορούμε να την δημιουργήσουμε και να την επεξεργαστούμε. Οι Σκηνές περιέχουν τα αντικείμενα του παιχνιδιού. Μπορούν να χρησιμοποιηθούν για να δημιουργήσουν ένα κύριο μενού, μεμονωμένα επίπεδα, και οτιδήποτε άλλο. Κάθε μοναδικό αρχείο σκηνής είναι ένα μοναδικό επίπεδο. Σε κάθε σκηνή τοποθετούμε το περιβάλλον, τα εμπόδια, τις διακοσμήσεις, το σχεδιασμό και την οικοδόμηση του παιχνιδιού σε κομμάτια.

Στην Hierarchy βρίσκονται όλα τα αντικείμενα της κάθε σκηνής, που μπορεί να είναι οργανωμένα σε επίπεδα δημιουργώντας σχέσεις γονέα-παιδιού που το παιδί θα κληρονομήσει την κίνηση και την περιστροφή του γονέα, και μας δίνει τη δυνατότητα να προσθέσουμε ή να αφαιρέσουμε αντικείμενα της σκηνής.

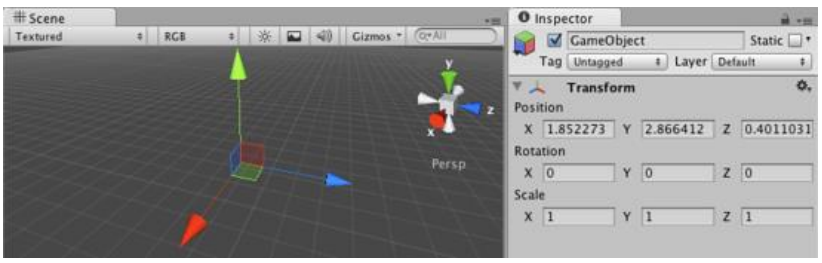
Στο Project βρίσκονται όλα τα στοιχεία-assets που μπορούμε να χρησιμοποιήσουμε στο παιχνίδι, κώδικες, ήρωες, αντικείμενα, φωτισμός, περιβάλλον, κάμερες, κτλ.

Με το Toolbar μπορούμε να επεξεργαστούμε το κάθε αντικείμενο, αλλάζοντάς του θέση, μέγεθος και περιστροφή. Επίσης πατώντας “Play” εμφανίζεται το Game View στο οποίο μπορούμε να δούμε το αποτέλεσμα του έργου μας, να παρακολουθήσουμε τις τιμές, να αλληλεπιδράσουμε με το περιεχόμενό μας δοκιμάζοντας κάποια πράγματα χωρίς να προκαλέσουμε μόνιμες αλλαγές, να αλλάξουμε τις ρυθμίσεις, ακόμη και ξαναμεταγλωττίσουμε τον κώδικα. Στη συνέχεια πατώντας παύση επιστρέφουμε στη Scene View όπου μπορούμε κάνουμε τις μόνιμες αλλαγές που επιθυμούμε.

Όπως αναφέραμε, τα παιχνίδια αποτελούνται από πολλαπλά GameObjects που περιέχουν πλέγματα, scripts, ήχους ή άλλα γραφικά στοιχεία, όπως φώτα. Ο Inspector εμφανίζει αναλυτικές πληροφορίες σχετικά με επιλεγμένο GameObject, συμπεριλαμβανομένων όλων των στοιχείων που συνδέονται, τις ιδιότητές του, τα χαρακτηριστικά του, τα components από τα οποία αποτελείται, καθώς και τις τιμές που περιέχει. Εδώ μπορούμε να τροποποιήσουμε τη λειτουργικότητα των GameObjects στη σκηνή μας. Κάθε στοιχείο που εμφανίζεται στον Inspector μπορεί να τροποποιηθεί άμεσα. Ακόμα και οι μεταβλητές του κώδικα μπορούν να αλλάξουν χωρίς την τροποποίηση του ίδιου του κώδικα. Μπορούμε να χρησιμοποιήσουμε τον Inspector για να αλλάξουμε τις μεταβλητές κατά το χρόνο εκτέλεσης για να πειραματιστούμε και να βρουμε το κατάλληλο gameplay για το παιχνίδι. Σε ένα κώδικα αν ορίσουμε μια δημόσια μεταβλητή ενός τύπου αντικειμένου (όπως GameObject ή Transform), μπορούμε να ορίσουμε στον Inspector ένα GameObject ή Prefab για την αντιστοίχηση.

3.3.4 Transform Component

Είναι αδύνατο να δημιουργηθεί ένα GameObject χωρίς Component Transform.

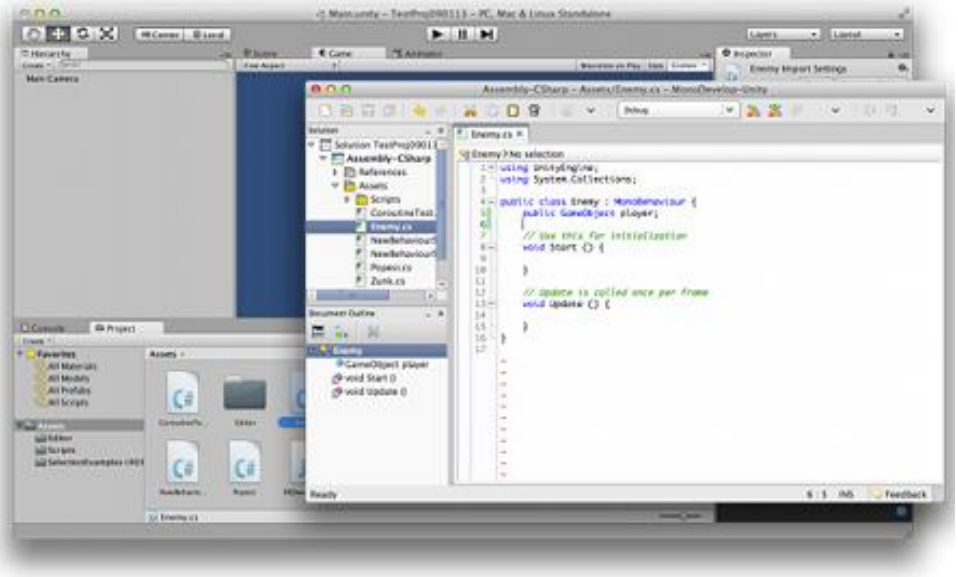


Αυτό συμβαίνει επειδή το Transform υπαγορεύει που βρίσκεται το GameObject, και πώς περιστρέφεται και κλιμακώνεται. Χωρίς Component Transform, η GameObject δεν θα έχουν θέση στον κόσμο. Το Component Transform είναι ένα από τα πιο σημαντικά συστατικά, δεδομένου ότι το σύνολο των ιδιοτήτων του μετασχηματισμού του GameObject είναι δυνατό με τη χρήση του παρόντος Component. Ορίζει τη θέση του GameObject σε X, Y, και Z συντεταγμένες στο χώρο, την περιστροφή του γύρω από τους X, Y, και Z άξονες μετρούμενη σε μοίρες και την κλίμακα του κάθε αντικειμένου στον κόσμο/Scene View του παιχνιδιού κατά μήκος των X, Y, και Z αξόνων. Η τιμή "1" είναι το αρχικό μέγεθος με το οποίο είχε εισαχθεί το αντικείμενο. Αυξάνοντας το μέγεθος του αντικειμένου αυξάνεται και η τιμή ανάλογα με τον άξονα που αυξάνεται.

Το Transform Component επιτρέπει επίσης μια έννοια που ονομάζεται Parenting. Το Parenting είναι μια από τις πιο σημαντικές έννοιες της Unity. Όταν ένα GameObject είναι γονέας ενός άλλου GameObject, το παιδί GameObject θα κινηθεί, περιστραφεί και κλιμακωθεί ακριβώς όπως ο γονέας του. Κάθε αντικείμενο μπορεί να έχει πολλαπλά παιδιά, αλλά μόνο ένα γονέα. Όλες οι ιδιότητες ενός Transform μετριούνται σε σχέση με το Transform του γονέα. Εάν το Transform δεν έχει γονέα, οι ιδιότητες μετριούνται σε σχέση με τις διαστάσεις του κόσμου του παιχνιδιού (World Space).

3.3.5 Scripting

Το Scripting είναι ένα βασικό συστατικό σε όλα τα παιχνίδια. Ακόμα και το πιο απλό παιχνίδι χρειάζεται κώδικα για να ανταποκριθεί στην εισαγωγή από τον παίκτη και να μεριμνήσει για να συμβούν τα γεγονότα στο gameplay όταν πρέπει. Με τον κώδικα μπορούμε να δημιουργήσουμε οπτικά εφέ, να ελέγξουμε τη φυσική συμπεριφορά των αντικειμένων ή ακόμη και να εφαρμόσουμε ένα AI σύστημα στους χαρακτήρες του παιχνιδιού.



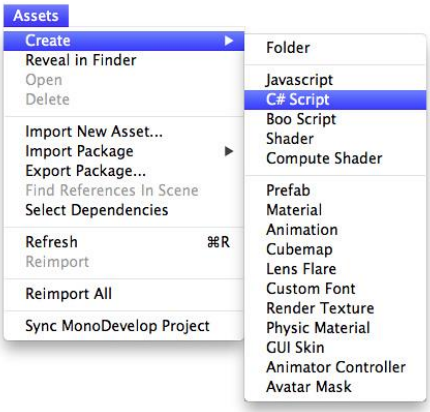
Η συμπεριφορά των GameObjects ελέγχεται από τα components που συνδέονται με αυτά. Μπορούμε να δημιουργήσουμε όμως δικά μας components ή να χειριστούμε τα ήδη υπάρχοντα με τη χρήση των scripts, προκαλώντας τα γεγονότα του παιχνιδιού, τροποποιώντας

τις ιδιότητες των Component με την πάροδο του χρόνου και αλληλεπιδρώντας με τον χρήστη με οποιοδήποτε τρόπο επιθυμούμε.

Η Unity υποστηρίζει τρεις γλώσσες προγραμματισμού:

- C# : είναι μια τυποποιημένη γλώσσα παρόμοια με την Java ή C++.
- UnityScript: μια γλώσσα που έχει σχεδιαστεί ειδικά για χρήση της Unity και διαμορφώθηκε σύμφωνα με τη JavaScript.
- Boo: μια .NET γλώσσα με παρόμοια σύνταξη με την Python. Ένα κώδικας Boo ακολουθεί περίπου την ίδια διάταξη με τον κώδικα της C#, αλλά η UnityScript λειτουργεί λίγο διαφορετικά.

Εκτός από αυτές, πολλές άλλες .NET γλώσσες μπορούν να χρησιμοποιηθούν με τη Unity αν μπορούν να καταρτίσουν ένα συμβατό DLL. Η Unity χρησιμοποιήσει το MonoDevelop, αλλά μπορούμε να επιλέξουμε οποιοδήποτε πρόγραμμα επεξεργασίας επιθυμούμε το οποίο υποστηρίζεται από αυτήν.



Σε αντίθεση με τα περισσότερα άλλα στοιχεία της Unity, τα scripts δημιουργούνται συνήθως μέσα στη Unity άμεσα. Μπορούμε να δημιουργήσουμε ένα νέο σενάριο από το «Create» στο μενού ή επιλέγοντας Assets > Create > C# Script (ή JavaScript/Boo script) από το κύριο μενού.

Το νέο script θα δημιουργηθεί σε όποιο φάκελο έχουμε επιλέξει στον πίνακα του έργου. Το όνομα του νέου αρχείου script θα επιλεγεί, ζητώντας να εισάγουμε ένα νέο όνομα. Είναι καλύτερο να εισάγουμε το όνομα του νέου script σε αυτό το σημείο και όχι μετά από επεξεργασία. Το όνομα που εισάγουμε θα χρησιμοποιηθεί για να δημιουργήσει το αρχικό κείμενο μέσα στο αρχείο.

Όταν κάνουμε διπλό κλικ σε ένα περιουσιακό στοιχείο σεναρίου της Unity, ανοίγει ένας επεξεργαστής κειμένου. Από προεπιλογή, η Unity χρησιμοποιεί το MonoDevelop, αλλά μπορούμε να επιλέξουμε οποιοδήποτε πρόγραμμα επεξεργασίας που μας αρέσει από το External Tools εργαλείων στις προτιμήσεις της Unity.

Τα αρχικά περιεχόμενα του αρχείου είναι κάτι σαν αυτό:

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {
        // Use this for initialization
```

```
        void Start () {
        }
        // Update is called once per frame
        void Update () {
        }
}
```

Ο κώδικας συνδέεται με την εσωτερική λειτουργία της Unity, εφαρμόζοντας μια κλάση που προέρχεται από την ενσωματωμένη κλάση που ονομάζεται MonoBehaviour. Μπορούμε να σκεφτούμε μια τάξη ως ένα είδος σχεδίου για τη δημιουργία ενός νέου τύπου Component που μπορεί να συνδεθεί σε GameObjects. Κάθε φορά που συνδέουμε ένα script component σε ένα GameObject, δημιουργείται ένα νέο στιγμιότυπο του αντικειμένου που καθορίζεται από το σχέδιο. Το όνομα της κατηγορίας έχει ληφθεί από το όνομα που παρέχεται όταν δημιουργήθηκε το αρχείο. Το όνομα της κλάσης και το όνομα του αρχείου πρέπει να είναι το ίδιο για να ενεργοποιηθεί το script component και να συνδεθεί με ένα GameObject.

Τα κύρια πράγματα που πρέπει να σημειωθούν, είναι οι δύο λειτουργίες που ορίζονται μέσα στην κλάση. Η συνάρτηση Update είναι το κατάλληλο μέρος για να θέσουμε τον κώδικα που θα χειριστεί την ενημέρωση πλαισίου για το GameObject. Αυτό μπορεί να περιλαμβάνει την κίνηση, προκαλώντας τις δράσεις και την ανταπόκριση στις εισόδους των χρηστών, και βασικά τίποτα που πρέπει να αντιμετωπιστεί σε βάθος χρόνου κατά τη διάρκεια του παιχνιδιού. Για να ενεργοποιήσουμε τη συνάρτηση Update πρέπει να είναι σε θέση να δημιουργήσει μεταβλητές, να διαβάσεις τις προτιμήσεις και να κάνεις τις συνδέσεις με άλλα GameObjects πριν από οποιαδήποτε δράση του παιχνιδιού. Η συνάρτηση Start θα κληθεί πριν από την έναρξη του παιχνιδιού (δηλαδή πριν κληθεί η Update για πρώτη φορά) και είναι ένα ιδανικό μέρος για να κάνουμε κάποια προετοιμασία στο παιχνίδι, όπως την αρχικοποίηση των τιμών μας η οποία δεν γίνεται χρησιμοποιώντας μια συνάρτηση κατασκευαστής.

Η αρχικοποίηση ενός αντικειμένου δεν γίνεται χρησιμοποιώντας μια συνάρτηση κατασκευαστής. Αυτό οφείλεται στο γεγονός ότι η κατασκευή των αντικειμένων γίνεται από τον επεξεργαστή και δεν λαμβάνει χώρα κατά την έναρξη του παιχνιδιού. Εάν επιχειρήσουμε να ορίσουμε ένα κατασκευαστή για ένα script component, θα παρέμβει στην κανονική λειτουργία της Unity και μπορεί να προκαλέσει σοβαρά προβλήματα με το πρόγραμμα.

Ένα σενάριο Boo ακολουθεί περίπου την ίδια διάταξη με ένα C# script, αλλά η UnityScript λειτουργεί λίγο διαφορετικά:

```
#pragma strict

function Start () {

}
```

```
function Update () {

}
```

Εδώ, οι συναρτήσεις Start και Update έχουν την ίδια έννοια, αλλά η κλάση δεν δηλώνεται ρητά. Το ίδιο το script θεωρείται πως καθορίζει την κλάση, απορρέει σιωπηρώς από το MonoBehaviour και λαμβάνει το όνομά του από το όνομα του αρχείου του script asset.

Όπως προαναφέρθηκε, ένα script ορίζει μόνο ένα προσχέδιο για ένα Component και έτσι κανένας κώδικας δε θα ενεργοποιηθεί μέχρι ένα στιγμιότυπο του script να συνδεθεί με ένα GameObject. Μπορούμε να επισυνάψουμε ένα script σύροντας το περιουσιακό στοιχείο ενός script σε ένα GameObject στον πίνακα ιεραρχίας ή στον επιθεωρητή του GameObject που έχει επιλεγεί. Υπάρχει επίσης ένα υπομενού Scripts για το Component μενού, το οποίο περιέχει όλα τα scripts που διατίθενται στο έργο, συμπεριλαμβανομένων και εκείνων που έχουμε δημιουργήσει μόνοι μας. Το αντικείμενο script μοιάζει πολύ με οποιαδήποτε άλλο Component στον Inspector.

Ο κώδικας ελέγχει τα Components, αλλά ενεργοποιείται μόνο όταν συνδεθεί με ένα GameObject. Το script αποτελεί το ίδιο ένα Component και μπορούμε να επιτρέψουμε την επεξεργασία των τιμών των μεταβλητών του μέσω του Inspector. Μόλις συνδεθεί, το script θα αρχίσει να λειτουργεί όταν πατήσουμε Play και ξεκινήσει το παιχνίδι. Μπορούμε να το ελέγξουμε αυτό με την προσθήκη του παρακάτω κώδικα στη λειτουργία εκκίνησης:

```
// Use this for initialization
void Start () {
        Debug.Log("I am alive!");
}
```

Το Debug.Log είναι μια απλή εντολή που απλά τυπώνει ένα μήνυμα στην έξοδο της κονσόλας Unity. Αν πατήσουμε τώρα Play, θα δούμε το μήνυμα στο κάτω μέρος του κύριου παραθύρου του επεξεργαστή Unity και στο παράθυρο της κονσόλας (μενού: Window> Console).

Variables

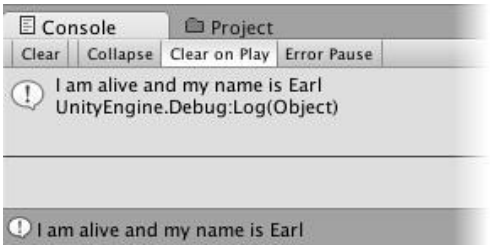
Ένα script λειτουργεί πολύ όπως και κάθε άλλο component κυρίως, και το στοιχείο για το script στον Inspector μοιάζει λίγο με τα άλλα Components στη δυνατότητα επεξεργασίας των ιδιοτήτων τους. Μπορούμε να επιτρέψουμε στις τιμές του script να είναι επεξεργάσιμες από τον Επιθεωρητή χρησιμοποιώντας μεταβλητές:

```
using UnityEngine;
using System.Collections;
```

```
public class MainPlayer : MonoBehaviour {
	public string myName;

	// Use this for initialization
	void Start () {
		Debug.Log("I am alive and my name is " + myName);
	}

	// Update is called once per frame
	void Update () {

	}
}
```

Αυτός ο κώδικας δημιουργεί ένα στοιχείο στον Inspector με την ετικέτα "My Name".

Η Unity δημιουργεί την ετικέτα στον Επιθεωρητή με την εισαγωγή ενός χώρου όπου ένα κεφαλαίο γράμμα εμφανίζεται στο όνομα της μεταβλητής. Ωστόσο, αυτό είναι καθαρά για λόγους επίδειξης και θα πρέπει πάντα να χρησιμοποιούμε το όνομα της μεταβλητής μέσα στον κώδικά μας. Εάν επεξεργαστούμε το όνομα και στη συνέχεια, πατήσουμε το πλήκτρο Play, θα δούμε ότι το μήνυμα περιλαμβάνει το κείμενο που έχουμε εισάγει.



Σε C# και Boo, πρέπει να δηλώσουμε μια μεταβλητή ως κοινή-public για να τη δούμε στην Inspector. Στην UnityScript, οι μεταβλητές είναι δημόσιες από προεπιλογή, εκτός αν καθορίσουμε ότι θα πρέπει να είναι ιδιωτικές-private:

```
#pragma strict

private var invisibleVar: int;

function Start () {

}
```

Η Unity μας αφήνει να αλλάζουμε την τιμή των μεταβλητών ενός script, ενώ το παιχνίδι βρίσκεται σε λειτουργία. Αυτό είναι πολύ χρήσιμο για να δούμε τις επιδράσεις των μεταβολών άμεσα χωρίς να χρειάζεται να σταματήσουμε και να επανεκκινήσουμε. Όταν τελειώνει το gameplay, οι τιμές των μεταβλητών θα επανέλθουν σε ό,τι ήταν πριν πατήσουμε το Play. Αυτό εξασφαλίζει ότι μπορούμε ελεύθερα να αλλάζουμε τις ρυθμίσεις του αντικειμένου μας, χωρίς το φόβο του να κάνουμε οποιαδήποτε μόνιμη βλάβη.

Accessing Components

Στον επεξεργαστή Unity, μπορούμε να κάνουμε αλλαγές σε Component ιδιότητες χρησιμοποιώντας τον Inspector. Έτσι, για παράδειγμα, μεταβολές στις τιμές θέσεως της συνιστώσας μετασχηματισμού(Transform Component) θα οδηγήσουν σε μια αλλαγή στη θέση του GameObject. Ομοίως, μπορούμε να αλλάζουμε το χρώμα του υλικού ενός Renderer ή της μάζας ενός Rigidbody με αντίστοιχη επίδραση στην εμφάνιση ή τη συμπεριφορά του GameObject. Το scripting χρησιμοποιείται κυρίως για την τροποποίηση των ιδιοτήτων των συνιστωσών για να χειραγωγήσουμε τα GameObjects. Η διαφορά, όμως, είναι ότι ένα script μπορεί να αλλάζει την τιμή ενός στοιχείου σταδιακά με την πάροδο του χρόνου ή ανάλογα με την είσοδο από το χρήστη. Με την αλλαγή, τη δημιουργία και την καταστροφή των αντικειμένων στο σωστό χρόνο, μπορεί να υλοποιηθεί οποιοδήποτε είδος του παιχνιδιού.

Η απλούστερη και πιο συνηθισμένη περίπτωση είναι όταν ένα script πρέπει να έχει πρόσβαση σε άλλα Components που συνδέονται με το ίδιο GameObject. Ένα Component είναι ένα στιγμιότυπο μιας κλάσης, έτσι το πρώτο βήμα είναι να κάνουμε μια αναφορά στο Component αντικείμενο με το οποίο θέλουμε να εργαστούμε και να αποκτήσουμε πρόσβαση σε αυτό. Αυτό γίνεται με τη συνάρτηση GetComponent. Συνήθως, θέλουμε να εκχωρήσουμε το Component αντικείμενο σε μια μεταβλητή, η οποία γίνεται σε C# χρησιμοποιώντας την ακόλουθη σύνταξη:

```
void Start () {
        Rigidbody rb = GetComponent<Rigidbody>();
}
```

Στην UnityScript, η σύνταξη είναι ελαφρά διαφορετική:

```
function Start () {
        var rb = GetComponent.<Rigidbody>();
}
```

Μόλις έχουμε μια αναφορά σε ένα Component, μπορούμε να ορίσουμε τις τιμές των ιδιοτήτων του, όπως θα κάναμε στον Inspector:

```
void Start () {
        Rigidbody rb = GetComponent<Rigidbody>();
        // Change the mass of the object's Rigidbody.
        rb.mass = 10f;
}
```

Ένα επιπλέον χαρακτηριστικό που δεν είναι διαθέσιμο στον Inspector είναι η δυνατότητα της κλήσης των συναρτήσεων των Component στιγμιοτύπων:

```
void Start () {
        Rigidbody rb = GetComponent<Rigidbody>();
        // Add a force to the Rigidbody.
```

```
        rb.AddForce(Vector3.up * 10f);
}
```

Επίσης μπορούμε να έχουμε περισσότερα από ένα script συνδεδεμένα στο ίδιο αντικείμενο. Η πρόσβαση από το ένα script στο άλλο γίνεται με την GetComponent και το όνομα του script στο οποίο θέλουμε να έχουμε πρόσβαση και το όνομα της script class για να καθορίσουμε το Component που θέλουμε.

Εάν προσπαθήσουμε να ανακτήσουμε ένα Component που στην πραγματικότητα δεν έχει προστεθεί στο GameObject τότε η GetComponent θα επιστρέψει null, θα πάρουμε μια null αναφορά λάθους κατά το χρόνο εκτέλεσης, εάν προσπαθήσουμε να αλλάξουμε οποιεσδήποτε τιμές σε μηδενικό αντικείμενο.

Σε ορισμένα είδη Components που χρησιμοποιούνται πολύ συχνά, η Unity παρέχει ενσωματωμένες μεταβλητές στις οποίες μπορούμε να έχουμε πρόσβαση στην κλάση της MonoBehaviour, ώστε να μπορούμε να χρησιμοποιήσουμε πράγματα όπως:

```
void Start () {
        transform.position = Vector3.zero;
}
```

χωρίς να χρειάζεται να χρησιμοποιήσουμε GetComponent για να αποκτήσουμε πρόσβαση στο Component Transform. Εάν το επιθυμητό συστατικό δεν συνδέεται με το αντικείμενο, η μεταβλητή θα περιέχει μια μηδενική τιμή.

Accessing Other Objects

Αν και μερικές φορές λειτουργούν σε απομόνωση, είναι κοινό για τα scripts να παρακολουθούν άλλα αντικείμενα. Για παράδειγμα, σε μια καταδίωξη του εχθρού ίσως χρειαστεί να γνωρίζουμε τη θέση του παίκτη. Η Unity παρέχει μια σειρά από διαφορετικούς τρόπους για να ανακτήσουμε άλλα αντικείμενα, κάθε ένας κατάλληλος για ορισμένες καταστάσεις.

Ο πιο απλός τρόπος για να βρούμε ένα σχετικό GameObject είναι να προσθέσουμε μια public GameObject μεταβλητή στο script:

```
public class Enemy : MonoBehaviour {
        public GameObject player;

        // Other variables and functions...
}
```

Η μεταβλητή αυτή θα είναι ορατή στον Inspector όπως και κάθε άλλη. Μπορούμε τώρα να σύρουμε ένα αντικείμενο από τη σκηνή ή τον πίνακα Hierarchy στη μεταβλητή αυτή για να το αναθέσουμε. Η συνάρτηση GetComponent και οι προσβάσιμες

μεταβλητές Component είναι διαθέσιμες για αυτό το αντικείμενο, όπως και για κάθε άλλο, ώστε να μπορούμε να χρησιμοποιήσουμε κώδικα σαν τον παρακάτω:

```
public class Enemy : MonoBehaviour {
	public GameObject player;

	void Start() {
		// Start the enemy ten units behind the player character.
		transform.position = player.transform.position - Vector3.forward * 10f;
	}
}
```

Επιπλέον, αν δηλώσουμε μια δημόσια μεταβλητή ενός τύπου Component στο script μας, μπορούμε να σύρουμε οποιοδήποτε GameObject που έχει το Component που είναι συνδεδεμένο σε αυτό. Αυτό θα έχει άμεση πρόσβαση στο Component και όχι το ίδιο το GameObject.

```
public Transform playerTransform;
```

Συνδέοντας αντικείμενα μαζί με τις μεταβλητές είναι πιο χρήσιμο όταν έχουμε να κάνουμε με μεμονωμένα αντικείμενα που έχουν μόνιμες συνδέσεις. Μπορούμε να χρησιμοποιήσουμε μια μεταβλητή πίνακα για να συνδέσουμε διάφορα αντικείμενα του ίδιου τύπου, αλλά οι συνδέσεις θα πρέπει ακόμη να γίνουν στον επεξεργαστή Unity και όχι κατά το χρόνο εκτέλεσης. Συχνά είναι βολικό να εντοπίσουμε τα αντικείμενα κατά το χρόνο εκτέλεσης και η Unity παρέχει δύο βασικούς τρόπους για να γίνει αυτό, όπως περιγράφεται παρακάτω.

Μερικές φορές, μια σκηνή παιχνιδιού θα κάνει χρήση ενός αριθμού αντικειμένων του ίδιου τύπου, όπως οι εχθροί, τα σημεία αναφοράς και τα εμπόδια. Αυτά μπορεί να χρειαστεί να παρακολουθούνται από ένα συγκεκριμένο script που εποπτεύει ή αντιδρά σε αυτά (π.χ., όλα τα σημεία μπορεί να χρειαστεί να είναι διαθέσιμα σε ένα script pathfinding). Η χρήση μεταβλητών για τη σύνδεση αυτών των αντικειμένων είναι μια πιθανότητα, αλλά θα καταστήσει τη διαδικασία σχεδιασμού κουραστική αν κάθε νέο σημείο πρέπει να συρθεί σε μια μεταβλητή σε ένα script. Ομοίως, αν ένα σημείο αναφοράς διαγραφεί, τότε θα πρέπει να αφαιρέσουμε την αναφορά μεταβλητής στο αντικείμενο που λείπει. Σε περιπτώσεις όπως αυτή, είναι συχνά καλύτερο να διαχειριστούμε ένα σύνολο αντικειμένων, κάνοντάς τα όλα παιδιά ενός γονέα αντικειμένου. Τα αντικείμενα παιδιά μπορούν να ανακτηθούν χρησιμοποιώντας το Transform Component του γονέα (αφού όλα τα GameObjects έχουν σιωπηρά Transform):

```
public class WaypointManager : MonoBehaviour {
	public Transform waypoints;

	void Start() {
```

```
                waypoints = new Transform[transform.childCount];
                int i = 0;

                for (Transform t in transform) {
                        waypoints[i++] = t;
                }
        }
}
```

Μπορούμε επίσης να εντοπίσουμε ένα συγκεκριμένο παιδί αντικειμένου με το όνομά του, χρησιμοποιώντας τη συνάρτηση Transform.Find:

```
transform.Find ("Gun")?
```

Αυτό μπορεί να είναι χρήσιμο όταν ένα αντικείμενο έχει ένα παιδί που μπορεί να προστεθεί ή να αφαιρεθεί κατά τη διάρκεια του παιχνιδιού. Ένα όπλο που μπορεί να σηκώνεται και αφήνεται κάτω, είναι ένα καλό παράδειγμα.

Είναι πάντα δυνατό να εντοπιστούν GameObjects οπουδήποτε στην ιεραρχία της σκηνής για όσο διάστημα έχουμε κάποιες πληροφορίες για τον εντοπισμό τους. Μεμονωμένα αντικείμενα μπορούν να ανακτηθούν από το όνομα, χρησιμοποιώντας τη συνάρτηση GameObject.Find:

```
GameObject player;

void Start() {
        player = GameObject.Find("MainHeroCharacter");
}
```

Ένα αντικείμενο ή μια συλλογή αντικειμένων μπορούν επίσης να βρίσκονται με βάση την ετικέτα τους, χρησιμοποιώντας τις συναρτήσεις GameObject.FindWithTag και GameObject.FindGameObjectsWithTag:

```
GameObject player;
GameObject[] enemies;

void Start() {
        player = GameObject.FindWithTag("Player");
        enemies = GameObject.FindGameObjectsWithTag("Enemy");
}
```

Event Functions

Ένα script στη Unity δεν είναι σαν την παραδοσιακή ιδέα ενός προγράμματος, όπου ο κώδικας τρέχει συνεχώς στο βρόχο μέχρι να ολοκληρώσει το έργο του. Αντ 'αυτού, η Unity

περνά τον έλεγχο σε ένα script κατά διαστήματα με την κλήση ορισμένων λειτουργιών που δηλώνονται μέσα σε αυτό. Μόλις μια λειτουργία έχει τελειώσει την εκτέλεση, ο έλεγχος περνά πίσω στη Unity. Οι λειτουργίες αυτές είναι γνωστές ως συναρτήσεις γεγονότων, δεδομένου ότι ενεργοποιούνται από την Unity ως απάντηση σε γεγονότα που συμβαίνουν κατά τη διάρκεια του παιχνιδιού. Η Unity χρησιμοποιεί ένα σύστημα ονοματοδοσίας για να προσδιορίσει ποιες συναρτήσεις να καλέσουμε για ένα συγκεκριμένο γεγονός, όπως είδαμε με τη συνάρτηση Update (που καλλείται πριν την ενημερωμένη του πλαισίου) και τη συνάρτηση Start (που καλλείται ακριβώς πριν από την πρώτη ενημέρωση του πλαισίου του αντικειμένου). Πολλά περισσότερα γεγονότων συναρτήσεων είναι διαθέσιμα στη Unity. Τα ακόλουθα είναι μερικά από τα πιο κοινά και σημαντικά γεγονότα.

Ένα παιχνίδι είναι σαν μια κινούμενη εικόνα, όπου τα animation frames δημιουργούνται σε πραγματικό χρόνο κρυπτογράφησης (on the fly). Μια βασική έννοια του προγραμματισμού παιχνιδιών είναι η πραγματοποίηση αλλαγών στη θέση, την κατάσταση και τη συμπεριφορά των αντικειμένων στο παιχνίδι λίγο πριν αποδοθεί κάθε πλαίσιο. Η συνάρτηση Update είναι ο κύριος τόπος για αυτό το είδος του κώδικα στη Unity. Η ενημέρωση καλείται πριν το πλαίσιο κατασταθεί και επίσης πριν υπολογιστούν τα κινούμενα σχέδια.

```
void Update() {
        float distance = speed * Time.deltaTime * Input.GetAxis("Horizontal");
        transform.Translate(Vector3.right * speed);
}
```

Η μηχανή φυσικής ενημερώνεται επίσης σε διακριτά χρονικά βήματα με παρόμοιο τρόπο με την παροχή πλαισίου. Μια ξεχωριστή συνάρτηση γεγονότων που ονομάζεται FixedUpdate καλείται αμέσως πριν από κάθε ενημέρωση της φυσικής. Αφού οι ενημερώσεις φυσικής και οι ενημερώσεις πλαισίου δεν εμφανίζονται με την ίδια συχνότητα, θα παίρνουμε πιο ακριβή αποτελέσματα από τον κώδικα της φυσικής, αν μπορούμε να τον τοποθετήσουμε στη συνάρτηση FixedUpdate παρά στην Update.

```
void FixedUpdate() {
        Vector3 force = transform.forward * driveForce * Input.GetAxis("Vertical");
        rigidbody.AddForce(force);
}
```

Επίσης, είναι χρήσιμο μερικές φορές να είμαστε σε θέση να κάνουμε πρόσθετες αλλαγές σε ένα σημείο μετά που θα έχουν κληθεί οι συναρτήσεις Update και FixedUpdate για όλα τα αντικείμενα στη σκηνή και μετά που θα έχουν υπολογιστεί όλα τα κινούμενα σχέδια. Ένα παράδειγμα είναι όταν μια φωτογραφική μηχανή πρέπει να παραμείνει σε ένα αντικείμενο-στόχο, η προσαρμογή με τον προσανατολισμό της φωτογραφικής μηχανής πρέπει να γίνει μετά που το αντικείμενο προορισμού θα έχει μετακινηθεί. Ένα άλλο παράδειγμα είναι όταν ο script κώδικας πρέπει να υπερισχύει το αποτέλεσμα μιας κίνησης (για παράδειγμα, να

κάνει το κεφάλι του χαρακτήρα θα στραφεί προς ένα αντικείμενο-στόχο στη σκηνή). Η LateUpdate συνάρτηση μπορεί να χρησιμοποιηθεί για τέτοιου είδους καταστάσεις.

```
void LateUpdate() {
        Camera.main.transform.LookAt(target.transform);
}
```

Είναι συχνά χρήσιμο να είμαστε σε θέση να καλέσουμε τον κώδικα αρχικοποίησης πριν από τις ενημερώσεις που συμβαίνουν κατά τη διάρκεια του παιχνιδιού. Η συνάρτηση αυτή καλείται πριν από το πρώτο καρέ ή την ενημέρωση της φυσικής σε ένα αντικείμενο. Η συνάρτηση Awake καλείται για κάθε αντικείμενο στη σκηνή κατά τη στιγμή που φορτώνει η σκηνή. Αν τα διάφορα αντικείμενα καλούνται στις Start και Awake συναρτήσεις με αυθαίρετη σειρά, όλα τα Awakes θα έχει ολοκληρωθούν πριν κληθεί το πρώτο Start. Αυτό σημαίνει ότι ο κώδικας στη συνάρτηση Start μπορεί να κάνει χρήση των άλλων αρχικοποιήσεων που προηγουμένως πραγματοποιήθηκαν στην Awake φάση.

Η Unity έχει ένα σύστημα για την απόδοση των ελέγχων GUI πάνω από την κύρια δράση στη σκηνή και ανταποκρίνεται σε κλικ σε αυτούς τους ελέγχους. Αυτόν τον κώδικα τον χειρίζεται κάπως διαφορετικά από την κανονική ενημέρωση πλαισίου και έτσι θα πρέπει να τοποθετηθεί στη συνάρτηση OnGUI, η οποία καλείται περιοδικά.

```
void OnGUI() {
        GUI.Label(labelRect, "Game Over");
}
```

Μπορούμε επίσης να ανιχνεύσουμε τα γεγονότα του ποντικιού που συμβαίνουν σε ένα GameObject κατά τη διάρκεια που εμφανίζεται στη σκηνή. Αυτό μπορεί να χρησιμοποιηθεί για στοχοθέτηση όπλων ή την εμφάνιση πληροφοριών για το χαρακτήρα υπό το δείκτη του ποντικιού. Ένα σύνολο συναρτήσεων OnMouseXXX γεγονότων (π.χ., onMouseOver, OnMouseDown) είναι διαθέσιμο για να επιτρέψει ένα script να αντιδράσει στις ενέργειες του χρήστη με το ποντίκι. Για παράδειγμα, αν το κουμπί του ποντικιού είναι πατημένο, ενώ ο δείκτης είναι πάνω από ένα συγκεκριμένο αντικείμενο τότε μια OnMouseDown συνάρτηση στο εν λόγω σενάριο αντικειμένου θα συγκληθεί, αν υπάρχει.

Η μηχανή φυσικής θα αναφέρει συγκρούσεις εναντίον ενός αντικειμένου καλώντας συναρτήσεις γεγονότων για τον κώδικα αυτού του αντικειμένου. Οι συναρτήσεις OnCollisionEnter, OnCollisionStay και OnCollisionExit θα κληθούν όταν πραγματοποιηθεί επαφή, παραμείνει μια επαφή και σταματήσει να υπάρχει επαφή. Οι αντίστοιχες OnTriggerEnter, OnTriggerStay και OnTriggerExit συναρτήσεις θα καλούνται όταν ο επιταχυντής (collider) του αντικειμένου έχει διαμορφωθεί ως Trigger (δηλαδή, ένας επιταχυντής που απλά ανιχνεύει όταν κάτι εισέρχεται, παρά αντιδρά φυσικά). Αυτές οι συναρτήσεις μπορούν να κληθούν αρκετές φορές διαδοχικά, εάν ανιχνεύονται περισσότερες από μία επαφές κατά την ενημέρωση της φυσικής και έτσι μια παράμετρος περνάει στη

συνάρτηση δίνοτας λεπτομέρειες της σύγκρουσης (τη θέση, την ταυτότητα του εισερχόμενου αντικειμένου, κλπ).

```
void OnCollisionEnter(otherObj: Collision) {
        if (otherObj.tag == "Arrow") {
                ApplyDamage(10);
        }
}
```

Δημιουργία και Καταστροφή GameObjects

Μερικά παιχνίδια κρατάνε ένα σταθερό αριθμό αντικειμένων στη σκηνή, αλλά είναι πολύ συνηθισμένο για τους χαρακτήρες, θησαυρούς και άλλα αντικείμενα να δημιουργούνται και να αφαιρούνται κατά τη διάρκεια του παιχνιδιού. Στην Unity, ένα GameObject μπορεί να δημιουργηθεί χρησιμοποιώντας τη συνάρτηση Instantiate που κάνει ένα νέο αντίγραφο ενός υπάρχοντος αντικειμένου:

```
public GameObject enemy;

void Start() {
        for (int i = 0; i < 5; i++) {
                Instantiate(enemy);
        }
}
```

Το αντικείμενο από το οποίο κατασκευάζεται το αντίγραφο δεν χρειάζεται να είναι παρόν στη σκηνή. Είναι πιο κοινό να χρησιμοποιήσουμε ένα prefab και να το σύρουμε σε μια public μεταβλητή από το Project panel στον editor. Επίσης, ένα στιγμιότυπο GameObject θα αντιγράψει όλα τα Components που υπάρχουν στο πρωτότυπο.

Υπάρχει επίσης μια συνάρτηση Destroy που θα καταστρέψει ένα αντικείμενο μετά που θα τελειώσει η ενημέρωση του πλαισίου ή προαιρετικά μετά από μια μικρή χρονική καθυστέρηση:

```
void OnCollisionEnter(otherObj: Collision) {
        if (otherObj == "Missile") {
                Destroy(gameObject,.5f);
        }
}
```

Η συνάρτηση Destroy μπορεί να καταστρέψει τα επιμέρους components, χωρίς να επηρεάζει το ίδιο το GameObject. Αν γράψουμε όμως:

```
Destroy(this);
```

καταστρέφει το script component που το καλεί αντί να καταστρέψει το GameObject που είναι συνδεδεμένο το script.

Coroutines

Όταν καλούμε μια συνάρτηση, τρέχει να ολοκληρωθεί πριν από την επιστροφή. Αυτό ουσιαστικά σημαίνει ότι κάθε δράση λαμβάνει χώρα σε μια συνάρτηση πρέπει να γίνει μέσα σε ένα ενιαίο πλαίσιο ενημέρωσης, μια κλήση συνάρτησης δεν μπορεί να χρησιμοποιηθεί για να περιέχει ένα διαδικαστικό animation ή μια ακολουθία γεγονότων με την πάροδο του χρόνου. Ως παράδειγμα, αναφέρεται η σταδιακή μείωση της alpha τιμής ενός αντικειμένου (αδιαφάνειας) μέχρι να γίνει εντελώς αόρατο.

```
void Fade() {
        for (float f = 1f; f >= 0; f -= 0.1f) {
                Color c = renderer.material.color;
                c.a = f;
                renderer.material.color = c;
        }
}
```

Η συνάρτηση Fade δεν θα έχει το αποτέλεσμα που θα περίμενε κανείς. Για να είναι ορατό το ξεθώριασμα, το alpha πρέπει να μειωθεί σε μια ακολουθία καρέ για να δείξει τις ενδιάμεσες τιμές που παρέχονται. Ωστόσο, η συνάρτηση θα εκτελεστεί στο σύνολό της εντός ενός ενιαίου πλαισίου ενημέρωσης. Οι ενδιάμεσες τιμές δεν θα φανούν και το αντικείμενο θα εξαφανιστεί αμέσως.

Είναι δυνατόν να χειριστούμε καταστάσεις όπως αυτή με την προσθήκη κώδικα στη συνάρτηση Update που εκτελεί το ξεθώριασμα με καρέ-καρέ βάση. Ωστόσο, είναι συχνά πιο βολικό να χρησιμοποιήσουμε μια coroutine για αυτό το είδος της εργασίας.

Μια coroutine είναι σαν μια συνάρτηση που έχει την δυνατότητα να κάνει παύση εκτέλεσης και επιστροφή του ελέγχου στην Unity αλλά στη συνέχεια να συνεχίσει από εκεί που σταμάτησε στο παρακάτω πλαίσιο. Σε C#, ένα coroutine δηλώνεται ως εξής:

```
IEnumerator Fade() {
        for (float f = 1f; f >= 0; f -= 0.1f) {
                Color c = renderer.material.color;
                c.a = f;
                renderer.material.color = c;
                yield return;
        }
}
```

Πρόκειται ουσιαστικά για μια συνάρτηση που δηλώνεται με ένα είδος επιστροφής του IEnumerator και με την δήλωση «yield return» να περιλαμβάνεται κάπου στο σώμα. Η γραμμή «yield return» είναι το σημείο στο οποίο η εκτέλεση θα διακοπεί προσωρινά και θα ξαναρχίσει το ακόλουθο πλαίσιο. Για να ορίσουμε μια λειτουργία coroutine, θα πρέπει να χρησιμοποιήσουμε τη συνάρτηση StartCoroutine:

```
void Update() {
	if (Input.GetKeyDown("f")) {
		StartCoroutine("Fade");
	}
}
```

Στην UnityScript, τα πράγματα είναι λίγο πιο απλά. Κάθε συνάρτηση που περιλαμβάνει τη δήλωση «yield» είναι κατανοητό ότι είναι μια coroutine και ο τύπος IEnumerator return δεν χρειάζεται να δηλώνεται ρητά:

```
function Fade() {
	for (var f = 1.0; f >= 0; f -= 0.1) {
		var c = renderer.material.color;
		c.a = f;
		renderer.material.color = c;
		yield;
	}
}
```

Επίσης, μια coroutine μπορεί να ξεκινήσει στη UnityScript καλώντας το σαν να ήταν μια φυσιολογική συνάρτηση:

```
function Update() {
	if (Input.GetKeyDown("f")) {
		Fade();
	}
}
```

Ο μετρητής βρόχου στη συνάρτηση Fade διατηρεί τη σωστή τιμή για τη διάρκεια ζωής της coroutine. Στην πραγματικότητα, οποιαδήποτε μεταβλητή ή παράμετρος θα πρέπει να διατηρηθεί σωστά μεταξύ των yields.

Από προεπιλογή, μια coroutine επανέρχεται στο πλαίσιο μετά το yield, αλλά είναι επίσης δυνατό να εισαγάγει μια χρονική καθυστέρηση χρησιμοποιώντας την WaitForSeconds:

```
IEnumerator Fade() {
	for (float f = 1f; f >= 0; f -= 0.1f) {
		Color c = renderer.material.color;
		c.a = f;
		renderer.material.color = c;
		yield return new WaitForSeconds(.1f);
	}
}
```

Στη UnityScript:

```
function Fade() {
	for (var f = 1.0; f >= 0; f -= 0.1) {
```

```
                var c = renderer.material.color;
                c.a = f;
                renderer.material.color = c;
                yield WaitForSeconds(0.1);
        }
}
```

Αυτό μπορεί να χρησιμοποιηθεί ως ένας τρόπος για να εξαπλωθεί μια επίδραση σε μια χρονική περίοδο, αλλά είναι επίσης μια χρήσιμη βελτιστοποίηση. Πολλά καθήκοντα σε ένα παιχνίδι πρέπει να πραγματοποιούνται σε τακτά χρονικά διαστήματα και ο πιο προφανής τρόπος για να γίνει αυτό είναι να τα συμπεριλάβουμε στη συνάρτηση Update. Ωστόσο, η συνάρτηση αυτή τυπικά θα καλείται πολλές φορές ανά δευτερόλεπτο. Όταν ένα έργο δεν χρειάζεται να επαναλαμβάνεται αρκετά τόσο συχνά, μπορούμε να το βάλουμε σε μία coroutine για να πάρουμε μια ενημέρωση σε τακτική βάση, αλλά όχι για κάθε καρέ. Ένα παράδειγμα αυτού θα μπορούσε να είναι ένας συναγερμός που προειδοποιεί τον παίκτη αν ο εχθρός είναι κοντά. Ο κώδικας μπορεί να μοιάζει κάπως έτσι:

```
function ProximityCheck() {
        for (int i = 0; i < enemies.Length; i++) {
                if (Vector3.Distance(transform.position, enemies[i].transform.position) <
                                                                        dangerDistance) {
                        return true;
                }
        }
        return false;
}
```

Εάν υπάρχουν πολλοί εχθροί, καλώντας αυτή τη συνάρτηση κάθε καρέ θα μπορούσε να προκαλέσει ένα σημαντικό overhead. Ωστόσο, μπορούμε να χρησιμοποιήσουμε μια coroutine που να την καλούμε κάθε δέκατο του δευτερολέπτου:

```
IEnumerator DoCheck() {
        for(;;) {
                ProximityCheck;
                yield return new WaitForSeconds(.1f);
        }
}
```

Αυτό θα μειώσει σημαντικά τον αριθμό των ελέγχων που διενεργούνται χωρίς αξιοσημείωτη επίδραση στο gameplay.

Special Folders και Script Compilation Order

Για το μεγαλύτερο μέρος, μπορούμε να επιλέξουμε οποιοδήποτε όνομα θέλουμε για τους φακέλους στο έργο μας, αλλά η Unity διατηρεί κάποια ονόματα για να δείξει ότι το περιεχόμενο έχει έναν ειδικό σκοπό. Μερικοί από αυτούς τους φακέλους έχουν συνέπειες για

τη σειρά σύνταξης σεναρίου. Ουσιαστικά, υπάρχουν τέσσερις χωριστές φάσεις σύνταξης σεναρίου και η φάση όπου θα πρέπει να καταρτιστεί ένα σενάριο καθορίζεται από τον φάκελο γονέα.

Αυτό είναι σημαντικό σε περιπτώσεις όπου ένα σενάριο πρέπει να αναφέρεται σε κλάσεις που ορίζονται σε άλλα scripts. Ο βασικός κανόνας είναι ότι οτιδήποτε που θα πρέπει να συνταχθεί σε μία φάση μετά την τωρινή δεν μπορεί να αναφέρεται. Οτιδήποτε που συντάσσεται στην παρούσα φάση ή σε προηγούμενη φάση είναι πλήρως διαθέσιμο.

Μια άλλη κατάσταση εμφανίζεται όταν ένα script γραμμένο σε μία γλώσσα πρέπει να αναφέρεται σε μια κλάση ορισμένη σε άλλη γλώσσα (π.χ., ένα αρχείο UnityScript που δηλώνει τις μεταβλητές μιας κλάσης ορισμένη σε C# script). Ο κανόνας εδώ είναι ότι η κλάση που αναφέρεται πρέπει να έχει συνταχθεί σε μια προηγούμενη φάση.

Οι φάσεις της σύνταξης έχουν ως εξής:

- Φάση 1: Runtime scripts σε φακέλους που ονομάζονται Standard Assets, Pro Standard Assets και Plugins.
- Φάση 2: Editor scripts σε φακέλους που ονομάζονται Standard Assets/Editor, Pro Standard Assets/Editor και Plugins/Editor.
- Φάση 3: Όλα τα άλλα σενάρια που δεν είναι μέσα σε ένα φάκελο που ονομάζεται Editor.
- Φάση 4: Όλα τα υπόλοιπα σενάρια (δηλαδή, αυτά που είναι μέσα σε ένα φάκελο που ονομάζεται Editor).

Επιπλέον, κάθε σενάριο μέσα σε ένα φάκελο που ονομάζεται WebPlayerTemplates στο κορυφαίο επίπεδο του φακέλου Assets δεν θα συνταχθεί καθόλου. Αυτή η συμπεριφορά είναι ελαφρώς διαφορετική από τα άλλα ειδικά ονόματα φακέλων που λειτουργούν επίσης σε υπο-φακέλους (π.χ., Scripts/Editor λειτουργεί ως ένας φάκελος script editor, αλλά Scripts/WebPlayerTemplates δεν εμποδίζει την σύνταξη).

Ένα κοινό παράδειγμα είναι όταν ένα αρχείο UnityScript πρέπει να αναφερθεί σε μια κλάση ορισμένη σε ένα αρχείο C#. Μπορούμε να το επιτύχουμε αυτό με την τοποθέτηση του C# αρχείου μέσα σε ένα φάκελο Plugins και το αρχείο UnityScript σε έναν μη-ειδικό φάκελο. Αν δεν το κάνουμε αυτό, θα πάρουμε ένα λάθος το οποίο θα λέει ότι η C# κλάση δεν μπορεί να βρεθεί.

Namespaces

Καθώς τα έργα γίνονται μεγαλύτερα και ο αριθμός των scripts αυξάνεται, η πιθανότητα να έχουμε συγκρούσεις μεταξύ των ονομάτων κλάσης σεναρίων μεγαλώνει όλο και περισσότερο. Αυτό ισχύει ιδιαίτερα όταν αρκετοί προγραμματιστές εργάζονται σε

διαφορετικές πτυχές του παιχνιδιού ξεχωριστά και θα συνδυάσουν στο τέλος τις προσπάθειές τους σε ένα έργο. Για παράδειγμα, ένας προγραμματιστής μπορεί να γράφει τον κώδικα που να ελέγχει τον κύριο χαρακτήρα, ενώ ένας άλλος παίκτης γράφει τον αντίστοιχο κώδικα για τον εχθρό. Και οι δύο προγραμματιστές μπορούν να επιλέξουν να ονομάσουν την κύρια κλάση του script Controller, αλλά αυτό θα προκαλέσει μια σύγκρουση, όταν τα έργα τους συνδυαστούν.

Σε κάποιο βαθμό, το πρόβλημα αυτό μπορεί να αποφευχθεί με τη θέσπιση μιας σύμβασης ονομασίας ή μετονομάζοντας τις κλάσεις κάθε φορά που ανακαλύπτεται μια σύγκρουση (π.χ., στις παραπάνω κλάσεις θα μπορούσαν να δοθούν ονόματα όπως PlayerController και EnemyController). Ωστόσο, αυτό είναι ενοχλητικό όταν υπάρχουν περισσότερες κλάσεις με συγκρουόμενα ονόματα ή όταν δηλώνονται μεταβλητές με τη χρήση αυτών των ονομάτων, γιατί κάθε αναφορά του παλιού ονόματος κλάσης θα πρέπει να αντικατασταθεί για την σύνταξη του κώδικα.

Η γλώσσα C# προσφέρει ένα χαρακτηριστικό που ονομάζεται namespaces που λύνει αυτό το πρόβλημα με ένα ισχυρό τρόπο. Το namespace είναι απλά μια συλλογή από κλάσεις που αναφέρονται χρησιμοποιώντας ένα επιλεγμένο πρόθεμα στο όνομα της κλάσης. Στο παρακάτω παράδειγμα, οι κατηγορίες Controller1 και Controller2 είναι μέλη ενός namespace που ονομάζεται Enemy:

```
namespace Enemy {
        public class Controller1 : MonoBehaviour {
                ...
        }
        public class Controller2 : MonoBehaviour {
                ...
        }
}
```

Στον κώδικα, αυτές οι κλάσεις αναφέρονται ως Enemy.Controller1 και Enemy.Controller2, αντίστοιχα. Αυτό είναι καλύτερο από τη μετονομασία των κλάσεων, εφόσον η δήλωση namespace μπορεί να παρεμβάλλεται γύρω από τις υφιστάμενες δηλώσεις κλάσεων (δηλαδή, δεν είναι απαραίτητο να αλλάξουμε τα ονόματα όλων των κλάσεων ξεχωριστά). Επιπλέον, μπορούμε να χρησιμοποιήσουμε πολλές αγκύλες τμημάτων namespace γύρω από τις κλάσεις όπου και αν συμβαίνουν, ακόμη και αν οι κλάσεις αυτές βρίσκονται σε διαφορετικά αρχεία προέλευσης.

Μπορούμε να αποφύγουμε να πληκτρολογήσουμε το πρόθεμα namespace επανειλημμένα με την προσθήκη μιας «using» οδηγία στο επάνω μέρος του αρχείου.

```
using Enemy;
```

Η γραμμή αυτή δείχνει ότι, όταν τα ονόματα κατηγορίας Controller1 και Controller2 βρεθούν, θα πρέπει να νοείται Enemy.Controller1 και Enemy.Controller2, αντίστοιχα. Αν το script πρέπει επίσης να αναφέρεται σε κλάσεις με το ίδιο όνομα από ένα διαφορετικό namespace (που ονομάζεται Player, ας πούμε), τότε μπορεί ακόμα να χρησιμοποιηθεί το πρόθεμα. Εάν δύο namespaces που περιέχουν συγκρουόμενα ονόματα των κλάσεων εισάγονται με using οδηγίες την ίδια στιγμή, ο compiler θα αναφέρει ένα σφάλμα.

Attributes

Τα χαρακτηριστικά- Attributes είναι δείκτες που μπορούν να τοποθετηθούν πάνω από μια κλάση, ιδιοκτησία ή συνάρτηση σε ένα script για να δείξει ιδιαίτερη συμπεριφορά. Για παράδειγμα, το χαρακτηριστικό HideInInspector μπορεί να προστεθεί πάνω από τη δήλωση ιδιοκτησίας για να εμποδίζει το στοιχείο από το να εμφανιστεί στον Inspector, ακόμα κι αν είναι δημόσιο. Στη JavaScript, το όνομα ενός γνωρίσματος ξεκινά με το σύμβολο «@», ενώ σε C# και Boo, περιέχεται μέσα σε αγκύλες:

```
// JS
@HideInInspector
var strength: float;

// C#
[HideInInspector]
public float strength;
```

Η Unity παρέχει μια σειρά από χαρακτηριστικά που αναφέρονται στην αναφορά script (επιλέγουμε Editor ή το τμήμα Runtime Attributes από το αναδυόμενο μενού στο sidebar). Υπάρχουν επίσης τα χαρακτηριστικά που ορίζονται στις βιβλιοθήκες .NET που μπορεί μερικές φορές να είναι χρήσιμα στον κώδικα Unity.

Unity Scripting Reference

Η αναφορά scripting είναι οργανωμένη σύμφωνα με τις κλάσεις που διατίθενται στα scripts που περιγράφονται μαζί με τις μεθόδους τους, τις ιδιότητές τους , καθώς και κάθε άλλη πληροφορία σχετική με τη χρήση τους .

Η Unity διαθέτει στις οδηγίες χρήσεις της, σελίδες με παραδείγματα κώδικα που είμαστε ελεύθεροι να χρησιμοποιήσουμε για οποιοδήποτε σκοπό. Τα παραδείγματα μπορούν να προβληθούν σε οποιαδήποτε από τις τρεις γλώσσες που υποστηρίζονται (C#, JavaScript και Boo) χρησιμοποιώντας το μενού στο πάνω μέρος της κάθε σελίδας. Η API είναι η ίδια, ανεξάρτητα από το ποια γλώσσα χρησιμοποιείται, έτσι ώστε η επιλογή της γλώσσας είναι καθαρά θέμα προτιμήσεων.

Υπάρχουν διαθέσιμες υποενότητες της αναφοράς, το τμήμα Runtime Classes που αποτελεί το κύριο τμήμα, και άλλα τμήματα του API, συμπεριλαμβανομένης την επέκταση API του Editor.

3.3.6 Mobile Input

Στα iOS και Android, η κλάση εισόδου (Input class) προσφέρει πρόσβαση στην οθόνη αφής (touchscreen), το επιταχυνσιόμετρο (accelerometer) και τη γεωγραφική θέση (geographical/location). Η πρόσβαση στο πληκτρολόγιο των κινητών συσκευών παρέχεται μέσω του πληκτρολογίου iOS.

Multi-Touch Screen

Οι συσκευές iPhone και iPod Touch είναι ικανές να ανιχνεύσουν έως και πέντε δάχτυλα που αγγίζουν την οθόνη ταυτόχρονα. Μπορούμε να ανακτήσουμε την κατάσταση του κάθε δαχτύλου που αγγίζει την οθόνη κατά τη διάρκεια του τελευταίου καρέ από την πρόσβαση στην Input.touches.

Οι Android συσκευές δεν έχουν ένα ενιαίο όριο για το πόσα δάχτυλα μπορούν να εντοπίσουν αλλά αυτό ποικίλλει από συσκευή σε συσκευή και μπορεί να ποικίλει, από δύο-touch για παλαιότερες συσκευές σε πέντε δάχτυλα σε κάποιες νεότερες συσκευές.

Κάθε άγγιγμα δαχτύλου αντιπροσωπεύεται από ένα **Input.Touch** στη δομή δεδομένων:

- **fingerId** : Το μοναδικό διάνυσμα για ένα άγγιγμα.
- **position** : Η θέση στην οθόνη αφής.
- **deltaPosition** : Η αλλαγή της θέσης της οθόνης από το τελευταίο καρέ.
- **deltaTime** : Το χρονικό διάστημα που έχει περάσει από την τελευταία αλλαγή κατάστασης.
- **tapCount** : Η οθόνη του iPhone/iPad είναι σε θέση να διακρίνει τα γρήγορα αγγίγματα του δαχτύλου από το χρήστη. Αυτός ο μετρητής δείχνει πόσες φορές ο χρήστης έχει ακουμπήσει την οθόνη χωρίς να κουνήσει το δάχτυλό προς τα πλάγια. Οι Android συσκευές δεν μετρούν τον αριθμό των γρήγορων αγγιγμάτων της οθόνης, για αυτό το tapCount είναι πάντα 1.
- **Phase** : Περιγράφει την φάση της αφής, ώστε να μπορούμε να προσδιορίσουμε αν το άγγιγμα μόλις ξεκίνησε, αν ο χρήστης μετακινεί το δάχτυλό του ή αν απλά σήκωσε το δάχτυλο.
    - **Began** : ένα δάχτυλο μόλις άγγιξε την οθόνη.
    - **Moved** : ένα δάχτυλο κινείται στην οθόνη.
    - **Stationary** : ένα δάκτυλο αγγίζει την οθόνη, αλλά δεν έχει μετακινηθεί από το τελευταίο καρέ.

- **Ended** : ένα δάκτυλο άφησε την οθόνη. Αυτή είναι η τελική φάση ενός αγγίγματος.
- **Canceled** : Το σύστημα ακύρωσε την παρακολούθηση της αφής, όπως όταν ο χρήστης τοποθετεί τη συσκευή στο πρόσωπό του ή συμβαίνουν ταυτόχρονα περισσότερα από πέντε αγγίγματα. Αυτή είναι η τελική φάση ενός αγγίγματος.

Επιταχυνσιόμετρο (Accelerometer)

Καθώς κινείται η κινητή συσκευή, ένα ενσωματωμένο επιταχυνσιόμετρο εκθέτει τις γραμμικές μεταβολές της επιτάχυνσης στους τρεις βασικούς άξονες στον τρισδιάστατο χώρο. Η επιτάχυνση κατά μήκος κάθε άξονα αναφέρεται απευθείας από το hardware ως τιμές G-Force. Η τιμή 1,0 παριστάνει ένα φορτίο περίπου +1 g κατά μήκος ενός δεδομένου άξονα, ενώ η τιμή -1,0 αντιπροσωπεύει -1 g. Εάν κρατάμε τη συσκευή σε όρθια θέση (με το κεντρικό κουμπί στο κάτω μέρος) μπροστά μας, ο άξονας X είναι θετικός προς τα δεξιά, ο άξονας Y είναι θετικός προς τα πάνω, και ο άξονας Z είναι θετικός δείχνοντας προς το μέρος μας.

Μπορούμε να ανακτήσουμε την τιμή του επιταχυνσιόμετρου με την πρόσβαση στην Input.acceleration. Διαβάζοντας τη μεταβλητή Input.acceleration δεν σημαίνει ότι παίρνουμε τιμές από το hardware. Η Unity παίρνει δείγματα του hardware σε συχνότητα 60Hz και αποθηκεύει το αποτέλεσμα στη μεταβλητή. Η δειγματοληψία του επιταχυνσιόμετρου δεν εμφανίζεται σε συνεπείς χρονικά διαστήματα, και έτσι μπορεί να αναφέρει 2 δείγματα κατά τη διάρκεια ενός καρέ, και 1 δείγμα κατά τη διάρκεια του επόμενου, για αυτό παίρνουμε το μέσο όρο των δειγμάτων που επιστρέφει.

Low-Pass Filter

Οι μετρήσεις του επιταχυνσιόμετρου μπορεί να είναι σπασμωδικές και θορυβώδεις. Εφαρμόζοντας χαμηλοπερατό φιλτράρισμα στο σήμα έχουμε πιο ομαλές μετρήσεις και απαλλαγόμαστε από το θόρυβο υψηλής συχνότητας.

3.3.7 Unity Mobile Scripting

Υπάρχουν μια σειρά από συγκεκριμένες ιδιότητες συσκευών που μπορούμε να έχουμε πρόσβαση.

Anti-Piracy Check

Συχνά πειρατές αποκτούν πρόσβαση σε μια εφαρμογή αφαιρώντας την AppStore DRM προστασία και στη συνέχεια την αναδιανέμουν δωρεάν. Η Unity διαθέτει ένα αντι-πειρατικό έλεγχο που μας επιτρέπει να καθοριστεί αν η εφαρμογή μας έχει αλλάξει μετά την υποβολή στο AppStore.

Μπορούμε να ελέγξουμε αν η εφαρμογή μας είναι γνήσια (not-hacked) με την ιδιότητα Application.genuine. Αν επιστρέψει false, τότε μπορούμε να ειδοποιήσουμε το χρήστη ότι χρησιμοποιεί μια hacked εφαρμογή ή ίσως να απενεργοποιήσουμε την πρόσβαση σε ορισμένες λειτουργίες της εφαρμογής μας.

Το Application.genuineCheckAvailable θα πρέπει να χρησιμοποιείται μαζί με το Application.genuine για να βεβαιωθούμε ότι η ακεραιότητα της εφαρμογής μπορεί πραγματικά να επιβεβαιωθεί. Η πρόσβαση στο Application.genuine είναι μια αρκετά ακριβή λειτουργία και έτσι δεν πρέπει να το κάνουμε κατά τη διάρκεια ενημερώσεων πλαισίου ή άλλου κρίσιμου χρόνου κώδικα.

Vibration Support

Υπάρχουν μια σειρά από ιδιότητες συγκεκριμένων συσκευών που μπορούμε να έχουμε πρόσβαση. Μπορούμε να ενεργοποιήσουμε μια δόνηση καλώντας τη συνάρτηση Handheld.Vibrate. Ωστόσο, οι συσκευές που έχουν έλλειψη hardware όσον αφορά τη δόνηση θα αγνοήσουν μόνο αυτό το κάλεσμα.

Activity Indicator

Τα Mobile λειτουργικά συστήματα έχουν ενσωματωμένους δείκτες δραστηριότητας, που μπορούμε να χρησιμοποιήσουμε κατά τη διάρκεια των αργών εργασιών με τη συνάρτηση Handheld.StartActivityIndicator.

Screen Orientation

Η Unity iOS/Android μας επιτρέπει να ελέγχουμε τον τρέχον προσανατολισμό της οθόνης. Η ανίχνευση μιας αλλαγής στον προσανατολισμό ή ο αναγκασμός σε κάποιο συγκεκριμένο προσανατολισμό μπορεί να είναι χρήσιμα αν θέλουμε να δημιουργήσουμε συμπεριφορές στο παιχνίδι ανάλογα με το πώς ο χρήστης κρατά τη συσκευή.

Μπορούμε να ανακτήσουμε τον προσανατολισμό της συσκευής με την πρόσβαση στην **Screen.orientation**. Ο προσανατολισμός μπορεί να είναι ένα από τα ακόλουθα:

- **Portrait** : Η συσκευή είναι σε λειτουργία πορτρέτου, με τη συσκευή σε όρθια θέση και το κεντρικό κουμπί στο κάτω μέρος.
- **PortraitUpsideDown** : Η συσκευή είναι σε λειτουργία πορτρέτου αλλά ανάποδα, με τη συσκευή σε όρθια θέση και το κεντρικό κουμπί στην κορυφή.
- **LandscapeLeft** : Η συσκευή είναι σε λειτουργία παράλληλα στο εδάφος, με τη συσκευή σε όρθια θέση και το κεντρικό κουμπί στη δεξιά πλευρά.
- **LandscapeRight** : Η συσκευή είναι σε λειτουργία παράλληλα στο εδάφος, με τη συσκευή σε όρθια θέση και το κεντρικό κουμπί στην αριστερή πλευρά.

Μπορούμε να ελέγξουμε τον προσανατολισμό της οθόνης ρυθμίζοντας τη Screen.orientation ή τη ScreenOrientation.AutoRotation. Όταν θέλουμε αυτόματη περιστροφή, μπορούμε να απενεργοποιήσουμε κάποιο προσανατολισμό σε κάθε περίπτωση χωριστά.

Determining Device Generation

Διαφορετικές γενιές συσκευών υποστηρίζουν διαφορετικές λειτουργίες και έχουν πολύ διαφορετικές επιδόσεις. Θα πρέπει ανάλογα με τη γενιά της συσκευής να αποφασίζουμε ποια λειτουργία θα πρέπει να απενεργοποιηθεί για να αντισταθμίσει τις πιο αργές συσκευές. Υπάρχουν μια σειρά από συγκεκριμένες ιδιότητες συσκευής που μπορούμε να έχουμε πρόσβαση για τη συσκευή που χρησιμοποιείται.

Για τα iPhone μπορούμε να βρούμε τη γενιά της συσκευής από την iPhone.generation. Για τα Android, το Android Marketplace κάνει κάποιο επιπλέον φιλτράρισμα συμβατότητας, έτσι ώστε να μην πρέπει να ανησυχούμε όταν μια ARMv7 εφαρμογή που βελτιστοποιείται για OGLES2 προσφέρεται σε κάποιες παλιές αργές συσκευές.

Splash Screen of Mobile Application

Σύμφωνα με το iOS και Android Basic, μια προεπιλεγμένη οθόνη splash θα εμφανιστεί ενώ φορτώνει το παιχνίδι, και προσανατολίζεται σύμφωνα με την επιλογή προεπιλεγμένου προσανατολισμού οθόνης (Default Screen Orientation) στις ρυθμίσεις αναπαραγωγής. Οι χρήστες μπορούν να χρησιμοποιήσουν οποιοδήποτε υφή στο έργο ως μία οθόνη.

iOS

Το μέγεθος της υφής εξαρτάται από τη συσκευή στόχο (320x480 pixels για 1-3^{ης} γενιάς συσκευές, 1024x768 για iPad mini/iPad 1^{ης}/2^{ης} γενιάς, 2048x1536 για iPad 3^{ης}/4^{ης} γενιάς, 640x960 για 4^{ης} γενιάς συσκευές iPhone/iPod και 640x1136 για 5^{ης} γενιάς συσκευές) και παρέχονται υφές που κλιμακώνονται ώστε να ταιριάζουν αν είναι απαραίτητο. Μπορούμε να ρυθμίσουμε τις υφές οθόνης εκκίνησης χρησιμοποιώντας τις ρυθμίσεις Player iOS.

Android

Μπορούμε να ρυθμίσουμε την υφή από τις ρυθμίσεις Android Player. Θα πρέπει να επιλέξουμε τη μέθοδο Splash κλίμακας από τις παρακάτω επιλογές:

- Κέντρο (μόνο κλίμακα προς τα κάτω) : θα επιστήσει την εικόνα στο φυσικό της μέγεθος, εκτός αν είναι πολύ μεγάλη, οπότε θα πρέπει να μειωθεί ώστε να χωρέσει.
- Κλιμάκωση για να χωρέσει (Scale to fit – letter box) : θα επιστήσει την εικόνα έτσι ώστε η μεγαλύτερη διάσταση να ταιριάζει με το μέγεθος της οθόνης ακριβώς. Ο άδειος χώρος γύρω από τις πλευρές στη μικρότερη διάσταση θα συμπληρωθεί με μαύρο χρώμα.

- Κλιμάκωση για να γεμίσει (Scale to fill - cropped) : θα αναβαθμίσει την εικόνα έτσι ώστε η μικρότερη διάσταση να ταιριάζει με το μέγεθος της οθόνης ακριβώς. Η εικόνα θα περικοπεί στη μεγαλύτερη διάσταση.

3.3.8 Ανάπτυξη iOS

Η δημιουργία παιχνιδιών για συσκευές όπως το iPhone και το iPad απαιτεί μια διαφορετική προσέγγιση από αυτή που χρησιμοποιούμε σε παιχνίδια υπολογιστών (desktop PC games). Σε αντίθεση με την αγορά των PC, το hardware στο οποίο στοχεύουμε είναι συγκεκριμένο και όχι τόσο γρήγορο ή ισχυρό όσο ενός υπολογιστή με μια ειδική κάρτα γραφικών. Εξαιτίας αυτού, θα πρέπει να προσεγγίσουμε την ανάπτυξη των παιχνιδιών για αυτές τις πλατφόρμες λίγο διαφορετικά. Επίσης οι διαθέσιμες λειτουργίες της Unity για iOS διαφέρουν ελαφρώς από εκείνες για desktop PCs.

Για να μπορέσουμε να εκτελέσουμε iOS παιχνίδια της Unity σε μια πραγματική συσκευή, θα πρέπει να έχουμε λογαριασμό Apple Developer, ο οποίος πρέπει να εγκριθεί. Ο λογαριασμός αυτός παρέχεται επί πληρωμής και είναι διαθέσιμος μόνο σε λογισμικό MAC OS. Επειδή δεν διαθέτουμε υπολογιστή Apple δεν δοκιμάσαμε το παιχνίδι σε πραγματική συσκευή με iOS, αλλά έχουμε δημιουργήσει το παιχνίδι έτσι ώστε να μπορεί να δουλέψει σε iOS θεωρητικά.

Όταν μετατρέπουμε το παιχνίδι σε εφαρμογή iOS μέσω Unity παράγεται ένα έργο Xcode, το οποίο προετοιμάζει το παιχνίδι κατάλληλα ώστε να υποστηρίζεται από τις συσκευές iOS. Η Unity παρέχει μια σειρά από scripting APIs για να έχουμε πρόσβαση στην multi-touch οθόνη, το επιταχυνσιόμετρο, τη συσκευή συστήματος γεωγραφικής της θέσης και πολλά άλλα. Επίσης υποστηρίζεται η συμπίεση υψηλής απόδοσης από σύνθετες σκηνές με πολλά αντικείμενα.

Η Unity επιτρέπει να καλέσουμε συναρτήσεις γραμμένες σε C, C++ ή Objective-C απευθείας από C# scripts. Ο δυναμικός προγραμματισμός στη JavaScript απενεργοποιείται πάντα στη Unity, όταν στόχευουμε σε iOS, και προστίθεται αυτόματα το «#pragma strict» σε όλα τα scripts. Ο στατικός προγραμματισμός βελτιώνει σημαντικά τις επιδόσεις, το οποίο είναι ιδιαίτερα σημαντικό για τις συσκευές iOS. Όταν αλλάζουμε ένα υπάρχον Unity project σε iOS, θα πάρουμε λάθη μεταγλώττισης, εάν χρησιμοποιούμε δυναμικό προγραμματισμό. Μπορούμε εύκολα να τα διορθώσουμε είτε χρησιμοποιώντας ρητά δηλωμένους τύπους για τις μεταβλητές που προκαλούν σφάλματα ή εκμεταλλευόμενοι την εξαγωγή τύπων

Για λόγους απόδοσης, η MP3 συμπίεση ευνοείται σε iOS συσκευές. Εάν το έργο περιέχει αρχεία ήχου με συμπίεση Ogg Vorbis, θα γίνει εκ νέου συμπίεση σε MP3 κατά τη διάρκεια της κατασκευής.

Τα MovieTextures δεν υποστηρίζονται στο iOS. Αντ 'αυτού, η αναπαραγωγή μιας συνεχούς ροής σε πλήρη οθόνη παρέχεται μέσω λειτουργιών scripting.

Η Unity iOS δεν υποστηρίζει DXT υφές, αλλή η συμπίεση PVRTC υφής υποστηρίζεται εγγενώς από συσκευές iPhone/iPad. Η PVRTC παρέχει υποστήριξη για RGB και RGBA (πληροφορίες χρώματος συν ένα άλφα κανάλι) μορφές υφής και μπορεί να συμπιέσει ένα pixel σε δύο ή τέσσερα κομμάτια. Η μορφή PVRTC μειώνει τη μνήμη και την κατανάλωση του εύρους ζώνης της μνήμης, δηλαδή τον ρυθμό με τον οποίο τα δεδομένα μπορούν να διαβαστούν από τη μνήμη, ο οποίος είναι συνήθως πολύ περιορισμένος σε κινητές συσκευές.

Η CPU και η GPU στο iPhone/iPad μοιράζονται την ίδια μνήμη. Το πλεονέκτημα είναι ότι δεν χρειάζεται να ανησυχούμε για την αυτονομία της μνήμης βίντεο για τις υφές. Το μειονέκτημα είναι ότι θα μοιράζονται το ίδιο εύρος μνήμης για το gameplay και τα γραφικά. Όσο μεγαλύτερο εύρος ζώνης μνήμης αφιερώνουν για τα γραφικά, τόσο λιγότερο θα αφιερώνει για το gameplay και τη φυσική.

Τα περισσότερα χαρακτηριστικά των συσκευών iOS εκτίθενται μέσω των κλάσεων Input και Handheld. Για τα έργα cross-platform, UNITY_IPHONE ορίζεται για την κατάρτιση iOS ειδικός κώδικας C#. Επίσης οι συναρτήσεις OnMouseDown, OnMouseEnter, OnMouseOver, OnMouseExit, OnMouseDown, OnMouseUp, OnMouseDrag δεν υποστηρίζονται.

Unity iOS Basics

Πρέπει αρχικά να πάρουμε έγκριση από το iPhone Developer της Apple. Στην πορεία κατεβάζουμε το SDK και το τρέχουμε στο Apple developer site όπου οργανώνουμε την ομάδα μας και τις συσκευές μας. Μας παρέχεται μια βασική λίστα με τα βήματα που πρέπει να ακολουθήσουμε για να ξεκινήσουμε.

Η Unity δεν μπορεί να τρέξει ενσωματωμένα παιχνίδια στο iPhone Simulator, αλλά μπορεί να χτίσει παιχνίδια στο iPad Simulator αν χρησιμοποιούμε την τελευταία έκδοση SDK. Ωστόσο, ο ίδιο ο προσομοιωτής δεν είναι πολύ χρήσιμος για την Unity, διότι δεν προσομοιώνει όλες τις εισόδους από το iOS ή δεν μιμείται κατάλληλα την απόδοση που μπορούμε να πάρουμε στο iPhone/iPad. Θα πρέπει να δοκιμάσουμε το gameplay απευθείας μέσα στη Unity χρησιμοποιώντας το iPhone/iPad ως τηλεχειριστήριο, ενώ εκτελεί την Unity Remote εφαρμογή. Στη συνέχεια, όταν είμαστε έτοιμοι να δοκιμάσουμε τις επιδόσεις και να βελτιστοποιήσουμε το παιχνίδι, το δημοσιεύουμε σε συσκευές iOS.

Στην αναφορά scripting μέσα στην εγκατάσταση Unity iOS, υπάρχουν κλάσσεις που είναι υπεύθυνες για την λειτουργικότητα της συσκευής που θα τρέξουμε τις εφαρμογές μας, όπως την οθόνη αφής και το επιταχυνσιόμετρο.

Το iOS έχει σχετικά χαμηλό fillrate. Για αυτό αν χρησιμοποιούμε σωματίδια που να καλύπτουν ένα αρκετά μεγάλο τμήμα της οθόνης με πολλαπλές στρώσεις, θα μειώσει την iOS απόδοση ακόμη και με τον πιο απλό shader. Πρέπει να κάνουμε baking τα particle effects σε μια σειρά από υφές off-line. Στη συνέχεια, κατά το χρόνο εκτέλεσης, μπορούμε να χρησιμοποιήσουμε 1-2 σωματίδια να τα εμφανίσουμε μέσω κινούμενες υφές. Μπορούμε να πάρουμε αρκετά αξιοπρεπή αποτελέσματα με ένα ελάχιστο ποσό overdraw με αυτόν τον τρόπο.

Η φυσική μπορεί να είναι ακριβή για iOS, δεδομένου ότι απαιτεί πολλές μετατοπίσεις κινητής υποδιαστολής. Θα πρέπει να αποφεύγουμε εντελώς τους MeshColliders, αν είναι δυνατόν, αλλά μπορούν να χρησιμοποιηθούν εάν είναι πραγματικά απαραίτητο. Για να βελτιώσουμε τις επιδόσεις, χρησιμοποιούμε ένα χαμηλό σταθερό framerate χρησιμοποιώντας Edit -> Time-> Fixed Delta Time. Συνιστάται μια framerate 10-30. Ενεργοποιούμε την rigidbody παρεμβολή για να επιτευχθεί η ομαλή κίνηση, ενώ χρησιμοποιούμε χαμηλά ποσοστά πλαισίων της φυσικής. Προκειμένου να επιτευχθεί εντελώς ρευστό framerate χωρίς ταλαντώσεις, το καλύτερο είναι να πάρουμε σταθερή τιμή deltaTime με βάση το μέσο όρο framerate που το παιχνίδι μας παίρνει στο iOS. Συνιστάται είτε 1:1 ή το μισό του ρυθμού καρέ. Για παράδειγμα, αν έχουμε 30 fps, θα πρέπει να χρησιμοποιήσουμε 15 ή 30 fps για σταθερό ρυθμό καρέ (0.033 ή 0.066).

Η Unity iPhone υποστηρίζει το εγγενές σύστημα plugin, όπου μπορούμε να προσθέσουμε οποιοδήποτε χαρακτηριστικό που χρειαζόμαστε - συμπεριλαμβανομένης της πρόσβασης σε Gallery, μουσική βιβλιοθήκη, iPod Player και οποιοδήποτε άλλο χαρακτηριστικό που εκθέτει το iOS SDK. Η Unity iOS δεν παρέχει ένα API για την πρόσβαση στα χαρακτηριστικά που αναφέρονται μέσω Unity scripts.

Το UnityGUI είναι αρκετά ακριβό, όταν χρησιμοποιούνται πολλοί έλεγχοι. Είναι ιδανικό για να περιορίσουμε τη χρήση του UnityGUI στα μενού του παιχνιδιού ή να έχουμε ελάχιστους ελέγχους GUI ενώ τρέχει το παιχνίδι μας. Είναι σημαντικό να σημειωθεί ότι για κάθε αντικείμενο με ένα script που περιέχει κλήση ενός OnGUI(), θα απαιτηθεί επιπλέον χρόνος επεξεργαστή, ακόμη και αν αυτό είναι ένα άδειο OnGUI() μπλοκ. Είναι καλύτερα να απενεργοποιήσουμε οποιαδήποτε script έχουν OnGUI() κλήση, εάν οι έλεγχοι GUI δεν χρησιμοποιούνται. Μπορούμε να το κάνουμε αυτό με τη επισημαίνοντας το script enabled = false.

Θα ήταν καλύτερα να χρησιμοποιήσετουμε το GUILayout όσο το δυνατόν λιγότερο. Εάν δεν χρησιμοποιούμε καθόλου το GUILayout σε μια OnGUI() κλήση, μπορούμε να απενεργοποιήσουμε όλη την απόδοση GUILayout χρησιμοποιώντας MonoBehaviour.useGUILayout = false; Αυτό διπλασιάζει την GUI rendering απόδοση. Τέλος,

πρέπει να χρησιμοποιούμε όσο το δυνατόν λιγότερα στοιχεία GUI όταν αποδίδουμε 3D σκηνές.

Unity Remote

Η Unity Remote είναι μια εφαρμογή που μας επιτρέπει να χρησιμοποιούμε την iOS συσκευή μας ως τηλεχειριστήριο για το έργο μας στη Unity. Αυτό είναι χρήσιμο κατά τη διάρκεια της ανάπτυξης, δεδομένου ότι είναι πολύ πιο γρήγορο να ελέγξουμε το έργο μας στον επεξεργαστή με τηλεχειριστήριο από το να χτίσουμε και να το περάσουμε στη συσκευή μετά από κάθε αλλαγή.

Η Unity Remote είναι διαθέσιμη για λήψη από το AppStore χωρίς χρέωση. Μπορούμε ακόμη και να δημιουργήσουμε και να αναπτύξουμε την εφαρμογή μόνοι μας κατεβάζοντας την πηγή στην ιστοσελίδα της Unity.

Αρχικά κατεβάζουμε τον πηγαίο κώδικα του έργου και το αποσυμπιέζουμε στην προτιμώμενη θέση μας. Το αρχείο zip περιέχει ένα Xcode project για να φτιάξουμε το Unity Remote και να το εγκαταστήσουμε στη συσκευή μας .

Πρέπει να έχουμε δημιουργήσει το προφίλ και να έχουμε εγκαταστήσει με επιτυχία το iOS builds στη συσκευή μας. Έπειτα ανοίγουμε το αρχείο του Xcode project UnityRemote.xcodeproj. Μόλις το Xcode ξεκινήσει, επιλέγουμε το "Build and Go" για να εγκαταστήσουμεε την εφαρμογή στην iOS συσκευή μας. Υπάρχουν επίσης, έτοιμα παραδείγματα της Apple για να εξοικειωθούμε με το Xcode και iOS.

Μόλις εγκατασταθεί το Unity Remote, πρέπει να βεβαιωθούμε ότι η συσκευή μας είναι συνδεδεμένη μέσω Wi-Fi στο ίδιο δίκτυο με το μηχάνημα ανάπτυξης ή αλλιώς συνδεδεμένο με το μηχάνημα απευθείας μέσω USB. Ξεκινάμε το Unity Remote για το iPhone/iPad, ενώ η Unity τρέχει στον υπολογιστή μας και επιλέγουμε τον υπολογιστή μας από τη λίστα που εμφανίζεται. Κάθε φορά που εισερχόμαστε στο Play mode του επεξεργαστή, η συσκευή μας θα λειτουργεί ως ένα τηλεχειριστήριο που μπορούμε να χρησιμοποιήσουμε για την ανάπτυξη και τη δοκιμή του παιχνιδιού μας. Μπορούμε να ελέγξουμε την εφαρμογή με τη συσκευή ασύρματα και θα δούμε επίσης μια χαμηλής ανάλυσης έκδοση της εφαρμογής στην οθόνη της συσκευής.

Ο Unity iOS editor δεν μπορεί να εξομοιώσει το υλικό της συσκευής τέλεια, έτσι ώστε να μην μπορεί να πάρει την ακριβή συμπεριφορά (απόδοση γραφικών, ανταπόκριση αφής, ήχους της αναπαραγωγής , κλπ) που θα κάναμε σε μια πραγματική συσκευή.

iOS Specific Optimizations

Οι περισσότερες από τις λειτουργίες του UnityEngine namespace υλοποιούνται σε C/C++. Η κλήση μιας C/C++ λειτουργίας από ένα Mono script περιλαμβάνει μια overhead

απόδοση. Μπορούμε να χρησιμοποιήσουμε τη iOS Script Call βελτιστοποίηση (μενού: Edit->Project Settings->Player) για να εξοικονομήσουμε περίπου 1 έως 4 χιλιοστά του δευτερολέπτου ανά πλαίσιο.

Η Unity iOS μας επιτρέπει να αλλάζουμε τη συχνότητα με την οποία η εφαρμογή μας θα προσπαθήσει να εκτελέσει το βρόχο της απόδοσή της, η οποία έχει οριστεί σε 30 καρέ ανά δευτερόλεπτο από προεπιλογή. Μπορούμε να μειώσουμε τον αριθμό αυτό για να εξοικονομήσουμε ενέργεια της μπαταρίας, αλλά φυσικά αυτή η εξοικονόμηση θα προέλθει σε βάρος των ανανεώσεων πλαισίου. Αντίθετα, μπορούμε να αυξήσουμε τη συχνότητα των καρέ για να δώσουμε προτεραιότητα απόδοσης σε σχέση με άλλες δραστηριότητες, όπως είσοδος αφής και επεξεργασίας επιταχυνσιομέτρου. Θα πρέπει να πειραματιστούμε με την επιλογή framerate για να καθορίσουμε πώς αυτή επηρεάζει το gameplay στην περίπτωσή μας.

Εάν η εφαρμογή μας περιλαμβάνει βαρύ υπολογισμό ή απόδοση και μπορεί να διατηρήσει μόνο 15 καρέ ανά δευτερόλεπτο, τότε καθορίζοντας τον επιθυμητό ρυθμό καρέ μεγαλύτερο από δεκαπέντε δεν θα δώσει καμία επιπλέον απόδοση. Η αίτηση πρέπει να βελτιστοποιηθεί αρκετά ώστε να επιτρέψει ένα υψηλότερο framerate.

Αν η είσοδος επιταχυνσιομέτρου επεξεργάζεται πολύ συχνά, τότε η συνολική απόδοση του παιχνιδιού μας μπορεί να μειώνεται. Από προεπιλογή, μια εφαρμογή Unity iOS θα δοκιμάζει το επιταχυνσιόμετρο 60 φορές ανά δευτερόλεπτο. Μπορούμε να δούμε κάποιο όφελος απόδοσης μειώνοντας τη συχνότητα δειγματοληψίας επιταχυνσιομέτρου και μπορούμε ακόμη και να την μηδενίσουμε για τα παιχνίδια που δεν χρησιμοποιούν είσοδο επιταχυνσιομέτρου. Μπορούμε να αλλάζουμε τη συχνότητα επιταχυνσιομέτρου από το iOS Player Settings.

Account Setup

Υπάρχουν μερικά βήματα που πρέπει να ακολουθήσουμε για να μπορέσουμε να δημιουργήσουμε και να εκτελέσουμε οποιονδήποτε κώδικα (συμπεριλαμβανομένων των ενσωματωμένων παιχνιδιών της Unity) στην iOS συσκευή μας. Τα βήματα αυτά είναι προϋπόθεση για τη δημοσίευση δικών μας παιχνιδιών iOS.

1. Αίτηση στην Apple για να γίνουμε Registered iPhone/iPad Developer.
   Μπορούμε να το κάνουμε αυτό μέσα από την ιστοσελίδα της Apple.
2. Αναβαθμίζουμε το λειτουργικό μας σύστημα και την iTunes Εγκατάσταση.
   Αυτές είναι οι απαιτήσεις της Apple ως μέρος της χρήσης του iPhone SDK, αλλά οι απαιτήσεις μπορεί να αλλάξουν από καιρό σε καιρό.
3. Κατεβάζουμε το iPhone SDK.
   Κατεβάζουμε την τελευταία έκδοση του iOS SDK από το κέντρο dev iOS και το εγκαταθιστούμε. Πρέπει να χρησιμοποιήσουμε μόνο την τελευταία έκδοση του SDK και

όχι την beta έκδοση. Το κατέβασμα και η εγκατάσταση του iPhone SDK θα εγκαταστήσει επίσης το Xcode.

4. Παίρνουμε το αναγνωριστικό της συσκευής μας (Device Identifier).
   Συνδέουμε τη συσκευή μας iOS στο Mac με το καλώδιο USB και τρέχουμε το Xcode. Το Xcode θα εντοπίσει το τηλέφωνό μας ως μια νέα συσκευή και θα πρέπει να εγγραφούμε με το «Use For Development», και θα ανοίξει το παράθυρο Organizer. Μόλις δούμε την iOS συσκευή μας στη λίστα συσκευών, την επιλέγουμε και σημειώνουμε τον κωδικό αναγνώρισης της συσκευής μας (το οποίο είναι περίπου 40 χαρακτήρες).
5. Προσθήκη της συσκευής μας.
   Συνδεόμαστε με το κέντρο για προγραμματιστές iPhone και εισέλθουμε στην πύλη του προγράμματος. Πηγαίνουμε στη σελίδα συσκευών και επιλέγουμε το κουμπί «Add Device». Πληκτρολογούμε ένα όνομα για τη συσκευή μας (αλφαριθμητικών χαρακτήρων μόνο) και τον κωδικός αναγνώρισης της συσκευής μας (που σημειώσαμε στο βήμα 5 παραπάνω), και επιλέγουμε το κουμπί «Submit».
6. Δημιουργία ενός πιστοποιητικού.
   Από το iPhone Developer Program Portal, επιλέγουμε τη σύνδεση Certificates και ακολουθούμε τις οδηγίες που αναφέρονται.
7. Κατεβάζουμε και εγκαταθιστούμε το πιστοποιητικό WWDR Intermediate.
   Το download link είναι στην ίδια ενότητα «Certificates» ως WWDR ενδιάμεσων πιστοποιητικών. Μετά τη λήψη, κάνουμε διπλό κλικ στο αρχείο πιστοποιητικού για να το εγκαταστήσουμε.
8. Δημιουργούμε ένα αρχείο Provisioning.
   Το προφίλ τροφοδότησης είναι ένα σύνθετο κομμάτι, και πρέπει να συσταθεί σύμφωνα με τον τρόπο που έχει οργανωθεί η ομάδα μας, ακολουθώντας τις οδηγίες Provisioning στην ιστοσελίδα της Apple Developer.

3.3.9 Ανάπτυξη Android

Η δημιουργία παιχνιδιών για μια συσκευή με λειτουργικό σύστημα Android απαιτεί μια προσέγγιση παρόμοια με εκείνη για την iOS ανάπτυξη. Ωστόσο, το hardware δεν είναι το ίδιο σε όλες τις συσκευές, και αυτό δημιουργεί ζητήματα που δεν υπάρχουν στην iOS ανάπτυξη. Υπάρχουν ορισμένες χαρακτηριστικές διαφορές στην Android έκδοση της Unity σε σχέση με την έκδοση iOS.

Θα πρέπει να έχουμε ρυθμίσει το Android developer περιβάλλον για να μπορέσουμε να ελέγξουμε τα παιχνίδια της Unity στη συσκευή. Αυτό συνεπάγεται τη λήψη και την εγκατάσταση του Android SDK με τις διάφορες Android plaforms και προσθέτοντας τη φυσική μας συσκευή στο σύστημά μας. Αυτή η διαδικασία εγκατάστασης εξηγείται στην ιστοσελίδα προγραμματισμού Android, και μπορεί να υπάρχουν πρόσθετες πληροφορίες που παρέχονται

από τον κατασκευαστή της συσκευής μας. Αυτή είναι μια σύνθετη διαδικασία και πρέπει να ολοκληρωθούν συγκεκριμένες διεργασίες για να μπορέσουμε να εκτελέσουμε κώδικα στην Android συσκευή ή στο Android emulator. Οι οδηγίες σχετικά με τις διεργασίες αυτές υπάρχουν αναλυτικά στο Android developer portal.

Η έκδοση της Unity για Android παρέχει scripting APIs για να αποκτήσουμε πρόσβαση σε διάφορα δεδομένα εισόδου και στις ρυθμίσεις. Μας επιτρέπει να καλέσουμε προσαρμοσμένες συναρτήσεις γραμμένες σε C/C++ απευθείας από C# scripts, ενώ οι συναρτήσεις Java μπορούν να καλεστούν έμμεσα.

Για λόγους απόδοσης, ο δυναμικός προγραμματισμός JavaScript απενεργοποιείται πάντα σε Unity Android, καθώς το «#pragma strict» εφαρμόζεται αυτόματα σε όλο τον κώδικα. Αν ξεκινήσουμε με ένα έργο που αναπτύχθηκε αρχικά για τις πλατφόρμες desktop θα έχουμε μη αναμενόμενα σφάλματα μεταγλώττισης κατά τη μετάβαση σε Android. Ο δυναμικός προραμματισμός είναι το πρώτο πράγμα που πρέπει να διερευνηθεί.

Αν και η Unity Android δεν υποστηρίζει υφές DXT/PVRTC/ATC, θα αποσυμπιέσει την υφή σε μορφή RGB (A) κατά το χρόνο εκτέλεσης, εάν αυτές οι μέθοδοι συμπίεσης δεν υποστηρίζονται από τη συγκεκριμένη συσκευή που χρησιμοποιείται. Αυτό θα μπορούσε να έχει επιπτώσεις στην ταχύτητα και την απόδοση GPU και για αυτό συνιστάται η χρήση της μορφής ETC αντ 'αυτού. Η ETC είναι το de facto πρότυπο μορφής συμπίεσης για Android, και υποστηρίζεται σε όλες τις συσκευές από την έκδοση Android 2.0 και μετά. Ωστόσο, η ETC δεν υποστηρίζει ένα κανάλι άλφα και το RGBA 16-bit μερικές φορές θα είναι το εναλλάσεται καλύτερα μεταξύ του μεγέθους, την ποιότητα και την ταχύτητα απόδοσης, όπου απαιτείται το κανάλι άλφα.

Είναι επίσης δυνατό να δημιουργηθούν ξεχωριστά αρχεία android (.apk) για κάθε μία από τις μορφές DXT/PVRTC/ATC, αφήνοντας το σύστημα φιλτραρίσματος του Android Market να επιλέξει τα σωστά αρχεία για διάφορες συσκευές.

Οι ταινίες υφών δεν υποστηρίζονται στο Android, αλλά μια αναπαραγωγή συνεχούς ροής πλήρους οθόνης παρέχεται μέσω λειτουργιών scripting.

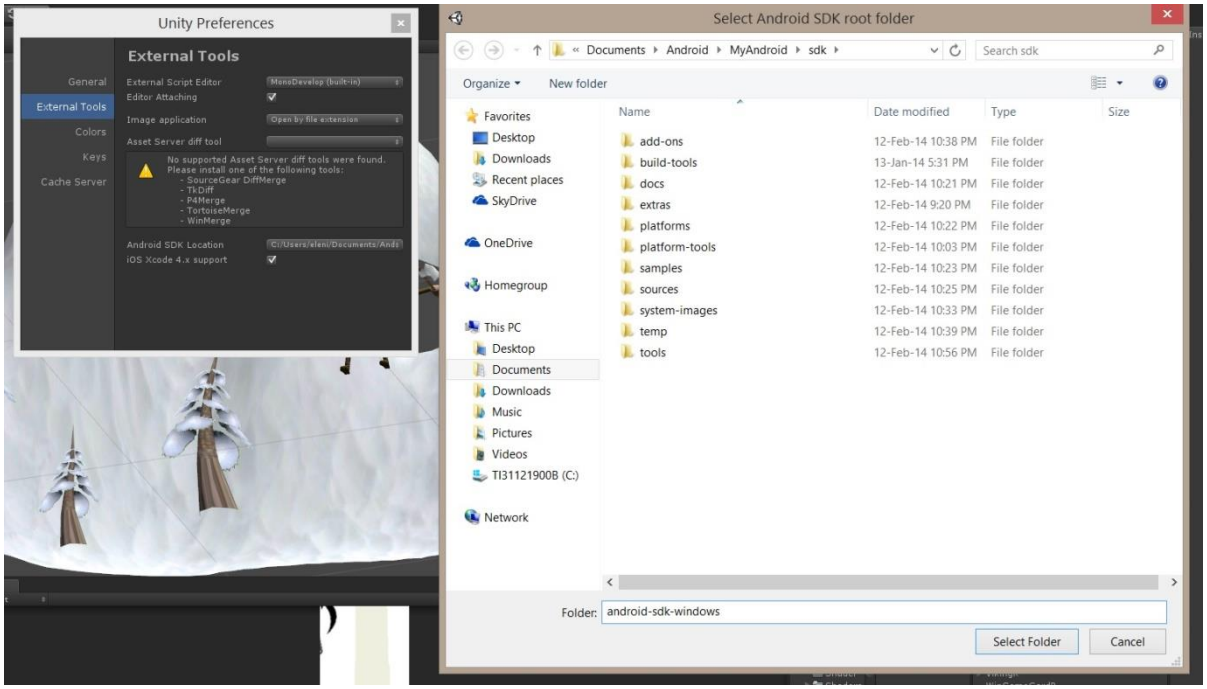
Τα περισσότερα χαρακτηριστικά των Android συσκευών εκτίθενται μέσω των κλάσεων Input και Handheld. Για τα έργα cross-platform, το UNITY_ANDROID ορίζεται για την κατάρτιση της μεταγλώτισης συγκεκριμένα του Android C# κώδικα. Επίσης οι συναρτήσεις OnMouseDown, OnMouseEnter, OnMouseOver, OnMouseExit, OnMouseDown, OnMouseUp, OnMouseDrag δεν υποστηρίζονται.

Android SDK Setup

Υπάρχουν μερικά βήματα που πρέπει να ακολουθήσουμε για να μπορέσουμε να δημιουργήσουμε και να εκτελέσουμε οποιονδήποτε κώδικα στην Android συσκευή μας . Αυτό ισχύει ανεξάρτητα από το αν χρησιμοποιούμε Unity ή γράφουμε τις Android εφαρμογές από το μηδέν.

1. Κατεβάζουμε το Android SDK.
   Πηγαίνουμε στην ιστοσελίδα Android Developer SDK. Κατεβάζουμε και αποσυμπιέζουμε το πιο πρόσφατο Android SDK.
2. Εγκατάσταση του Android SDK.
   Ακολουθούμε τις οδηγίες στην ενότητα Εγκατάστασης του SDK (αν και μπορούμε να παραλείψουμε ελεύθερα τα προαιρετικά εξαρτήματα που σχετίζονται με το Eclipse). Στο βήμα 4 Installing the SDK, πρέπει να προσθέσουμε τουλάχιστον μία Android platform με επίπεδο API ίσο ή υψηλότερο του 9 (Platform 2.3 ή μεγαλύτερη), τα Platform Tools, και τους USB drivers, αν χρησιμοποιείτε τα Windows .
3. Φροντίζουμε να αναγνωριστεί η συσκευή από το σύστημά μας.
   Αυτό μπορεί να είναι δύσκολο, ιδίως στο πλαίσιο των συστημάτων που βασίζονται στα Windows, όπου οι drivers έχουν συνήθως προβλήματα. Επίσης , η συσκευή μας μπορεί να έχει πρόσθετες πληροφορίες ή ειδικούς drivers από τον κατασκευαστή.
   Για Windows : Εάν η συσκευή Android αναγνωρίζεται αυτόματα από το σύστημα, ίσως εξακολουθεί να χρειάζεται να ενημερώσουμε τους drivers με αυτά από το Android SDK. Αυτό γίνεται μέσα από τη Διαχείριση Συσκευών των Windows (Windows Device Manager). Εάν η συσκευή δεν αναγνωρίζεται αυτόματα, χρησιμοποιούμε τους drivers από το Android SDK, ή συγκεκριμένους drivers που παρέχονται από τον κατασκευαστή.
   Για Mac: Αν αναπτύσσουμε σε Mac OSX, τότε δεν απαιτούνται συνήθως πρόσθετα προγράμματα οδήγησης.
   Σημείωση: Δεν πρέπει να ξεχάσουμε να γυρίσουμε σε "Debugging USB" τη συσκευή μας. Πηγαίνουμε στις Ρυθμίσεις -> Επιλογές Developer (Settings -> Developer options), στη συνέχεια, ενεργοποιούμε το USB debugging. Από το Android Jelly Bean 4.2 οι επιλογές Developer είναι κρυφές από προεπιλογή. Για να τις ενεργοποιήσουμε επιλέγουμε στο Ρυθμίσεις -> Σχετικά με το τηλέφωνο -> Build Έκδοση (Settings -> About Phone -> Build Version) πολλές φορές (συγκεκριμένα εφτά φορές). Στη συνέχεια, θα είμαστε σε θέση να αποκτήσουμε πρόσβαση στις Ρυθμίσεις -> Επιλογές Developer.
4. Προσθέτουμε το Android SDK μονοπάτι στην Unity.
   Την πρώτη φορά που δημιουργούμε ένα έργο για το Android (ή αν η Unity αργότερα αποτυγχάνει να εντοπίσει το SDK) θα μας ζητηθεί να εντοπίσουμε το φάκελο όπου έχουμε εγκαταστήσει το Android SDK (θα πρέπει να επιλέξουμε το ριζικό φάκελο της εγκατάστασης SDK). Η θέση του Android SDK μπορεί επίσης να αλλάξει στον

επεξεργαστή επιλέγοντας Unity->Preferences από το μενού και, στη συνέχεια, επιλέγοντας το External Tools στο preferences window.



Android Remote

Το Android Remote είναι μια εφαρμογή Android που κάνει τη συσκευή μας σαν τηλεχειριστήριο για το έργο μας στη Unity. Αυτό είναι χρήσιμο για την ταχεία ανάπτυξη, όταν δεν θέλουμε να περνάμε το project μας στη συσκευή για κάθε αλλαγή.

Για να χρησιμοποιήσουμε το Android Remote, θα πρέπει πρώτον, να βεβαιωθούμε ότι έχουμε την τελευταία έκδοση του Android SDK εγκαταστημένη (αυτό είναι απαραίτητο για να ρυθμίσουμε την θύρα προώθησης της συσκευής). Στη συνέχεια, συνδέουμε τη συσκευή στον υπολογιστή μας με ένα καλώδιο USB και ξεκινάμε την εφαρμογή του Android Remote. Όταν πιέσουμε το πλήκτρο Play στον επεξεργαστή Unity, η συσκευή θα λειτουργεί ως τηλεχειριστήριο και θα περάσει το επιταχυνσιόμετρο και τις εισόδους αφής γεγονότων στο παιχνίδι που τρέχει.

Αντιμετώπιση προβλημάτων Android ανάπτυξη

Αν δεν μπορεί να εντοπιστεί η πλατφόρμα, πρέπει να βεβαιωθούμε δεν έχουμε δεν έχουμε περισσότερα από ένα αντίγραφο του Android SDK. Η Java πρέπει να είναι ενημερωμένη με μια σταθερή έκδοση. Κατεβάζουμε ένα νέο αντίγραφο της τελευταίας έκδοσης του ADT Bundle, εγκαταθιστώντας σωστά ακολουθώντας τις οδηγίες. Πρέπει να διαμορφώσουμε τη Unity με το εγκατεστημένο Android SDK .

Αν η Unity αποτυγχάνει να εγκαταστήσει την εφαρμογή στη συσκευή μας, πρέπει να βεβαιωθούμε ότι ο υπολογιστής μας πρέπει να μπορεί να δεί και να επικοινωνεί με τη συσκευή, και ελέγχουμε τα μήνυματα σφάλματος στην Unity κονσόλα.

Εάν λάβουμε ένα μήνυμα λάθους "Δεν είναι δυνατή η εγκατάσταση APK, αποτυχία πρωτοκόλλου" κατά τη διάρκεια μιας κατασκευής, τότε αυτό σημαίνει ότι η συσκευή είναι συνδεδεμένη σε θύρα χαμηλής ισχύος USB. Αν συμβεί αυτό, συνδέουμε τη συσκευή σε μια θύρα USB στον ίδιο τον υπολογιστή.

Αν η εφαρμογή μας κολλάει αμέσως μετά την έναρξη, πρέπει να βεβαιωθούμε ότι δεν προσπαθούμε να χρησιμοποιήσουμε NativeActivity με συσκευές που δεν το υποστηρίζουν. Αφαιρούμε οποιαδήποτε μητρικά plugins έχουμε, απενεργοποιούμε το stripping και χρησιμοποιούμε adb logcat για να πάρει την αναφορά σφάλματος από τη συσκευή μας.

Χαρακτηριστικά που ακόμα δεν υποστηρίζονται από τη Unity Android:

- Μη τετράγωνες υφές δεν υποστηρίζονται από τη μορφή ETC.
- Υφές ταινιών δεν υποστηρίζονται, αντ 'αυτού χρησιμοποιούμε την αναπαραγωγή συνεχούς ροής πλήρους οθόνης.
- OnMouseEnter, OnMouseOver, OnMouseExit, OnMouseDown, OnMouseUp και OnMouseDrag γεγονότα δεν υποστηρίζεται από το Android.
- Δυναμικά χαρακτηριστικά, όπως Duck Typing δεν υποστηρίζονται. Χρησιμοποιούμε το #pragma αυστηρά για τον κώδικά μας για να αναγκάσει τον compiler να αναφέρει δυναμικά χαρακτηριστικά, όπως τα λάθη.
- Video streaming μέσω WWW class δεν υποστηρίζεται.

Υποστήριξη για το Split Application Binary (.OBB)

Σύμφωνα με τις ρυθμίσεις Player Settings | Publishing Settings βρίσκουμε την επιλογή για να χωρίσουμε τη δυαδική εφαρμογή (.apk) σε αρχεία επέκτασης (.apk + .obb). Αυτός ο μηχανισμός είναι απαραίτητος μόνο όταν δημοσιεύουμε στο Google Play Store, εάν η εφαρμογή είναι μεγαλύτερη από 50 MB.

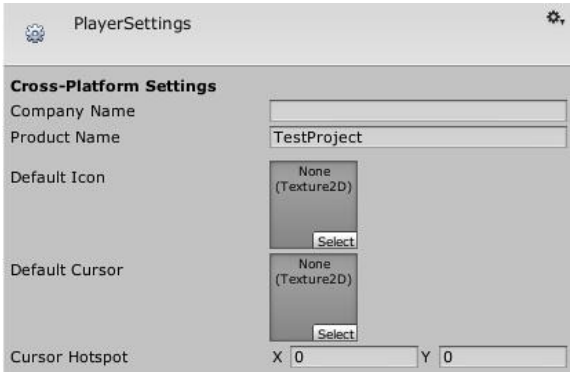
Όταν η επιλογή Split Application Binary ενεργοποιηθεί το εκτελέσιμο παιχνίδι και τα δεδομένα θα χωρίσουν, παράγοντας .apk (κύρια δυαδική εφαρμογή) που αποτελείται μόνο

από το εκτελέσιμο αρχείο (Java, Native) κώδικα (περίπου 10MB), οποιαδήποτε και όλα τα script/plugin κώδικα, και τα δεδομένα για την πρώτη σκηνή. Οτιδήποτε άλλο (όλες οι πρόσθετες σκηνές, οι πόροι , η συνεχή ροή περιουσιακών στοιχείων ...), θα είναι στη σειρά ξεχωριστά σε ένα αρχείο επέκτασης APK (.obb).

- Όταν ξεκινάμε ένα .apk φτιαγμένο με Split Application Binary ενεργοποιώντας την εφαρμογή θα ελέγξει για να δει εάν μπορεί να προσπελάσει το αρχείο .obb από τη θέση του σχετικά με την sdcard.
- Αν δεν μπορεί να βρεθεί το αρχείο επέκτασης (.obb), μόνο το πρώτο επίπεδο μπορεί να προσεγγιστεί (δεδομένου ότι το υπόλοιπο των δεδομένων είναι σε .obb).
- Το πρώτο επίπεδο, τότε υποχρεούται να κάνει το αρχείο .obb διαθέσιμο στην sdcard, πριν η εφαρμογή να μπορεί να προχωρήσει για να φορτώσει επόμενες σκηνές/δεδομένα.
- Εάν το .obb έχει βρει το Application.dataPath θα στραφεί από .apk μονοπάτι, σε σημείο .obb. Δεν είναι απαραίτητο να κατεβάσουμε το .obb.
- Τα περιεχόμενα του .obb δεν χρησιμοποιούνται ποτέ με το χέρι. Αντιμετωπίζουμε πάντα το .apk + .obb ως ένα μοναδικό πακέτο, με τον ίδιο τρόπο που θα αντιμετωπίζαμε ένα ενιαίο μεγάλο .apk .

Η Split Application Binary επιλογή δεν είναι ο μόνος τρόπος για να χωρίσει ένα .Apk σε .apk/.obb (άλλες επιλογές περιλαμβάνουν 3rd party plugins/δέσμες στοιχείων ενεργητικού/etc), αλλά αυτός είναι ο μόνος αυτόματος μηχανισμός σχισίματος που υποστηρίζεται επίσημα.

- Το αρχείο επέκτασης (.obb) μπορεί (αλλά δεν είναι απαραίτητο, στη σημερινή του μορφή τουλάχιστον) να φιλοξενείται στο Google Play servers .
- Αν το αρχείο .obb δημοσιεύεται μαζί με την .apk στο Google Play , θα πρέπει να περιλαμβάνει και τον κωδικό για να κατεβάσουμε το .obb (για αυτές τις συσκευές που το απαιτούν , και για τα σενάρια όπου το .obb χάνεται)
- Το κατάστημα περιουσιακών στοιχείων έχει ένα plugin (προσαρμοσμένο από τα παραδείγματα Google Apk Expansion), το οποίο θα κατεβάσει το .obb και θα το βάλει στη σωστή θέση για την sdcard.
- Όταν χρησιμοποιούμε το asset store plugin θα πρέπει να καλέσουμε αυτό το plugin από την πρώτη σκηνή.
- Το asset store plugin μπορεί επίσης να χρησιμοποιηθεί για να κατεβάσουμε το δημιουργημένο .obb με κάποιο άλλο τρόπο



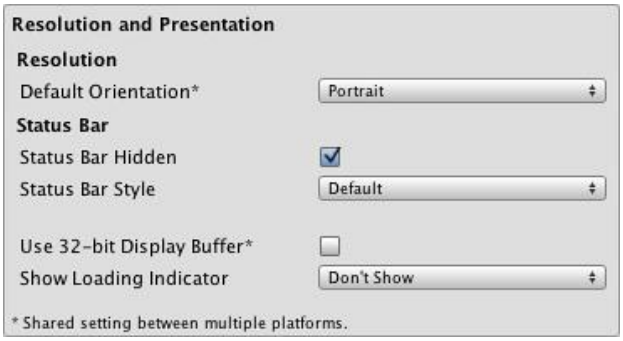
(ενιαίο αρχείο δεδομένων , πακέτα περιουσιακών στοιχείων , κ.λπ.).

- Για να ελέγξουμε το obb downloader, η Android συσκευή πρέπει να συνδεθεί ως σωστός λογαριασμός google (λογαριασμός tester).

3.3.10 Player Settings

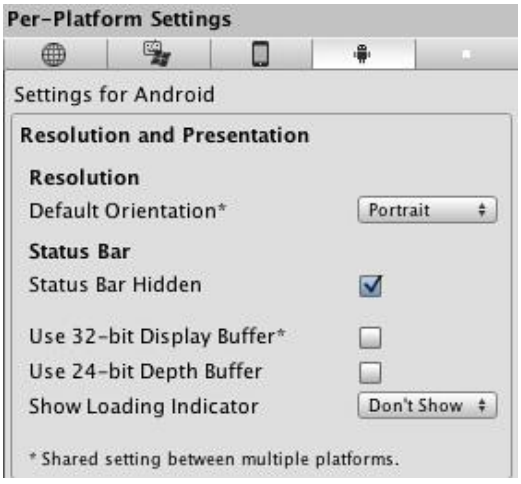
Στις Ρυθμίσεις Player μπορούμε να ορίσουμε διάφορες παραμέτρους (ειδική πλατφόρμα) για το τελικό παιχνίδι που θα φτιάξουμε στη Unity. Μερικές από αυτές τις τιμές, για παράδειγμα χρησιμοποιούνται στο Resolution Dialog που ξεκινά όταν ανοίγουμε ένα αυτόνομο παιχνίδι, άλλες έχουν χρησιμοποιηθεί από το Xcode όταν χτίζεται το παιχνίδι μας για τις iOS συσκευές, γι 'αυτό είναι σημαντικό να τα συμπληρώσουμε σωστά.

Cross-Platform Properties:



- Company Name: Το όνομα της εταιρείας μας. Αυτό χρησιμοποιείται για να εντοπίσουμε το αρχείο προτιμήσεων.
- Product Name: Το όνομα που θα εμφανίζεται στη γραμμή μενού, όταν το παιχνίδι μας λειτουργεί και χρησιμοποιείται για να εντοπίσουμε το αρχείο προτιμήσεων επίσης.
- Default Icon: Προεπιλεγμένο εικονίδιο της εφαρμογής που θα είναι σε κάθε πλατφόρμα.

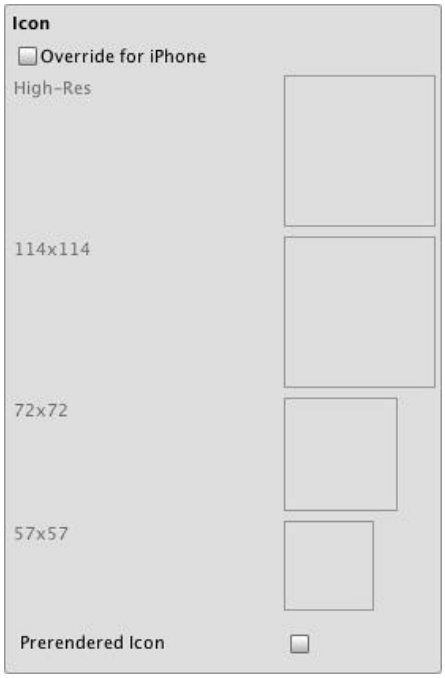
Resolution (Κοινές ρυθμίσεις μεταξύ των iOS και Android συσκευές):



- Default Orientation:
  - Portrait: Η συσκευή είναι σε λειτουργία πορτρέτου, με τη συσκευή σε όρθια θέση και το κεντρικό κουμπί στο κάτω μέρος.
  - Portrait Upside Down: Η συσκευή είναι σε λειτουργία πορτρέτου, αλλά ανάποδα, με τη συσκευή σε όρθια θέση και το κεντρικό κουμπί στην κορυφή (διατίθεται μόνο με Android OS 2.3 και νεότερες).
  - Landscape Right: Η συσκευή είναι σε λειτουργία τοπίου , με τη συσκευή σε όρθια θέση και το κεντρικό κουμπί στην αριστερή πλευρά (διατίθεται μόνο με Android OS 2.3 και νεότερες) .
  - Landscape Left: Η συσκευή είναι σε λειτουργία τοπίου, με τη συσκευή σε όρθια θέση και το κεντρικό κουμπί στη δεξιά πλευρά .
- Auto Rotation settings:

- Use Animated Autorotation: Όταν ελεγχθεί, η αλλαγή προσανατολισμού είναι κινούμενη. Αυτό ισχύει μόνο όταν ο Default orientation έχει οριστεί σε Auto Rotation.

- Allowed Orientations for Auto Rotation:
  - Portrait: Όταν επιλεχθεί, επιτρέπεται κατακόρυφο προσανατολισμό. Αυτό ισχύει μόνο όταν ο προεπιλογμένος προσανατολισμός έχει οριστεί σε Auto Rotation.
  - Portrait Upside Down Όταν ελέγχονται , πορτραίτο ανάποδα προσανατολισμό επιτρέπεται . Αυτό ισχύει μόνο όταν Προεπιλογή προσανατολισμός έχει οριστεί σε Αυτόματη περιστροφή .
  - Landscape Right: Όταν είναι επιλεγμένο, επιτρέπεται προσανατολισμός με το κεντρικό κουμπί στην αριστερή πλευρά. Αυτό ισχύει μόνο όταν ο προεπιλεγμένος προσανατολισμός έχει οριστεί σε Αυτόματη περιστροφή.
  - Landscape Left: Όταν είναι επιλεγμένο, επιτρέπεται προσανατολισμό με το κεντρικό κουμπί στη δεξιά πλευρά. Αυτό ισχύει μόνο όταν ο προεπιλεγμένος προσανατολισμός έχει οριστεί σε Αυτόματη περιστροφή.
- Status Bar:
  - Status Bar Hidden: Καθορίζει αν η γραμμή κατάστασης είναι αρχικά κρυμμένη όταν ξεκινά η εφαρμογή.
  - Status Bar Style: Καθορίζει το στυλ της γραμμής κατάστασης και της φόρτωσης της εφαρμογής: Default, Black Translucent, Black Opaque
- Use 32-bit Display Buffer: Καθορίζει εάν θα πρέπει να δημιουργηθεί Display Buffer για να κρατήσει τις τιμές χρώματος 32 - bit (16 -bit από προεπιλογή). Το χρησιμοποιούμε το αν δούμε banding, ή χρειαζόμαστε άλφα στις ImageEffects μας, καθώς θα δημιουργήσει RTs στην ίδια μορφή με το Display Buffer. Δεν υποστηρίζεται σε συσκευές που τρέχουν προ-Gingerbread OS (θα αναγκαστούν να χρησιμοποιήσουν 16 - bit) .
- Use 24-bit Depth Buffer: Αν οριστεί Depth Buffer θα δημιουργηθεί για να κρατήσει (τουλάχιστον) τις τιμές βάθους 24 - bit. Το χρησιμοποιούμε μόνο αν δούμε «z-fighting» ή άλλα αντικείμενα, καθώς μπορεί να έχει επιπτώσεις απόδοσης.
- Show Loading Indicator: Δείκτης για την ένδειξη φόρτισης:
  - Don't Show: Κανένας δείκτης.
  - White Large: Δείκτης φαίνεται μεγάλος και σε λευκό.
  - White: Δείκτης εμφανίζεται σε κανονικό μέγεθος στο λευκό.
  - Gray: Δείκτης εμφανίζεται σε κανονικό μέγεθος σε γκρι.

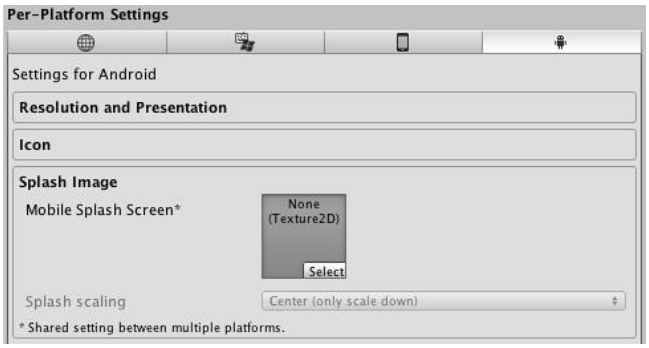


Icon:

Override for Android: Ελέγχουμε εάν θέλουμε να ορίσουμε ένα προσαρμοσμένο εικονίδιο που θα θέλαμε να χρησιμοποιείται για το παιχνίδι Android μας. Διαφορετικά μεγέθη της εικόνας θα πρέπει να συμπληρωθούν στα παρακάτω τετράγωνα.

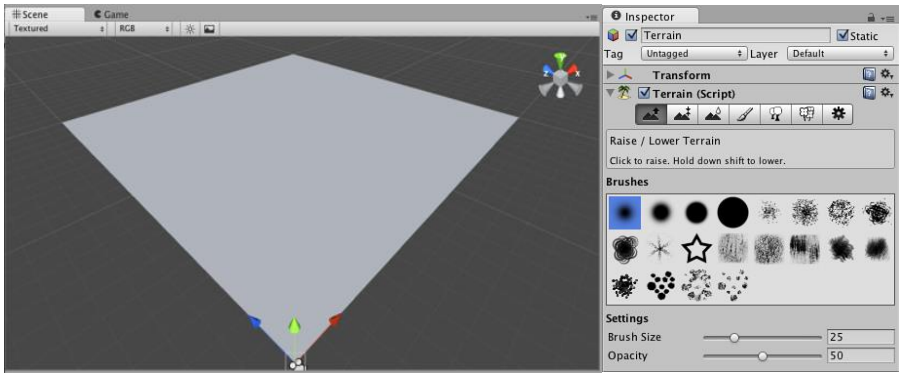
Splash Image:

- Mobile Splash Screen: Καθορίζει την υφή που πρέπει να χρησιμοποιείται από την iOS Splash Screen. Το στάνταρ μέγεθος Splash Screen είναι 320x480. (Αυτό μοιράζεται μεταξύ Android και iOS).
- Splash Scaling: Καθορίζει πώς θα είναι η κλιμάκωση splash image στη συσκευή.



3.3.11 Terrain

Το Terrain είναι το έδαφος της πίστας του παιχνίδιού, πάνω στο οποίο διαδραματίζονται τα αντικείμενά μας. Η Unity διαθέτει πολλά εργαλεία με τα οποία μπορούμε να επεξεργαστούμε το terrain και να φτιάξουμε τον κόσμο του παιχνιδιού όπως επιθυμούμε.



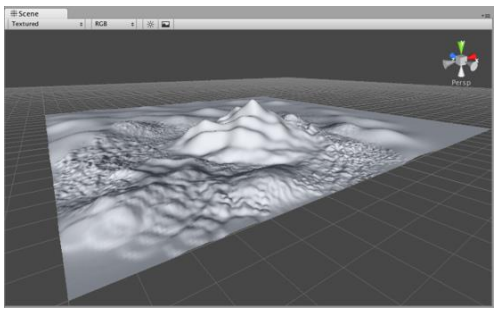
Μπορούμε να ρυθμίσουμε τις διαστάσεις που θέλουμε να έχει η πίστα, και με τις διάφορες βούρτσες που διαθέτει να ρυθμίσουμε το ύψος δημιουργώντας βουνά, το χρώμα και το σχέδιο του εδάφους, να προσθέσουμε δέντρα, γρασίδι βράχια, θάμνους, και κάθε είδους βλάστηση, τα οποία αλληλεπιδρούν με τον αέρα του περιβάλλοντος, και ακόμη να το φωτίσουμε με lightmap.

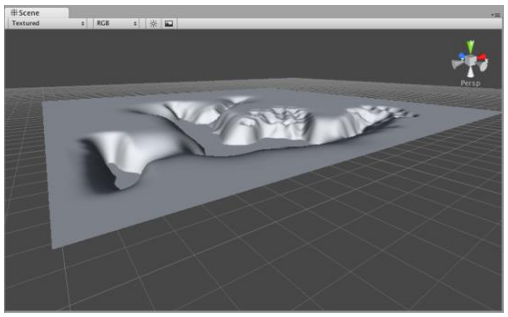
Τα εδάφη λειτουργούν διαφορετικά από τα άλλα GameObjects. Υπάρχουν εργαλεία για να αλλάξουμε το ύψος, να ζωγραφίσουμε και να χειριστούμε το έδαφος, ή να τοποθετήσουμε δέντρα και βράχια. Τα περισσότερα από τα εργαλεία κάνουν χρήση μιας βούρτσας-πινέλου, διαφορετικών μεγεθών και λειτουργιών, με την οποία ζωραφίζουμε απευθείας επάνω στο Terrain. Αν θέλουμε να το αλλάξουμε, μπορούμε να τροποποιήσουμε τις τιμές του μετασχηματισμού Θέσης. Αυτό μας επιτρέπει να μετακινήσουμε το Terrain, αλλά δεν μπορούμε να το περιστρέψουμε ή να το κλιμακώσουμε.

Σημείωση απόδοσης σε Κινητά

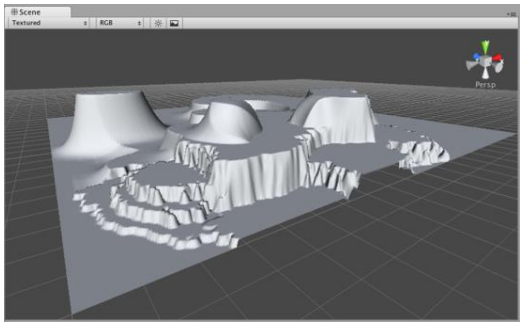
Η απόδοση εδάφους είναι αρκετά ακριβή, οπότε η μηχανή εδάφους (terrain engine) δεν είναι πολύ πρακτική για τις lower-end κινητές συσκευές. Επειδή στο παιχνίδι χρησιμοποιήθηκε το Terrain, και σε αρκετά πολύπλοκες μορφές, το παιχνίδι αποδίδει καλύτερα σε κινητές συσκευές με διπύρηνο επεξεργαστή και πάνω, ενώ σε μονοπύρηνο τα γραφικά της κάθε πίστας κολλάνε.

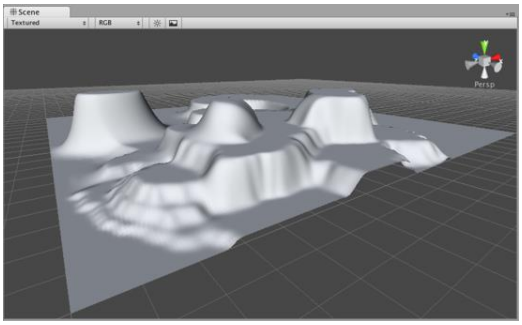
Όλα τα εργαλεία επεξεργασίας εδάφους είναι πολύ απλά. Με το πρώτο εργαλείο αυξάνεται και μειώνεται το ύψος του εδάφους.



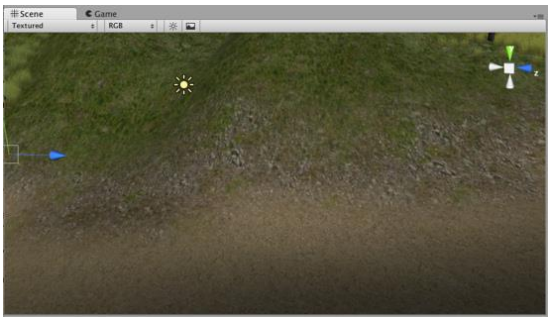


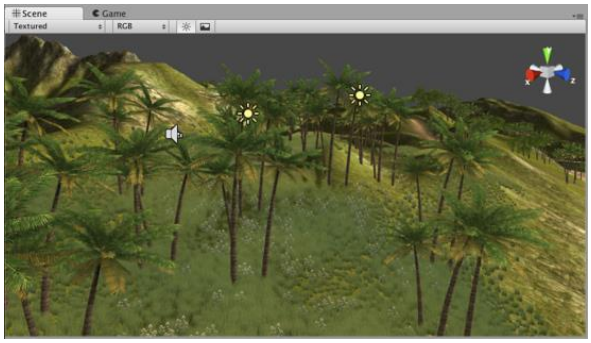
Με το δεύτερο εργαλείο καθορίζουμε το ύψος-στόχο που θέλουμε να έχει το εδάφος. Μόλις το έδαφος φτάσει το ύψος του στόχου, σταματάει να αυξάνεται ή να μειώνεται. Με το τρίτο εργαλείο οι τυχόν διαφορές ύψους ομαλοποιούνται στην περιοχή που ζωγραφίζουμε.



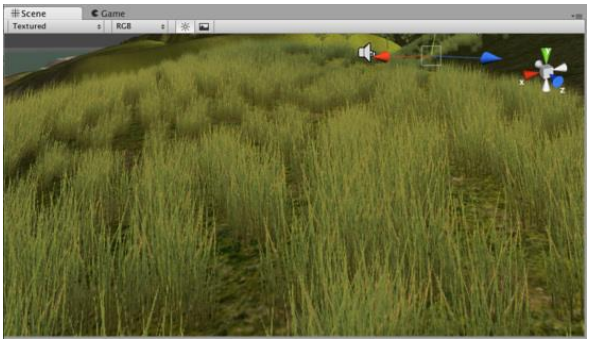


Με το τέταρτο εργαλείο διακοσμούμε το τοπίο του εδάφους τοποθετώντας υφές στα πλακάκια του Terrain. Μπορούμε να συνδυάσουμε υφές στο Terrain για πιο ομαλή μετάβαση από το ένα μέρος στο άλλο, ή για χάρη της ποικιλίας. Οι υφές εδάφους ονομάζεται splat maps, και μπορούμε να ορίσουμε πολλές επαναλαμβανόμενες υφές υψηλής ανάλυσης οι οποίες δένουν μεταξύ τους αυθαίρετα, χρησιμοποιώντας χάρτες άλφα. Επειδή οι υφές δεν είναι μεγάλες σε σύγκριση με το μέγεθος του εδάφους, η κατανομή μεγέθους των υφών είναι πολύ μικρή.





Η μηχανή Terrain υποστηρίζει τα δέντρα με το πέμπτο εργαλείο. Μπορούμε να βάλουμε χιλιάδες δέντρα σε ένα Terrain, τα οποία αποδίδοντα στο παιχνίδι με ένα πρακτικό ρυθμό καρέ. Τα δέντρα κοντά στην κάμερα είναι 3D, ενώ όταν είναι πολύ μακριά είναι 2D εικόνες, οι οποίες ενημερώνονται αυτόματα ώστε να προσανατολιστούν σωστά όπως τα βλέπουμε από διαφορετικές οπτικές γωνίες. Έτσι έχουμε ένα λεπτομερές περιβάλλον δέντρων πολύ απλό στην απόδοση. Έχουμε πλήρη έλεγχο σε αλλαγές των παραμέτρων ώστε να μπορούμε να έχουμε την καλύτερη απόδοση που χρειαζόμαστε.



Το έκτο εργαλείο μας επιτρέπει να ζωγραφίσουμε γρασίδι, πέτρες, ή άλλες διακοσμήσεις στο Έδαφος. Δεν χρειάζεται να δημιουργηθεί ένα πλέγμα για το γρασίδι, μόνο μια υφή.

3.3.12 Image Effect Reference

Όλα τα εφέ εικόνας χρησιμοποιούν τη λειτουργία OnRenderImage όπου κάθε MonoBehavior συνδέεται με μια κάμερα.

Η ομίχλη (Global Fog) των image effect δημιουργεί ομίχλη βασισμένη σε κάμερα. Όλοι οι υπολογισμοί γίνονται στον παγκόσμιο χώρο του παιχνίδιού, και μπορούμε να ελέγξουμε το ύψος της ομίχλης. Το εφέ αυτό απαιτεί μια κάρτα γραφικών για κινητό OpenGL ES 2.0 με στήριξη βάθος υφής. Όλα τα εφέ εικόνας απενεργοποιούνται αυτόματα όταν δεν μπορεί να τρέξει στην κάρτα γραφικών του χρήστη.

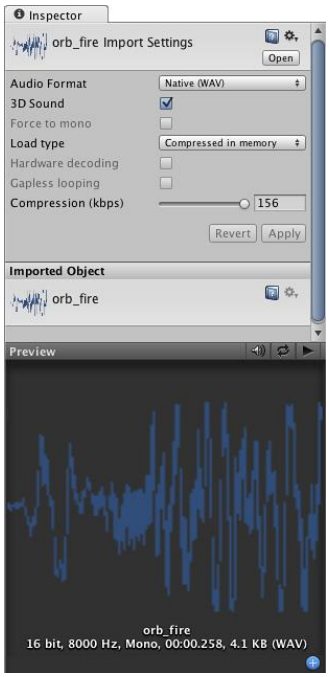
3.3.13 Ήχος (Audio Components)

Η Unity υποστηρίζει στο λειτουργικό iOS και Android δύο τύπους ήχου: αποσυμπιεσμένο ήχο (Uncompressed Audio) ή συμπιεσμένο ήχο MP3. Οποιοσδήποτε τύπος αρχείου ήχου που εισάγεται στο παιχνίδια θα μετατραπεί σε μία από αυτές τις μορφές.

Τύπος αρχείου μετατροπής

- .AIFF : Εισαγωγή ως ασυμπίεστος ήχος για σύντομα ηχητικά εφέ, μπορεί εξαρχής να είναι συμπιεσμένος στον Επεξεργαστή.
- .WAV : Εισαγωγή ως ασυμπίεστος ήχος για σύντομα ηχητικά εφέ, μπορεί εξαρχής να είναι συμπιεσμένος στον Επεξεργαστή.
- .MP3 : Για τα iOS η εισαγωγή γίνεται ως Apple Native συμπιεσμένη μορφή για μεγαλύτερο χρονικό διάστημα μουσικών κομματιών. Μπορεί να παίξει στο hardware της συσκευής. Για τα Android η εισαγωγή γίνεται ως MP3 συμπιεσμένη μορφή για μεγαλύτερο χρονικό διάστημα μουσικών κομματιών.
- .OGG : Για τα iOS η OGG συμπιεσμένη μορφή ήχου είναι ασυμβίβαστη με τη συσκευή iPhone. Για τα Android η OGG συμπιεσμένη μορφή ήχου είναι συμβατή με κάποιες συσκευές Android, έτσι η Unity δεν το υποστηρίζει για την πλατφόρμα Android. Και στις δύο περιπτώσεις πρέπει να χρησιμοποιήσουμε MP3 συμπιεσμένο ήχο αντ 'αυτού.

Με την εισαγωγή των αρχείων ήχου, είναι δυνατή η προσάρτησή τους σε οποιοδήποτε GameObject. Το αρχείο ήχου θα δημιουργήσει ένα Component πηγής ήχου (Audio Source) της επιλογής μας σε ένα GameObject.



Audio Source

Η πηγή ήχου (Audio Source) αναπαράγει ένα ηχητικό κλιπ (Audio Clip) στη σκηνή. Εάν το Audio Clip είναι ένα 3D κλιπ, η πηγή αναπαράγεται σε μια δεδομένη θέση και αλλάζει την έντασή της αναλόγως την απόσταση. Ο ήχος μπορεί να απλώνεται μεταξύ των ηχείων (Spread) και να μεταμορφώνεται μεταξύ 3D και 2D (PanLevel). Αυτό μπορεί να ελεγχθεί εξ 'αποστάσεως με καμπύλες falloff. Επίσης εάν ο ακροατής είναι μέσα σε μία ή πολλαπλές ζώνες αντήχησης, εφαρμόζεται αντηχήση στην πηγή. Στην επαγγελματική έκδοση, μεμονωμένα φίλτρα μπορούν να εφαρμοστούν σε κάθε πηγή ήχου για έναν ακόμη πιο εμπλουτισμένο ήχο.

Audio Clip

Οι πηγές ήχου δεν κάνουν τίποτα χωρίς το κλιπ ήχου. Το Audio Clip είναι το πραγματικό αρχείο ήχου που θα αναπαραχθεί, και περιέχει τα δεδομένα του ήχου που χρησιμοποιούνται από την Audio Source. Η Πηγή είναι σαν ένας ελεγκτής για την έναρξη και τη διακοπή της αναπαραγωγής των εν λόγω κλιπ, και τροποποιεί άλλες ιδιότητες του ήχου. Η Unity υποστηρίζει μονό, στερεοφωνικό και πολυκάναλο ήχο περιουσιακών στοιχείων (μέχρι οκτώ κανάλια).

Η είσοδος του ήχου είναι είτε Native ή Compressed, και υποστηρίζονται οι περισσότερες κοινές μορφές. Τα αρχεία που εισάγονται είναι της μορφής .Aif, .Wav, .Mp3, και .Ogg και

μπορούν να εισαχθούν μονάδες tracker σε μορφές .Xm, .Mod, .It, και .S3M, τα οποία συμπεριφέρονται με τον ίδιο τρόπο όπως και οποιαδήποτε άλλα περιουσιακά στοιχεία ήχου. Η προεπιλεγμένη λειτουργία είναι Native, όπου τα μουσικά δεδομένα από το αρχικό αρχείο εισάγονται αμετάβλητα. Ωστόσο η Unity μπορεί να συμπιέσει τα δεδομένα ήχου κατά την εισαγωγή, απλά επιτρέπει την Compressed επιλογή του εισαγωγέα.

- Native: (WAV, AIFF) Χρησιμοποιείται για σύντομα ηχητικά εφέ. Τα δεδομένα του ήχου είναι μεγαλύτερα, αλλά οι ήχοι δεν αποκωδικοποιούνται κατά το χρόνο εκτέλεσης. Δίνουν μεγαλύτερη ποιότητα ήχου χωρίς να αυξάνεται η επιβάρυνση της CPU, αλλά τα αρχεία σε αυτές τις μοφές είναι συνήθως πολύ μεγαλύτερα από τα συμπιεσμένα αρχεία. Μπορούν να χρησιμοποιηθούν αρχεία modules (.Mod, .It, .S3m, .Xm) για να προσφέρουν πολύ υψηλής ποιότητας ήχο.
- Compressed : Τα ηχητικά δεδομένα είναι μικρά, αλλά θα πρέπει να αποσυμπιεστούν από τη CPU κατά το χρόνο εκτέλεσης, το οποίο συνεπάγεται επιπλέον επεξεργασία. Η αποσυμπίεση μπορεί να γίνει κατά τη διάρκεια του παιχνιδιού ή μόλις φορτώσει το παιχνίδι. Η μορφή του συμπιεσμένου ήχου είναι καλύτερη για μεγάλα αρχεία, όπως η μουσική υπόκρουση ή διαλόγου. Επίσης μπορούμε να ρυθμίσουμε την μέγεθος της συμπίεσης. Ανάλογα με το στόχο, η Unity θα κωδικοποιήσει τον ήχο σε Ogg Vorbis (Mac/PC/Κονσόλες) ή MP3 (Mobile πλατφόρμες). Για την καλύτερη ποιότητα ήχου, παρέχουμε τον ήχο σε ασυμπίεστη μορφή, όπως WAV ή AIFF (που περιέχει δεδομένα PCM) αφήνοντας στη Unity την κωδικοποίηση. Εάν στοχεύουμε μόνο σε Mac και PC πλατφόρμες (συμπεριλαμβανομένων δύο standalones και webplayers) η εισαγωγή ενός αρχείου Ogg Vorbis δεν θα υποβαθμίσει την ποιότητα. Ωστόσο σε κινητές συσκευές, τα Ogg Vorbis και MP3 αρχεία θα πρέπει να κωδικοποιηθούν εκ νέου σε MP3 κατά την εισαγωγή , η οποία θα εισαγάγει μια μικρή υποβάθμιση της ποιότητας.

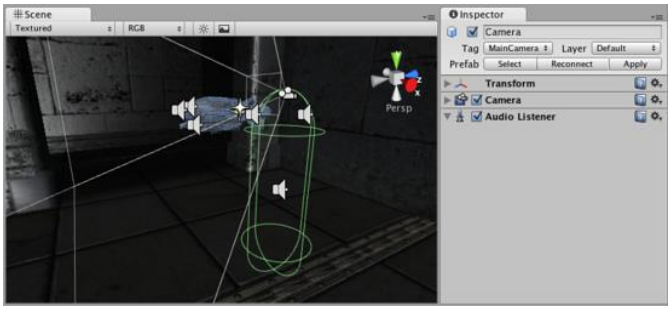
Κάθε αρχείο ήχου που εισάγεται είναι διαθέσιμο από τον κώδικα ως κλιπ ήχου. Τα κλιπ πρέπει να χρησιμοποιούνται σε συνδυασμό με τον ήχο της πηγής και έναν Audio Listener, προκειμένου να δημιουργήσουν ήχο. Όταν συνδέσουμε το κλιπ ήχου σε ένα αντικείμενο στο παιχνίδι, προστίθεται ένα Audio Source component στο αντικείμενο, το οποίο έχει ένταση, και ένα πλήθος άλλων ιδιοτήτων. Ενώ μια πηγή παίζει, ένας Audio ακροατής μπορεί να "ακούσει" όλες τις πηγές εντός εμβέλειας , και ο συνδυασμός των πηγών αυτών δίνει τον ήχο που όντως θα ακούγεται από τα ηχεία. Μπορεί να υπάρχει μόνο ένας Listener ήχου στη σκηνή, και αυτός συνήθως συνδέεται με την κύρια κάμερα.

Στις κινητές πλατφόρμες ο συμπιεσμένος ήχος κωδικοποιείται ως MP3 για να επωφεληθούν από την αποσυμπίεση του Hardware που είναι ταχύτερη, η οποία όμως μπορεί να αφαιρέσει δείγματα στο τέλος του κλιπ και ενδεχομένως να σπάσει ένα «τέλειο-looping" κλιπ, για αυτό πρέπει να ορίσουμε ακριβώς τα όρια του MP3 δείγματος για την αποφυγή της

περικοπής του. Σε iOS συσκευές, για λόγους απόδοσης και βελτίωσης των επιδόσεων, το κλιπ ήχου μπορεί να αναπαραχθεί με τη χρήση του κωδικοποιητή hardware της Apple. Εάν ο αποκωδικοποιητής hardware δεν είναι διαθέσιμος, η αποσυμπίεση θα γίνει με τον αποκωδικοποιητή του λογισμικού, όπου χρησιμοποιείται κατά προτίμηση ο αποκωδικοποιητής λογισμικού της Apple αντί αυτόν της Unity (FMOD).

Audio Listener

Ο ακροατής ήχου (Audio Listener) ενεργεί ως συσκευή μικροφώνου. Δέχεται εισόδους από οποιαδήποτε πηγή ήχου στη σκηνή και αναπαράγει ήχους μέσα από τα ηχεία του υπολογιστή. Για τις περισσότερες περιπτώσεις αρκεί να συνδέσουμε τον ακροατή στην κύρια κάμερα. Αν ένας ηχητικός ακροατής (audio listener) είναι εντός των ορίων της αντήχησης Reverb Zone εφαρμόζεται σε όλους τους ήχους στη σκηνή. Επίσης μπορούν να εφαρμοστούν ηχητικά εφέ στον ακροατή τα οποία εφαρμόζονται σε όλους τους ήχους στη σκηνή. Ο Audio Listener δεν έχει ιδιότητες, απλά πρέπει να προστεθεί για να λειτουργήσει και προστίθεται στην κύρια κάμερα από προεπιλογή.



Ο Audio Listener λειτουργεί σε συνδυασμό με τις πηγές ήχου. Όταν ο ακροατής ήχου είναι συνδεδεμένος με ένα GameObject στη σκηνή, τυχόν πηγές που είναι αρκετά κοντά στον ακροατή εντάσσονται στον ήχο που αναπαράγεται από τα ηχεία του υπολογιστή. Κάθε σκηνή μπορεί να έχει μόνο έναν Audio Listener για να λειτουργήσει σωστά. Θα πρέπει να συνδέσουμε τον Audio Listener είτε στην κύρια κάμερα ή στο GameObject που αντιπροσωπεύει τον παίκτη. Αν οι πηγές είναι 3D, ο ακροατής θα μιμηθεί τη θέση, την ταχύτητα και τον προσανατολισμό του ήχου στον 3D κόσμο. Οι πηγές 2D αγνοούν οποιαδήποτε 3D επεξεργασία

3.3.14 Υφές (Textures)

Η Unity υποστηρίζει όλες τις μορφές της εικόνας. Ακόμα και όταν εργαζόμαστε με αρχεία Photoshop, εισάγονται χωρίς να διαταραχθεί η μορφή τους. Αυτό μας επιτρέπει να εργαζόμαστε με ένα ενιαίο αρχείο υφής. Οι διαστάσεις των υφών πρέπει να είναι με βάση τη δύναμη του δύο (π.χ. 32x32, 64x64, 128x128, 256x256, κλπ.). Αφού εισαχθεί η υφή, την αναθέτουμε σε ένα υλικό (Material). Το υλικό μπορεί στη συνέχεια να εφαρμοστεί σε ένα πλέγμα (mesh), σε ένα Particle συστήμα ή σε μια GUI υφή. Χρησιμοποιώντας τις ρυθμίσεις εισαγωγής, μπορεί επίσης να μετατραπεί σε ένα ή Cubemap Normalmap για διαφορετικούς τύπους εφαρμογών στο παιχνίδι.

Οι υφές φέρνουν τα πλέγματα, τα σωματίδια και τις διεπαφές στη ζωή. Πρόκειται για την εικόνα ή την ταινία αρχείων που έχουμε θέσει πάνω ή τυλίξει γύρω από τα αντικείμενα, και έχουν πολλές ιδιότητες.

Οι shaders που χρησιμοποιούμε για τα αντικείμενα θέτουν συγκεκριμένες απαιτήσεις σχετικά με την υφή, αλλά η βασική αρχή είναι ότι μπορούμε να βάλουμε οποιοδήποτε αρχείο εικόνας μέσα στο παιχνίδι. Εάν πληρεί τις απαιτήσεις μεγέθους, θα εισαχθεί και βελτιστοποιηθεί για τη χρήση του παιχνιδιού. Αυτό ισχύει και για τα πολλαπλά στρώματα Photoshop ή τα αρχεία TIFF, τα οποία ενώνονται σε ένα στρώμα ή παραμένουν άθικτα.

Οι υφές προέρχονται από τα αρχεία εικόνας στο Project Folder. Το Texture Type menu μας επιτρέπει να επιλέξουμε τον τύπο της υφής που θέλουμε να δημιουργήσουμε από το αρχείο προέλευσης της εικόνας.

Πρέπει να προσέξουμε το μέγεθος και την ποιότητα των υφών αναλόγων την πλατφόρμα που στοχεύουμε. Αν δεν στοχεύουμε σε ένα συγκεκριμένο hardware, μπορούμε να χρησιμοποιήσουμε τη συμπίεση. Αν χρειαστεί μπορούμε να αποθηκεύσουμε ένα εξωτερικό κανάλι άλφα (η RGBA16 bit συμπίεση υποστηρίζεται από όλους τους προμηθευτές Hardware) και εξακολουθούμε να επωφελούμαστε από τη χαμηλότερη υφή.

Υφές μπορούν να εισαχθούν από αρχεία DDS αλλά μόνο DXT ή ασυμπίεστες μορφές pixel υποστηρίζονται αυτήν τη στιγμή. Εάν η εφαρμογή χρησιμοποιεί μια μη υποστηριζόμενη συμπίεση υφής, η υφή θα είναι ασυμπίεστη σε RGBA 32 και θα αποθηκευτεί στη μνήμη μαζί με τα συμπιεσμένα. Έτσι θα χάσουμε σε χρόνο αποσυμπίεσης υφών και σε μνήμη με την διπλή αποθήκευσή τους. Μπορεί επίσης να έχει πολύ αρνητικές επιπτώσεις στις επιδόσεις της απόδοσης.

Η ενότητα μπορεί να διαβάσει τις παρακάτω μορφές αρχείων: PSD, TIFF, JPG, TGA, PNG, GIF, BMP, IFF, PICT, μπορεί να εισάγει πολλαπλά στρώματα PSD και TIFF αρχεία τα οποία ενωποιούνται αυτόματα κατά την εισαγωγή, αλλά τα στρώματα διατηρούνται στο ενεργητικό τους, έτσι ώστε να μην χαθούν τα αρχικά αρχεία. Επιτρέπει να έχουμε μόνο ένα αντίγραφο υφών που μπορούμε να χρησιμοποιήσουμε από το Photoshop, μέσα από εφαρμογές 3D modeling στη Unity.

Τα ιδανικά μεγέθη υφή πρέπει να είναι δυνάμεις του δύο στις πλευρές , όπως 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 κλπ. pixels. Οι υφές δεν χρειάζεται να είναι τετράγωνες, δηλαδή το πλάτος μπορεί να είναι διαφορετικό από το ύψος. Κάθε πλατφόρμα μπορεί να επιβάλει το μέγιστο μέγεθος υφής. Είναι δυνατή η χρήση άλλων (όχι δύναμη του δύο - "NPOT") μεγεθών υφής με το Unity. Το μέγεθος της υφής που δεν είναι δύναμη του δύο γενικά χρειάζεται λίγο περισσότερη μνήμη και μπορεί να είναι πιο αργή για να διαβαστεί από την GPU. Αν η πλατφόρμα ή GPU δεν υποστηρίζει NPOT μεγέθη υφής, τότε η Unity θα

αναβαθμίσει το μέγεθος της υφής μέχρι την επόμενη δύναμη του, το οποίο θα χρησιμοποιήσει ακόμα περισσότερη μνήμη και θα κάνει πιο αργή τη φόρτωση (αυτό συμβαίνει πάντα σε Flash και κάποιες παλαιότερες συσκευές Android).

UV Mapping

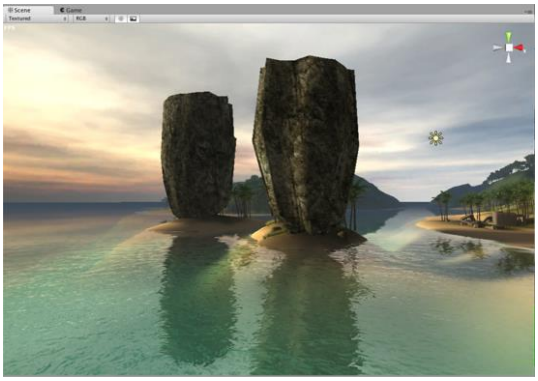
Κατά τη χαρτογράφηση μιας 2D υφής σε ένα 3D μοντέλο γίνεται ένα είδος περιτυλίγματος. Αυτό ονομάζεται UV χαρτογράφηση και γίνεται σε εφαρμογές σχεδιασμού 3D. Με αυτόν τον τρόπο δίνουμε χρώμα και σχέδιο στα τρισδιάστατα μοντέλα. Μέσα στη Unity μπορούμε να αναβαθμίσουμε και να μετακινήσουμε την υφή χρησιμοποιώντας υλικά. Η κλιμάκωση και οι λεπτομερείς χάρτες είναι ιδιαίτερα χρήσιμοι.

Normal Maps

Οι κανονικοί χάρτες χρησιμοποιούνται από τους shaders για να κάνουν τα μοντέλα με λίγα πολύγωνα να φαίνονται σαν να περιέχουν περισσότερες λεπτομέρειες. Οι normal maps κωδικοποιούνται ως εικόνες RGB, και μπορούμε να δημιουργήσουμε ένα κανονικό χάρτη από μια ασπρόμαυρη εικόνα.

Λεπτομερείς Χάρτες

Για τη δημιουργία του εδάφους (terrain), χρησιμοποιούμε την κύρια υφή για να δείξουμε πού υπάρχουν περιοχές με χόρτα, βράχια άμμο, κλπ. Αν το εδάφος είναι πολύ μεγάλο η εικόνα θα είναι θολή. Οι λεπτομερείς υφές εξαφανίζουν τη θολή εικόνα καθώς έχουν μικρές λεπτομέρειες στην κύρια υφή όσο πλησιάζουμε κοντά. Στις λεπτομερείς υφές ένα ουδέτερο γκρι είναι αόρατο, το άσπρο είναι δύο φορές πιο φωτεινό και το μαύρο είναι εντελώς μαύρο στην κύρια υφή.



Αντικατοπτρισμοί (Cube Maps)

Αν θέλουμε να χρησιμοποιήσουμε την υφή για αντικατοπτρισμό (π.χ. να χρησιμοποιούμε τις Ανακλάσεις ενσωματωμένων shaders), θα πρέπει να χρησιμοποιήσουμε Cubemap υφές.

3.3.15 Φυσική

Για να έχει ένα αντικείμενο φυσική συμπεριφορά σε ένα παιχνίδι πρέπει να επιταχύνει σωστά και να επηρεάζεται από τις συγκρούσεις, τη βαρύτητα και τις άλλες δυνάμεις. Η ενσωματωμένη μηχανή φυσικής της Unity παρέχει συστατικά που χειρίζονται τη προσομοίωση της φυσικής. Με λίγες ρυθμίσεις παραμέτρων μπορούμε να δημιουργήσουμε αντικείμενα που συμπεριφέρονται παθητικά με ρεαλιστικό τρόπο, όπως η μετακίνησή τους μετά από συγκρούσεις και πτώσεις, αλλά δεν θα αρχίσουν να κινούνται από μόνα τους. Με τον έλεγχο της φυσικής μέσω του κώδικα, μπορούμε να δώσουμε σε ένα αντικείμενο τη δυναμική ενός οχήματος, μιας μηχανής ή ακόμα και ενός κινούμενου κομματιού υφάσματος.

Η Unity έχει ενσωματωμένη τη φυσική μηχανή NVIDIA PhysX. Στην πραγματικότητα υπάρχουν δύο ξεχωριστές μηχανές φυσικής, η μία για 3D φυσική και η άλλη για 2D φυσική. Οι κύριες έννοιες είναι ταυτόσημες μεταξύ των δύο κινητήρων (εκτός από τη διάσταση σε 3D), αλλά υλοποιούνται με διαφορετικά συστατικά . Έτσι υπάρχει Rigidbody συστατικό για τη 3D φυσική και ένα ανάλογο Rigidbody 2D για τη 2D φυσική.

Rigidbodies

Τα Rigidbodies επιτρέπουν στα GameObjects να ενεργούν υπό τον έλεγχο της φυσικής, καθώς μπορούν να λαμβάνουν δυνάμεις και ροπές για να κάνουν τα αντικείμενα να κινούνται με ρεαλιστικό τρόπο. Είναι η φυσική προσομοίωση των αντικειμένων. Κάθε GameObject πρέπει να περιέχει ένα Rigidbody ώστε να επηρεάζεται από τη βαρύτητα, να ενεργεί από ένα σύνολο δυνάμεων μέσω scripting, ή να αλληλεπιδρά με άλλα αντικείμενα μέσα από τη μηχανή φυσικής PhysX NVIDIA.

Το Rigidbody είναι το κύριο συστατικό που ενεργοποιεί τη φυσική συμπεριφορά ενός αντικειμένου. Προσαρτώντας ένα Rigidbody στο αντικείμενο έχουμε άμεση ανταπόκτιση με τη βαρύτητα. Εάν προστεθούν επίσης ένα ή περισσότερα συστατικά Επιταχυντή (Collider components) τότε το αντικείμενο θα μετακινείται από εισερχόμενες συγκρούσεις.

Δεδομένου ότι ένα Rigidbody component αναλαμβάνει την κίνηση του αντικειμένου στο οποίο είναι συνδεμένο, η μετακίνησή του πρέπει να γίνεται μέσω κώδικα εφαρμόζοντας δυνάμεις ώθησης σε αυτό αφήνοντας στη μηχανή φυσικής τον υπολογισμό των αποτελεσμάτων και όχι αλλάζοντας τις ιδιότητες μετασχηματισμού του, όπως τη θέση και την περιστροφή.

Υπάρχουν ορισμένες περιπτώσεις όπου ένα αντικείμενο έχει ένα Rigidbody χωρίς να ελέγχεται από την μηχανή φυσικής η κίνηση του. Αυτό το είδος της μη φυσικής κίνησης που παράγεται από τον κώδικα είναι γνωστή ως κινηματική κίνηση (kinematic motion). Το Rigidbody συστατικό έχει μια ιδιότητα που ονομάζεται Is Kinematic που το αφαιρεί από τον έλεγχο της μηχανής φυσικής και το αφήνει να κινηθεί κινηματικά από μια δέσμη ενεργειών του κώδικα. Είναι δυνατόν να αλλάξουμε την τιμή του Is Kinematic από τον κώδικα για να επιτρέψουμε τη φυσική να επηρεάζει ή να μην επηρεάζει το αντικείμενο. Εάν μετακινήσουμε το Transform ενός μη-Κινηματικού Rigidbody μπορεί να μην συγκρούεται σωστά με άλλα αντικείμενα. Αντ 'αυτού θα πρέπει να κινηθεί το Rigidbody εφαρμόζοντας δυνάμεις και ροπές σε αυτό. Μπορούμε επίσης να προσθέσουμε τις αρθρώσεις στα rigidbodies για να κάνουμε τη συμπεριφορά πιο πολύπλοκη, όπως μια πόρτα ή ένα γερανό με μια αλυσίδα swinging .

Μπορούμε επίσης να χρησιμοποιήσουμε Rigidbodies σε οχήματα, αεροπλάνα, αυτοκίνητα, 4 Wheel Colliders, ρομπότ με την προσθήκη διαφόρων αρθρώσεων και ένα

κώδικα εφαρμογής των δυνάμεων. Τα Rigidbodies συχνά χρησιμοποιούνται σε συνδυασμό με colliders .

Colliders

Οι συνιστώσες του επιταχυντή (Collider components) ορίζουν το σχήμα ενός αντικειμένου για τους σκοπούς φυσικών συγκρούσεων. Ένας επιταχυντής, ο οποίος είναι αόρατος, δεν χρειάζεται να είναι ακριβώς το ίδιο σχήμα με πλέγμα του αντικειμένου και μια πρόχειρη προσέγγιση είναι συχνά πιο αποτελεσματική και διακριτή στο gameplay. Οι απλούστεροι επιταχυντές σε 3D, είναι τα Box Collider , Sphere Collider και Capsule Collider. Οποιοσδήποτε αριθμός από αυτά μπορεί να προστεθεί σε ένα αντικείμενο για να δημιουργήσει μια ένωση επιταχυντών. Με προσεκτική τοποθέτηση και το κατάλληλο μέγεθος, η ένωση επιταχυντών μπορεί να προσεγγίσει το σχήμα ενός αντικειμένου αρκετά καλά, διατηρώντας παράλληλα χαμηλή επιβάρυνση του επεξεργαστή. Περαιτέρω ευελιξία μπορεί να αποκτηθεί από επιπλέον επιταχυντές σε αντικείμενα των παιδιών του γονέα-GameObject. Ωστόσο πρέπει να υπάρχει μόνο ένα Rigidbody και αυτό θα πρέπει να τοποθετηθεί στο αντικείμενο ρίζας στην ιεραρχία.

Οι Επιταχυντές συνεργαζονται με τα Rigidbodies να υπάρξει φυσική στο παιχνίδι. Τα Rigidbodies επιτρέπουν στα αντικείμενα να ελέγχονται από τη φυσική, και οι Colliders αφήνουν τα αντικείμενα να συγκρούονται μεταξύ τους. Οι Επιταχυντές πρέπει να προστεθούν στα αντικείμενα ανεξάρτητα από τα Rigidbodies. Ένας Collider δεν χρειάζεται απαραίτητα ένα Rigidbody, αλλά χρειάζεται προκειμένου το αντικείμενο να κινηθεί ως αποτέλεσμα των συγκρούσεων.

Υπάρχουν κάποιες περιπτώσεις όμως , όπου ακόμη και η ένωση επιταχυντών δεν είναι αρκετά ακριβής. Σε 3D μπορούμε να χρησιμοποιήσουμε Mesh Colliders για να ταιριάξει ο επιταχυντής ακριβώς με το σχήμα του αντικειμένου. Επίσης ένας Mesh Collider δεν είναι σε θέση να συγκρουστεί με άλλον επιταχυντή πλέγματος, δηλαδή τίποτα δεν θα συμβεί όταν έρθουν σε επαφή, μπορούμε όμως να χρησιμοποιήσουμε Convex με τον οποίο μπορούν να συγκρουστούν οι κινούμενοι χαρακτήρες. Οι Mesh Colliders πρέπει γενικά να προσεγγίζουν τη γεωμετρία και το σχήμα των αντικειμένων.

Επιταχυντές μπορούν να προστεθούν σε ένα αντικείμενο χωρίς Rigidbody για τη δημιουργία πατώματος, τοίχων και άλλων ακίνητων στοιχείων μιας σκηνής, και αναφέρονται ως στατικοί επιταχυντές. Η αλλαγή της θέσης Transform των στατικών επηρεάζει σε μεγάλο βαθμό από την απόδοση του κινητήρα φυσικής, για αυτό πρέπει να αποφεύγεται. Οι επιταχυντές σε ένα αντικείμενο που έχει ένα Rigidbody είναι γνωστοί ως δυναμικοί επιταχυντές. Οι στατικοί επιταχυντές μπορούν να αλληλεπιδράσουν με τους δυναμικούς επιταχυντές, αλλά δεδομένου ότι δεν έχουν ένα Rigidbody, δεν θα κινούνται ως απόκριση των συγκρούσεων.

Triggers

Ένας εναλλακτικός τρόπος χρήσης των Colliders είναι η επισήμανσή τους ως Trigger, και έτσι θα αγνοηθούν από την μηχανή φυσικής. Για να ενεργοποηθούν δύο Triggers, ή ένα Trigger με έναν Collider, όταν έρθουν σε επαφή πρέπει ένα από αυτά να περιέχει Rigidbody.

Το scripting μπορεί να ανιχνεύσει πότε συμβαίνουν συγκρούσεις και να ξεκινήσει ενέργειες χρησιμοποιώντας τη συνάρτηση OnCollisionEnter. Μπορούμε επίσης να χρησιμοποιήσουμε την μηχανή φυσικής απλά για να ανιχνεύσουμε πότε ένας επιταχυντής εισέρχεται στον χώρο ενός άλλου, χωρίς να δημιουργεί σύγκρουση. Ένας επιταχυντής Trigger (χρησιμοποιώντας την ιδιότητα Is Trigger) δεν συμπεριφέρεται ως στερεό αντικείμενο αλλά επιτρέπει άλλους επιταχυντές να περνάνε μέσα από αυτόν ενημερώνοντάς για την επαφή τους μέσω της συνάρτησης OnTriggerEnter.

Character Controllers

Οι Ελεγκτές χαρακτήρων είναι κατάλληλοι για τη δημιουργία ανθρωποειδών χαρακτήρων, όπως ο κύριος χαρακτήρας σε παιχνίδι τρίτου προσώπου, ο FPS shooter ή οποιοιδήποτε χαρακτήρες του εχθρού. Ο χαρακτήρας σε ένα παιχνίδι πρώτου ή τρίτου προσώπου χρειάζεται συχνά κάποια σύγκρουση με βάση τη φυσική, έτσι ώστε να μην πέφτει από το δάπεδο ή να μην εισχωρούν τα πόδια του μέσα από τοίχους. Συνήθως όμως η επιτάχυνση του χαρακτήρα και η κίνηση δεν είναι φυσικά ρεαλιστικές, έτσι ώστε αυτός να μπορέσει να επιταχυνθεί, να σταματήσει και να αλλάξει κατεύθυνση σχεδόν αμέσως χωρίς να επηρεάζεται από την ορμή.

Οι Ελεγκτές χαρακτήρων δεν επηρεάζονται από τις δυνάμεις, αλλά μπορούν να ωθήσουν τα Rigidbodies εφαρμόζοντας τις δυνάμεις από μια δέσμη ενεργειών. Οι Ελεγκτές χαρακτήρων είναι αφύσικοι για αυτό είναι απαραίτητη η χρήση ενός Rigidbody που μας επιτρέπει να χρησιμοποιήσετε τις αρθρώσεις και τις δυνάμεις στον χαρακτήρα μας. Είναι πάντα ευθυγραμμισμένοι κατά μήκος του άξονα Y, έτσι ώστε να πρέπει να χρησιμοποιήσουμε ένα Rigidbody αν ο χαρακτήρας πρέπει να αλλάξει τον προσανατολισμό στο χώρο.

Σε 3D φυσική μπορεί να δημιουργηθεί αυτό το είδος της συμπεριφοράς με τη χρήση ενός ελεγκτή χαρακτήρων (Character Controller). Αυτό το στοιχείο δίνει στο χαρακτήρα μια κάψουλα επιταχυντή απλή σε σχήμα, που είναι πάντα όρθια. Ο ελεγκτής έχει δικές του λειτουργίες για τη ρύθμιση της ταχύτητας και την κατεύθυνση του αντικειμένου, αλλά σε αντίθεση με αληθινούς επιταχυντές το rigidbody δεν είναι απαραίτητο και οι επιπτώσεις της ορμής δεν είναι ρεαλιστικές .

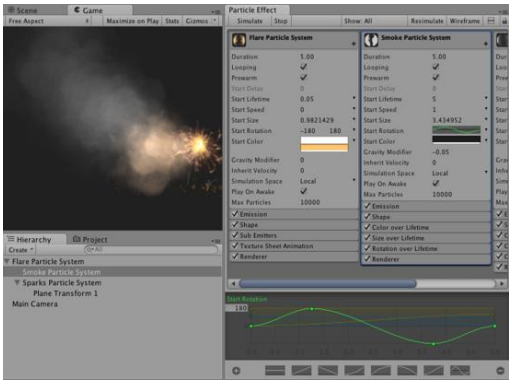
Ένας Character Controller δεν μπορεί να περάσει μέσα από τους στατικούς επιταχυντές σε μια σκηνή, και έτσι ακολουθεί το πατώματα και εμποδίζεται από τους τοίχους . Μπορεί να ωθήσει rigidbody αντικείμενα στην άκρη ενώ κινείται, αλλά δεν επιταχύνεται από τις

εισερχόμενες συγκρούσεις. Έτσι μπορούμε να χρησιμοποιήσουμε 3D colliders για να δημιουργήσουμε μια σκηνή στην οποία ο ήρωας θα περπατήσει, αλλά δεν θα περιορίζεται από ρεαλιστική φυσική συμπεριφορά.

3.3.16 Visual Effects Reference

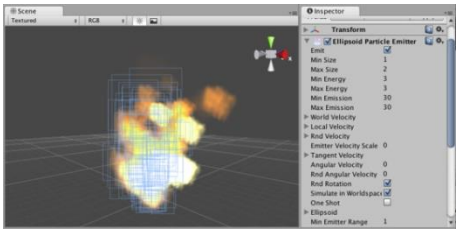
Συστήματα σωματιδίων (Particle Systems)

Τα σωματίδια είναι ουσιαστικά εικόνες 2D σε 3D χώρο. Χρησιμοποιούνται κυρίως για εφέ, όπως για τη δημιουργία σύννεφων καπνού, ατμού, φωτιάς, εκρήξεων, καταρακτών, σταγονιδίων νερού, άλλων ατμοσφαιρικών επιδράσεων και φύλλων. Ένα σύστημα σωματιδίων αποτελείται από τρεις ξεχωριστές συνιστώσες: πομπός σωματιδίων (Particle Emitter), Particle Animator, και Particle Renderer. Για τη δημιουργία στατικών σωματιδίων χρησιμοποιήσουμε ένα πομπό Σωματιδίων και έναν Renderer μαζί. Ο Animator Σωματιδίων κινεί σωματιδία σε διαφορετικές κατευθύνσεις και αλλάζει τα χρώματα. Μπορούμε επίσης να έχουμε πρόσβαση σε κάθε σωματίδιο ενός συστήματος σωματιδίων μέσω scripting.



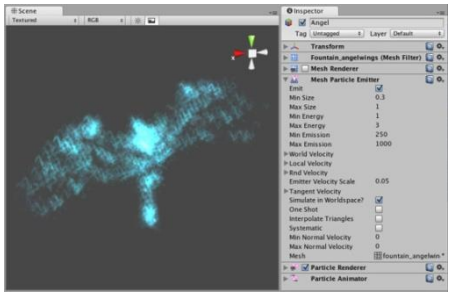
Μπορούμε να μεταβάλουμε τις τιμές, το μέγεθος, την ταχύτητά τους με το χρόνο, το χρώμα, το σχήμα, την έκτασή τους στο χώρο, την κατεύθυνση στην οποία θα κινηθούν πάνω στους άξονες, την έντασή τους και τη διάρκειά τους. Ανήκουν επίσης στην κατηγορία των Particle Effects.

Ένα σύστημα σωματιδίων αποτελείται από ένα προκαθορισμένο σύνολο μονάδων που μπορεί να ενεργοποιηθεί και να απενεργοποιηθεί. Αρχικά μόνο λίγες μονάδες είναι ενεργοποιημένες. Η πρόσθεση ή η αφαίρεση μονάδων αλλάζει τη συμπεριφορά του συστήματος σωματιδίων. Τα διάφορα στοιχεία που είναι διαθέσιμα στο legacy σύστημα σωματιδίων είναι τα παρακάτω.



Ellipsoid Particle Emitter (Legacy)

Ο ελλειψοειδής πομπός Σωματιδίων δημιουργεί σωματίδια μέσα σε μια σφαίρα, η οποία μπορεί να κλιμακωθεί και να επεκταθεί.

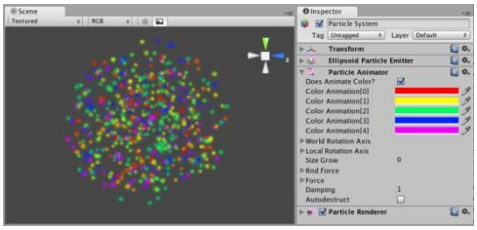


Mesh Particle Emitter (Legacy)

Ο πομπός πλεγμάτος Σωματιδίων εκπέμπει σωματίδια γύρω από ένα πλέγμα. Τα σωματίδια δημιουργούνται από την επιφάνεια του πλέγματος, που μπορεί να είναι απαραίτητο όταν

θέλουμε να κάνουμε τα σωματίδια να αλληλεπιδρούν με ένα πολύπλοκο τρόπο με τα αντικείμενα.

Particle Animator (Legacy)



Οι Animators σωματιδίων κινούν τα σωματίδια με την πάροδο του χρόνου, και μπορούμε να τους χρησιμοποιήσουμε για να εφαρμόσουμε τον αέρα, το χρώμα, την κατεύθυνση του συστήματος σωματιδίων.

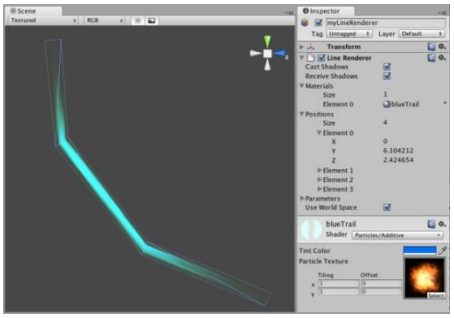
Particle Renderer (Legacy)

Οι Renderers σωματιδίων απαιτούνται για τα συστήματα σωματιδίων που εμφανίζονται στην οθόνη. Μπορούμε να καθορίσουμε το υλικό και τον shader που καθιστά και τις δύο πλευρές του υλικού.

World Particle Collider (Legacy)

Ο Collider Σωματιδίων χρησιμοποιείται για να συγκρουστούν τα σωματίδια έναντι άλλων Colliders στη σκηνή.

Line Renderer



Η Γραμμή Renderer παίρνει μια σειρά από δύο ή περισσότερα σημεία στο 3D χώρο και σχεδιάζει μια ευθεία γραμμή μεταξύ τους. Μια ενιαία γραμμή Renderer Component μπορεί να χρησιμοποιηθεί για να επιστήσει οτιδήποτε, όπως ακτίνες λέιζερ. Η γραμμή είναι πάντα συνεχής, εάν πρέπει να καταλήξει σε δύο ή περισσότερες εντελώς ξεχωριστές γραμμές χρησιμοποιούμε πολλά GameObjects, το καθένα με τη δική του γραμμή Renderer. Η Γραμμή Renderer δεν καθιστά ένα pixel σε λεπτές γραμμές. Καθιστά billboard γραμμές που έχουν πλάτος και υφή.

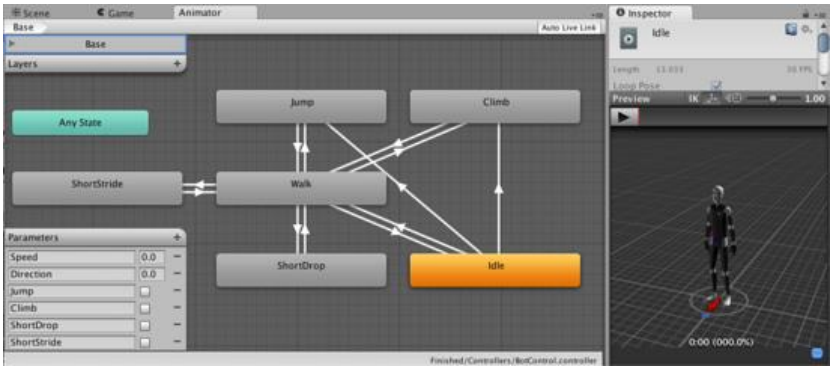
3.3.17 Mecanim Animation System

Η Unity έχει ένα πλούσιο και εκλεπτυσμένο σύστημα animation που ονομάζεται Mecanim. Το Mecanim προβλέπει τα εξής:

- Εύκολη ροή εργασίας και ρύθμιση των κινουμένων εικόνων (animations) σε ανθρωποειδούς χαρακτήρες.
- Animation retargeting: η δυνατότητα να εφαρμόζονται κινούμενα σχέδια από ένα μοντέλο σε ένα άλλο.
- Απλοποιημένη ροή εργασίας για την ευθυγράμμιση των animation κλιπς.
- Βολική προεπισκόπηση των animation κλιπ, μεταβάσεων και αλληλεπιδράσεων μεταξύ τους. Αυτό επιτρέπει στους animators να εργάζονται ανεξάρτητα από τους

προγραμματιστές, καθώς είναι διαθέσιμη η προεπισκόπηση των κινήσεων των πρωτότυπων χαρακτήρων τους, πριν συνδέσουμε τον κώδικα του παιχνιδιού σε αυτά.

- Διαχείριση πολύπλοκων αλληλεπιδράσεων μεταξύ κινουμένων σχεδίων με ένα οπτικό εργαλείο προγραμματισμού.
- Animating διάφορα μέρη του σώματος με διαφορετική λογική.



Το workflow στο Mecanim μπορεί να χωριστεί σε τρία βασικά στάδια :

- Προετοιμασία των περιουσιακών στοιχείων και των εισαγωγών . Αυτό γίνεται από τους καλλιτέχνες ή animators, με εργαλεία όπως το Max ή Maya. Αυτό το βήμα είναι ανεξάρτητο από τα Mecanim χαρακτηριστικά.
- Ρύθμιση χαρακτήρων για το Mecanim, η οποία μπορεί να γίνει με 2 τρόπους. Ο ένας τρόπος περιλαμβάνει την ανθρωποειδή εγκατάσταση χαρακτήρων. Το Mecanim έχει μια ειδική ροή εργασίας για τα ανθρωποειδή μοντέλα, με εκτεταμένη υποστήριξη GUI και επαναστόχευση. Η εγκατάσταση περιλαμβάνει τη δημιουργία και τη ρύθμιση ενός Avatar και μικροαλλαγές στους ορισμούς των μυών. Και ο δεύτερος τρόπος περιλαμβάνει τη Generic εγκατάσταση χαρακτήρων, η οποία χρησιμοποιείται για πλάσματα, κινούμενα σκηνικά, τετράποδα ζώα, κλπ. Η επαναστόχευση δεν είναι δυνατή εδώ.
- Φέρνοντας χαρακτήρες στη ζωή. Περιλαμβάνει τη δημιουργία animation clips, καθώς και τις αλληλεπιδράσεις μεταξύ τους, και την εγκατάσταση των State Machines και Blend Trees, εκθέτοντας Animation Παραμέτρους, και τον έλεγχο των κινουμένων σχεδίων από τον κώδικα.

3.3.18 Animation και Mecanim όροι

Όροι Animation Clip

- Animation Clip : Δεδομένα κίνησης που μπορούν να χρησιμοποιηθούν για χαρακτήρες κινουμένων σχεδίων ή απλά κινούμενα σχέδια. Είναι ένα από ενιαίο κομμάτι κίνησης, όπως "Idle", "Walk" ή "Run".
- Μάσκα σώματος: Η προδιαγραφή για την οποία τα μέρη του σώματος συμπεριλαμβάνονται ή όχι στο σκελετό. Χρησιμοποιείται σε στρώματα Animation και στον εισαγωγέα.

- Καμπύλες animation: Οι καμπύλες μπορούν να συνδεθούν με animation κλιπ και ελέγχονται από διάφορες παραμέτρους στο παιχνίδι.

Όροι Avatar

- Avatar: Μια διεπαφή για την επαναστόχευση ενός σκελετού σε έναν άλλον.
- Επαναστόχευση : Η εφαρμογή animations που δημιουργήθηκαν για ένα μοντέλο σε ένα άλλο.
- Rigging: Η διαδικασία οικοδόμησης μιας ιεραρχίας των αρθρώσεων των οστών στο πλέγμα του χαρακτήρα. Διενεργείται με ένα εξωτερικό εργαλείο, όπως το Max ή το Maya.
- Skinning: Η διαδικασία δέσμευσης αρθρώσεων των οστών στο πλέγμα του χαρακτήρα ή αλλιώς «δέρμα». Διενεργείται με ένα εξωτερικό εργαλείο, όπως το Max ή το Maya.
- Ορισμός των Μυών : Μια ενέργεια του Mecanim, η οποία μας επιτρέπει να έχουμε ένα πιο διαισθητικό έλεγχο στο σκελετό του χαρακτήρα. Όταν το Avatar είναι τοποθετημένο σωστά, το Mecanim εργάζεται στο χώρο των μυών, το οποίο είναι πιο αποτελεσματικό από το διάστημα των οστών.
- T-πόζα: Η στάση στην οποία ο χαρακτήρας έχει τα χέρια του στο πλάι σχηματίζοντας ένα «T». Η πόζα αυτή είναι απαιτούμενη για να γίνει ο χαρακτήρας Avatar.
- Bind-πόζα: Η στάση στην οποία ο χαρακτήρας σχεδιάστηκε.
- Ανθρώπινο πρότυπο: Μια προ-καθορισμένη χαρτογράφηση οστών. Χρησιμοποιείται για το ταίριασμα των οστών από τα αρχεία FBX στο Avatar.

Όροι animator και Animator Controller

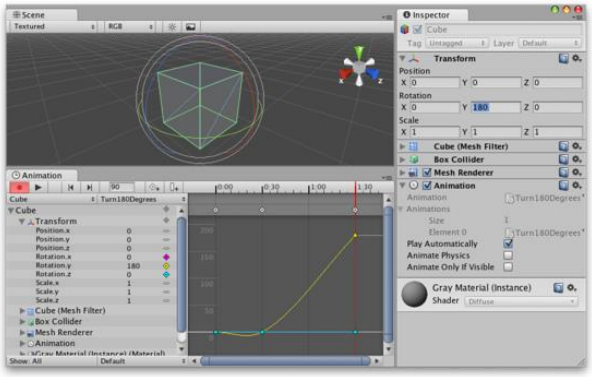
- Animator Component: Συστατικό που παρέχεται στο μοντέλο animations χρησιμοποιώντας το σύστημα animation Mecanim. Το component αναφέρεται σε ένα Animator Controller Asset που ελέγχει την κίνηση.
- Root κίνησης: Η κίνηση της ρίζας του χαρακτήρα, είτε ελέγχεται από το ίδιο το animation ή εξωτερικά .
- Animator Controller (Asset): Ελέγχει το animation μέσω των Layers Animation με Animation State Machines και Animation Blend Trees, που ελέγχονται από τους Animation Parameters. Ο ίδιος Animator Controller μπορεί να αναφερθεί από πολλαπλά μοντέλα με τα Animator components.
- Animator Controller (Window): Το παράθυρο όπου ο Animation Controller Asset οπτικοποιείται και επεξεργάζεται.
- Animation Layer: Ένα Layer Animation περιέχει Animation State Machine που ελέγχει τα animations ενός μοντέλου ή ένα μέρος τους . Μπορούμε να έχουμε σε ένα επίπεδο ολόκληρο το σώμα, για το περπάτημα ή το άλμα, και ένα υψηλότερο επίπεδο για κινήσεις του άνω μέρους του σώματος, όπως η ρίψη ενός αντικειμένου ή ο

πυροβολισμός. Τα υψηλότερα επίπεδα υπερισχύουν για τα μέρη του σώματος που ελέγχουν.

- Animation State Machine: Ένα γράφημα ελέγχου της αλληλεπίδρασης μεταξύ των Animation States. Κάθε κατάσταση αναφέρεται σε ένα Animation Blend Tree ή σε ένα μόνο Animation Clip.
- Animation Blend Tree: Χρησιμοποιείται για τη συνεχή ανάμειξη μεταξύ παρόμοιων Animation Clips βασισμένα σε παραμέτρους float Animation Parameters.
- Παράμετροι Animation: Χρησιμοποιείται για να επικοινωνούν μεταξύ τους το scripting και ο Animator Controller. Μερικοί παράμετροι μπορούν να ρυθμιστούν μέσω scripting και να χρησιμοποιηθούν από τον ελεγκτή, ενώ οι άλλοι παράμετροι βασίζονται στο Custom Curves στο Animation Clips, στις οποίες μπορούμε να επέμβουμε μέσω του scripting API .
- Inverse Κινηματική (IK): Η ικανότητα να ελέγχουμε τα μέρη του σώματος του χαρακτήρα όταν αλληλεπιδρά με διάφορα αντικείμενα στον κόσμο.

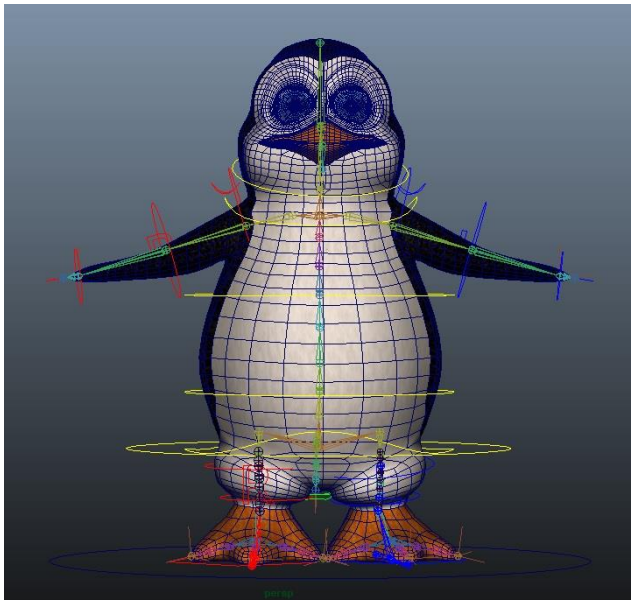
Animation View

Η προβολή Animation μας επιτρέπει να δημιουργήσουμε και να τροποποιήσουμε τα Animation Clips απευθείας μέσα στη Unity. Λειτουργεί ως ένα εναλλακτικό εργαλείο των εξωτερικών προγραμμάτων 3D animation. Για τη δημιουργία της κίνησης μέσω animation, μπορούμε να τροποποιήσουμε τις μεταβλητές των υλικών και των components και να προσθέσουμε στα Animation Clips των Animation Events λειτουργίες που καλούνται σε καθορισμένα σημεία κατά μήκος της λωρίδας χρόνου.



Προετοιμασία και εισαγωγή ενός Asset

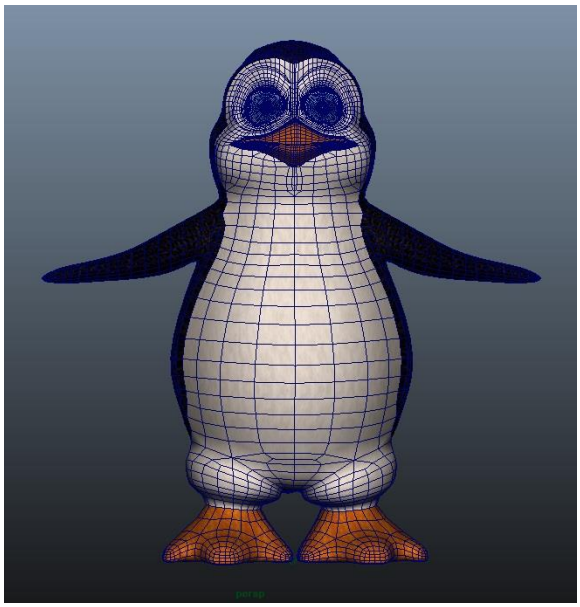
Υπάρχουν τρία βασικά βήματα για τη δημιουργία ενός animated ανθρωποειδούς χαρακτήρα από το μηδέν: μοντελοποίηση, rigging και skinning. Οι διαδικασίες αυτές γίνονται στο Maya. Για να επωφεληθούμε πλήρως από το ανθρωποειδές σύστημα animation και την επαναστόχευση του Mecanim, θα πρέπει να έχουμε ολοκληρώσει επιτυχώς και τα τρία αυτά βήματα.



Μοντελοποίηση

Μοντελοποίηση είναι η διαδικασία δημιουργίας ενός ανθρωποειδούς πλέγματος σε ένα 3D πρόγραμμα μοντελοποίησης, όπως 3DSMax, Maya, Blender, κλπ. Ένα μοντέλο ενός χαρακτήρα γενικά αποτελείται από πολύγωνα ενός 3D πρόγραμματος ή μετατρέπεται σε πολυγωνικό ή τριγωνικό πλέγμα από ένα πιο σύνθετο τύπο πλεγμάτων πριν από την εξαγωγή. Υπάρχουν μερικές οδηγίες με τις οποίες εξασφαλίζεται η καλή λειτουργία ενός μοντέλου με animation στη Unity.

- Είναι απαραίτητη μια λογική τοπολογία: Οι κορυφές και τα τρίγωνα του μοντέλου αλλοιώνονται καθώς αυτό κινείται. Μια κακή τοπολογία δεν θα επιτρέψει το μοντέλο να κινηθεί χωρίς αντιαισθητικές παραμόρφωσεις του πλέγματος.
- Πρέπει να ληφθεί υπόψιν η κλίμακα του πλέγματος. Η σύγκριση του μεγέθους των εισαγόμενων μοντέλων μπορεί να γίνει με έναν κύβο-GameObject της Unity, καθώς η κάθε πλευρά του έχει μήκος μια μόναδα Unity η οποία ισοδυναμεί με το ένα μέτρο στη φυσική της Unity. Οι μονάδες των 3D προγραμμάτων προσαρμόζονται στις ρυθμίσεις εξαγωγής, έτσι ώστε το μέγεθος του μοντέλου να είναι σε σωστή αναλογία με τον κύβο. Αν τα μοντέλα έχουν αφύσικα μεγέθη η φυσική της Unity δεν θα δουλεύει σωστά αλλά θα προσαρμόζεται στο μέγεθος του αντικειμένου. Τα μεγέθη των αντικειμένων πρέπει να είναι ομοιόμορφα μεταξύ τους σε μία πίστα για να έχουμε καλά αποτελέσματα.

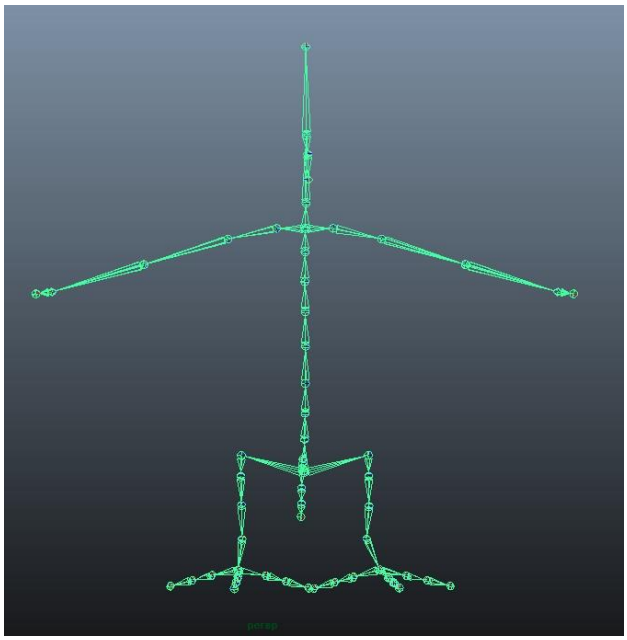


- Πρέπει το πλέγμα να τοποθετητεί έτσι ώστε τα πόδια του χαρακτήρα να στέκονται στην τοπική προέλευση ή το κεντρικό σημείο του μοντέλου (anchor point – pivot). Δεδομένου ότι ένας χαρακτήρας συνήθως περπατά όρθιος στο πάτωμα, είναι πολύ πιο εύκολο να τον χειριστούμε αν το κεντρικό του σημείο είναι ακριβώς πάνω στο πάτωμα .
- Το μοντέλο πρέπει να είναι σε T-πόζα. Έτσι έχουμε χώρο για να βελτιώσουμε λεπτομέρειες του πολυγώνου και είναι πιο εύκολη η διαδικασία rigging στο εσωτερικό του πλέγματος.
- Πρέπει να καθαρίσουμε το μοντέλο όπου είναι δυνατόν, απομακρύνοντας ότι δεν χρειάζεται, κρυμμένες επιφάνειες ή κορυφές, διευκολύνοντας έτσι τη διαδικασία skinning, ιδιαίτερα τις αυτοματοποιημένες διαδικασίες skinning.

Rigging

Το Rigging είναι η διαδικασία της δημιουργίας ενός σκελετού αρθρώσεων για να ελέγχει τις κινήσεις του μοντέλου. Η ιεραρχία των αρθρώσεων-joint ή του σκελετού ορίζει τα οστά στο εσωτερικό του πλέγματος και την κίνησή τους σε σχέση με τα άλλα, και πρέπει να δημιουργηθεί για να ελέγχει την κίνηση του χαρακτήρα. Η διαδικασία για τη δημιουργία της ιεραρχίας joint είναι γνωστή ως rigging.

Τα 3D προγράμματα παρέχουν μια σειρά από τρόπους για να δημιουργήσουν αρθρώσεις για ανθρωποειδή rig. Υπάρχουν έτοιμοι δίποδοι σκελετοί που αυξομειώνονται σε κλίμακα ώστε να χωρέσει στο κάθε πλέγμα, και διατίθονται εργαλεία για την ατομική δημιουργία των οστών και την ένωση μεταξύ τους μέσω parenting για να δημιουργήσουμε οποιαδήποτε δομή οστών επιθυμούμε. Για να εργαστούμε με το Mecanim οι γοφοί πρέπει να είναι η ρίζα της ιεραρχίας των οστών, και απαιτούνται τουλάχιστον δεκαπέντε οστά στο σκελετό.



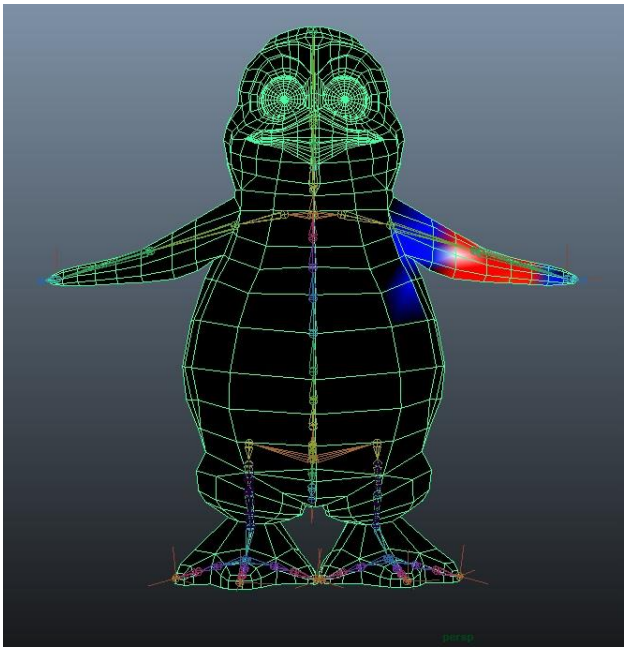
Η ιεραρχία αρθρώσεων/οστών θα πρέπει να ακολουθεί μια φυσική δομή για το χαρακτήρα που δημιουργούμε. Δεδομένου ότι τα χέρια και τα πόδια έρχονται σε ζεύγη, θα πρέπει να χρησιμοποιήσουμε μια συνεπή σύμβαση για την ονομασία τους (π.χ., "arm_L" για το αριστερό χέρι, "arm_R" για το δεξί χέρι, κλπ) ώστε να μπορέσουν να αναγνωριστούν από το Mecanim.

Skinning

Το Skinning είναι η διαδικασία της συνδέσης του πλέγματος στο σκελετό. Το πλέγμα ή το δέρμα θα πρέπει να συνδεθεί με την ιεραρχία των αρθρώσεων προκειμένου να καθορίσει ποια μέρη του πλέγματος του χαρακτήρα κινούνται όταν κινείται ένα συγκεκριμένο joint.

Το Skinning δεσμεύει τις κορυφές του πλέγματος με τα οστά, είτε άμεσα (με άκαμπτη δεσμεύση-rigid bind) ή με μικτή επίδραση σε μια σειρά από οστά (ομαλή δέσμευση- soft bind). Διαφορετικά προγράμματα λογισμικού χρησιμοποιούν διαφορετικές μεθόδους, π.χ. αναθέτουν επιμέρους κορυφές και ζωγραφίζουν τα βάρη επιρροής ανά οστό πάνω στο πλέγμα. Η αρχική ρύθμιση είναι συνήθως αυτοματοποιημένη, δηλαδή η εύρεση της πλησιέστερης επιρροής ή τη χρήση «Heatmaps». Το Skinning πρέπει να ελεγχθεί αρκετές φορές μέσω δοκιμών κίνησης προκειμένου να εξασφαλιστούν ικανοποιητικά αποτελέσματα όσον αφορά την παραμόρφωση του δέρματος. Μερικές γενικές οδηγίες για τη διαδικασία αυτή περιλαμβάνουν :

- Αρχικά τη χρήση μιας αυτοματοποιημένης διαδικασίας για τη δημιουργία μερικού skinning, την οποία μπορούμε να δούμε από σχετικά tutorials για 3DMax, Maya, κλπ.
- Τη δημιουργία ενός απλού animation για το rig ή την εισαγωγή ορισμένων δεδομένων κίνησης η οποία θα λειτουργήσει ως δοκιμή για το skinning. Αυτό θα μας δώσει μια αξιολόγηση κατά πόσον ή όχι φαίνεται καλό το skinning σε κίνηση.
- Τη σταδιακή επεξεργασία και βελτίωση του skinning.
- Τη διατήρηση το πολύ τεσσάρων επιρροών σε ένα joint όταν χρησιμοποιούμε την ομαλή δεσμεύση, δεδομένου ότι αυτός είναι ο μέγιστος αριθμός που χειρίζεται η Unity. Εάν υπάρχουν περισσότερες από τέσσερις επιδράσεις που επηρεάζουν ένα μέρος του πλέγματος τότε τουλάχιστον κάποιες πληροφορίες θα χαθούν όταν παίζει το animation στη Unit



Η Unity εισάγει έναν αριθμό διαφορετικών generic και native 3D μορφών αρχείων, όπως native αρχεία Maya (.mb or .ma) και Cinema 4D (.c4d), και generic FBX αρχεία τα οποία μπορούν να εξαχθούν από τα περισσότερα προγράμματα animation. Η μορφή που προτιμήσαμε για την εξαγωγή και την επαλήθευση του μοντέλου μας είναι του FBX 2013, δεδομένου ότι θα μας επιτρέψει την εξαγωγή του πλέγματος με την ιεραρχία του σκελετού, των normals, των υφών και των animations, την επανεισαγωγή του αρχείου στο 3D προγράμμα για να ελέγξουμε το κινούμενο μοντέλο, και την εξαγωγή κινουμένων σχεδίων χωρίς πλέγματα.

Διάσπαση Animations

Ένας κινούμενος χαρακτήρας έχει συνήθως μια σειρά από διαφορετικές κινήσεις που ενεργοποιούνται στο παιχνίδι σε διαφορετικές περιστάσεις. Οι κινήσεις αυτές ονομάζονται Animation Clips. Για παράδειγμα μπορεί να έχουμε ξεχωριστά κλιπ κινουμένων σχεδίων για το περπάτημα, το τρέξιμο, το άλμα, τη ρίψη, το θάνατο, κλπ. Ανάλογα με τον τρόπο που δημιουργήθηκαν τα κινουμένα σχεδία του μοντέλου, αυτές οι διαφορετικές κινήσεις μπορούν να εισαχθούν ως ξεχωριστά κλιπ κινουμένων σχεδίων ή ως ένα ενιαίο κλιπ όπου η κάθε κίνηση είναι απλά η συνέχεια του προηγούμενου κλιπ. Σε περιπτώσεις όπου υπάρχει μόνο ένα κλιπ το κλιπ χωρίζεται σε component animation clips εντός της Unity.

Non-humanoid animations

Υπάρχουν δύο επιλογές για μη-ανθρωποειδή animation: Generic και Legacy. Τα Generic animations εισάγονται με τη χρήση του συστήματος Mecanim αλλά δεν επωφεληθούνται από τις επιπλέον λειτουργίες που είναι διαθέσιμες για τα ανθρωποειδή κινούμενα σχέδια. Τα Legacy Animations χρησιμοποιούν το σύστημα του animation που παρέχεται από τη Unity πριν το Mecanim.

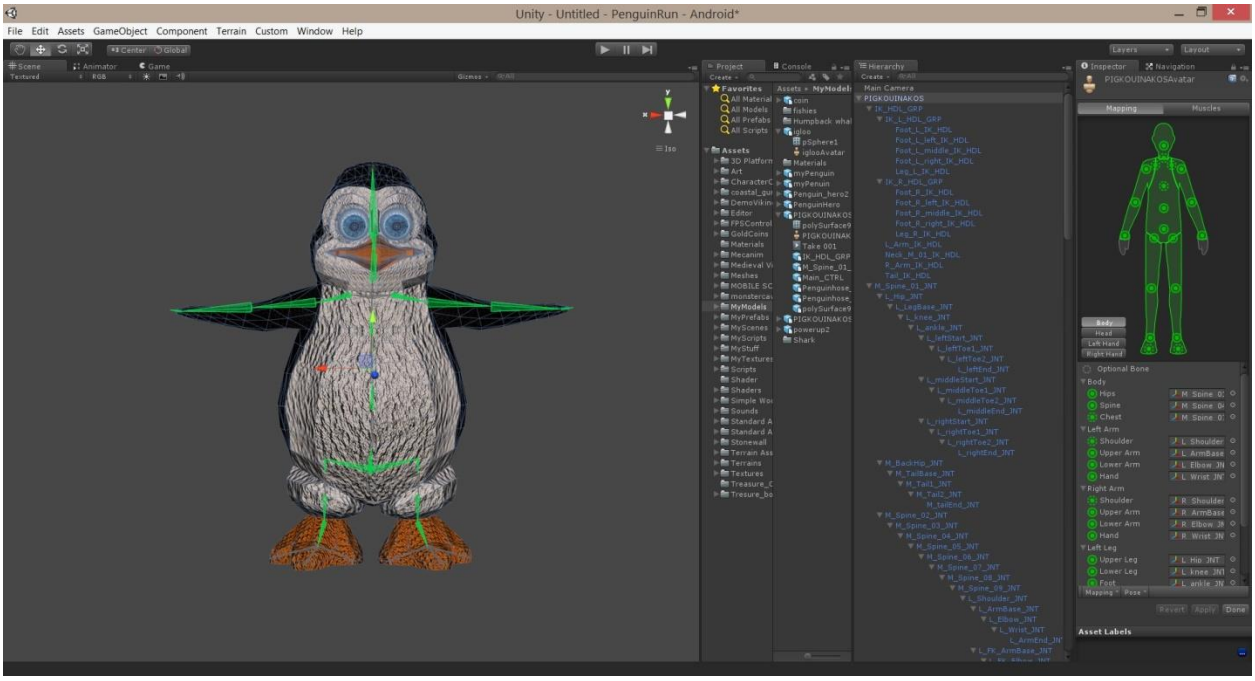
Humanoid Animations

Λόγω της ομοιότητας στη δομή των οστών, είναι δυνατό να χαρτογραφηθούν κινούμενες εικόνες από ένα ανθρωποειδή σκελετό σε έναν άλλον, επιτρέποντας την επαναστόχευση και την αντίστροφη κινηματική.

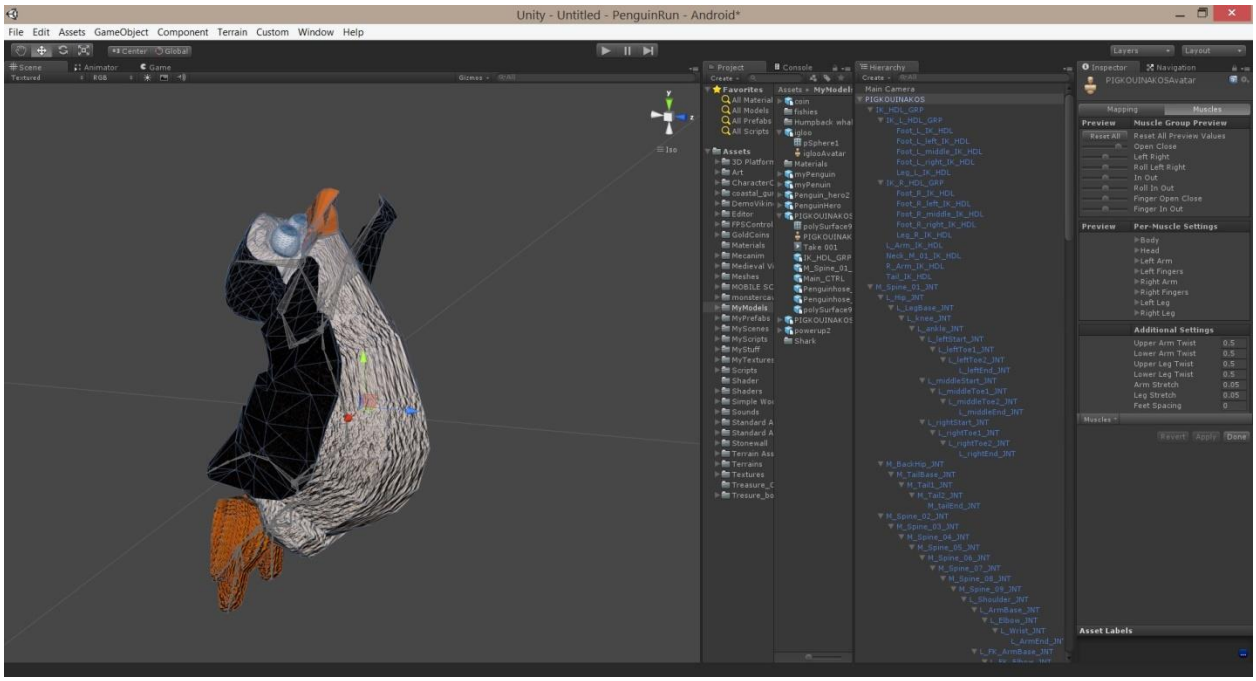
Με σπάνιες εξαιρέσεις, τα ανθρωποειδή μοντέλα που εκπροσωπούν τα σημαντικότερα τμήματα των αρθρώσεων του σώματος, του κεφαλιού και των άκρων, αναμένεται να έχουν την ίδια βασική δομή, έτσι το σύστημα Mecanim απλοποιεί το rigging και τον ελέγχο των κινούμενων εικόνων. Ένα βασικό βήμα για τη δημιουργία ενός animation είναι η δημιουργία μίας χαρτογράφησης μεταξύ της απλουστευμένης ανθρωποειδούς δομής των οστών κατανοητή από το Mecanim, και των πραγματικών οστών που υπάρχουν στο σκελετό. Στην ορολογία του Mecanim, αυτή η χαρτογράφηση ονομάζεται Avatar.

Το Mecanim ταιριάζει την υφιστάμενη δομή των οστών με τη δομή των οστών του Avatar. Σε πολλές περιπτώσεις, αυτό μπορεί να γίνει αυτόματα αναλύοντας τις συνδέσεις μεταξύ των οστών στο rig. Αν τα ταιριάξει επιτυγχώς εμφανίζεται η ένδειξη “Configure”, διαφορετικά θα πρέπει να τα ταιριάξουμε χειροκίνητα.

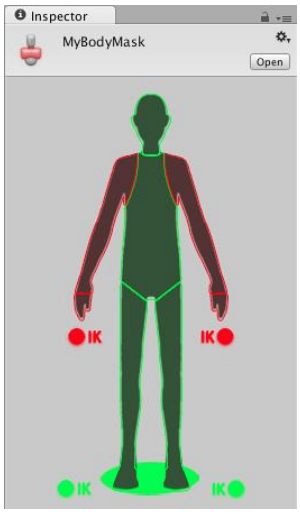
Η δομή των οστών του χαρακτήρα πρέπει να ταιριάζει με την προκαθορισμένη δομή των οστών του Mecanim και το μοντέλο να είναι σε T-στάση. Η χαρτογράφηση των οστών φαίνεται σε έναν Avatar Configuration inspector. Ο επιθεωρητής δείχνει ποια από τα οστά χρειάζονται και ποια είναι προαιρετικά. Ο σκελετός πρέπει να έχει τα απαιτούμενα οστά στη θέση τους. Για να πραγματοποιηθεί σωστά η αυτόματη έρευση των οστών από τον Avatar, τα ονόματα των οστών πρέπει να αντανακλούν τα μέρη του σώματος που εκπροσωπούν (ονόματα όπως "LeftArm", "RightForearm").



Το Mecanim επιτρέπει τον έλεγχο του εύρους της κίνησης των διαφόρων οστών, χρησιμοποιώντας τους μύες. Μόλις ρυθμιστεί σωστά ο Avatar, το Mecanim θα "καταλάβει" τη δομή των οστών και θα αρχίσει να εργάζεται στους μύες του Avatar Inspector, καθώς εκεί είναι πολύ εύκολος ο χειρισμός της κίνησης του χαρακτήρα και επίσης μπορούμε να ελέγξουμε αν η παραμορφώση γίνεται με πειστικό τρόπο.



Avatar Body Mask



Συγκεκριμένα μέρη του σώματος μπορούν να ενεργοποιηθούν ή να απενεργοποιηθούν σε μια κινούμενη εικόνα, χρησιμοποιώντας τη λεγόμενη μάσκα σώματος επιλεκτικά. Οι μάσκες σώματος χρησιμοποιούνται στην καρτέλα Animation του πλέγματος και στα Επίπεδα του Animation. Επιτρέπουν την προσαρμογή ενός κινουμένου σχεδίου ώστε να ταιριάζει περισσότερο με τις ειδικές απαιτήσεις του χαρακτήρα μας. Για παράδειγμα, μπορεί να έχουμε ένα animation περπατήματος που περιλαμβάνει την κίνηση των ποδιών και των χεριών, αλλά αν ο χαρακτήρας μεταφέρει ένα μεγάλο αντικείμενο με τα δύο χέρια δεν θέλουμε τα χέρια του να ταλαντεύονται στο πλάι καθώς περπατά, αλλά θα μπορούσαμε να χρησιμοποιήσουμε το animation περπατήματος σβήνοντας τις κινήσεις των χεριών στη μάσκα σώματος .

Τα μέρη του σώματος που περιλαμβάνονται είναι το κεφάλι, το χέρια (αριστερό και δεξί), τις παλάμες (αριστερή και δεξιά) τα πόδια (αριστερό και δεξί) και τη ρίζα Root που συμβολίζεται με την σκιά κάτω από τα πόδια. Στη μάσκα σώματος μπορούμε να αλλάξουμε την αντίστροφη κινηματική (IK) για τα χέρια και τα πόδια, η οποία θα καθορίσει κατά πόσον οι IK καμπύλες θα πρέπει να περιλαμβάνονται σε animation blending.

Ένα από τα πιο ισχυρά χαρακτηριστικά του Mecanim είναι η αναδιάταξη των ανθρωποειδών animations. Αυτό σημαίνει ότι μπορούμε να εφαρμόσουμε το ίδιο σύνολο των animations σε διάφορα μοντέλα χαρακτήρων. Η επαναστόχευση είναι δυνατή μόνο για τα μοντέλα ανθρωποειδών όπου ο Avatar έχει ρυθμιστεί, διότι αυτό μας δίνει μια αντιστοιχία μεταξύ της δομής των οστών των μοντέλων.

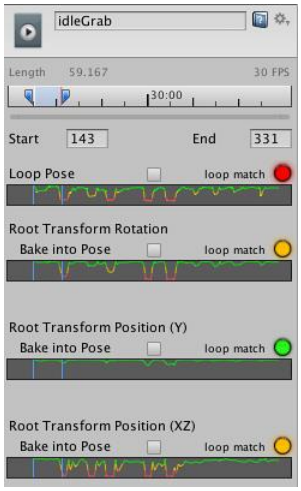
Inverse Kinematics

Τα περισσότερα animations παράγονται από την περιστροφή των γωνιών των αρθρώσεων σε ένα σκελετό σε προκαθορισμένες τιμές. Η θέση ενός παιδιού μιας άρθρωσης αλλάζει ανάλογα με την περιστροφή του γονέα του και έτσι το τελικό σημείο μιας αλυσίδας αρθρώσεων μπορεί να προσδιορισθεί από τις γωνίες και τις σχετικές θέσεις των μεμονωμένων αρθρώσεων που περιέχει. Αυτή η μέθοδος παρουσιάζει ένα σκελετό που είναι γνωστός ως forward kinematics.

Η τοποθέτηση αρθρώσεων γίνεται και με αντίστροφή διαδικασία, δηλαδή δίνεται μια επιλεγμένη θέση στο χώρο και εργαζόμαστε προς τα πίσω για να βρούμε έναν έγκυρο τρόπο χάραξης των αρθρώσεων έτσι ώστε το τελικό σημείο να τοποθετηθεί σε αυτή τη θέση. Αυτό χρειάζεται όταν θέλουμε ο χαρακτήρας μας να αγγίξει ένα αντικείμενο σε ένα σημείο που επιλέγεται από το χρήστη ή να τοποθετήσει τα πόδια του με πειστικό τρόπο σε μια ανώμαλη

επιφάνεια. Η προσέγγιση αυτή είναι γνωστή ως αντίστροφη κινηματική (IK) και υποστηρίζεται στο Mecanim για κάθε ανθρωποειδή χαρακτήρα με σωστά ρυθμισμένο Avatar.

Looping animation clips

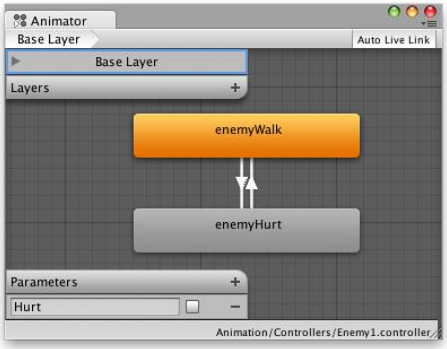


Μια κοινή λειτουργία των κινουμένων σχεδίων είναι η σωστή λειτουργία του βρόγχου επανάληψης. Είναι σημαντικό το κλιπ κινουμένων σχεδίων να αρχίζει και να τελειώνει σε μια παρόμοια στάση (π.χ. στο περπάτημα το αριστερό πόδι στο έδαφος), ώστε να εξασφαλίζεται ότι δεν υπάρχουν παράξενες σπασμωδικές κινήσεις. Το Mecanim παρέχει πρακτικά εργαλεία για το σκοπό αυτό. Το Animation κλιπ μπορεί επαναληφθεί με βάση την πόζα, την περιστροφή και τη θέση.

Επιλέγοντας την αρχή και το τέλος του animation clip βλέπουμε τις καμπύλες καταλληλότητας του βρόγχου επανάληψης για όλες τις παραμέτρους βάσει των οποίων είναι δυνατή η επανάληψη. Ο δείκτης loop match ανάλογα με το χρώμα που παίρνει δείχνει πόσο κατάλληλο είναι το looping για τις επιλεγμένες περιοχές, όπου πράσινο είναι το καταλληλότερο looping.

Animator και Animator Controller

Το Avatar ορίζει τη δομή του σκελετού ενός αντικειμένου, αλλά απαιτείται ένας Animator ελεγκτής για να εφαρμόσει τα animations στο σκελετό. Ο Animator Controller δημιουργείται μέσα στη Unity και μας επιτρέπει να διατηρούμε μια σειρά από κινούμενα σχέδια για ένα χαρακτήρα και να τα εναλλάσσουμε μεταξύ τους όταν συμβαίνουν ορισμένες συνθήκες του παιχνιδιού. Για παράδειγμα μπορούμε να αλλάξουμε από ένα animation περπατήματος σε ένα άλμα όταν πιέσουμε το spacebar. Ο ελεγκτής διαχειρίζεται τις μεταβάσεις μεταξύ των animations χρησιμοποιώντας τη State Machine, ένα απλό πρόγραμμα γραμμένο σε μια οπτική γλώσσα προγραμματισμού εντός της Unity.

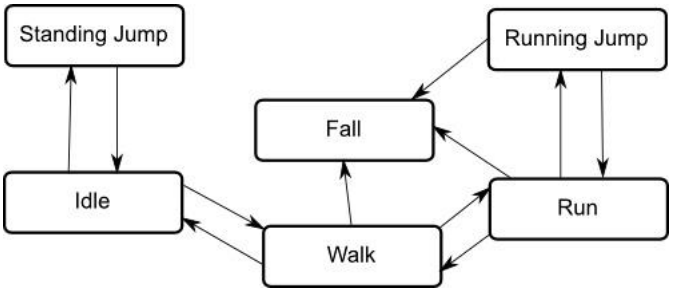


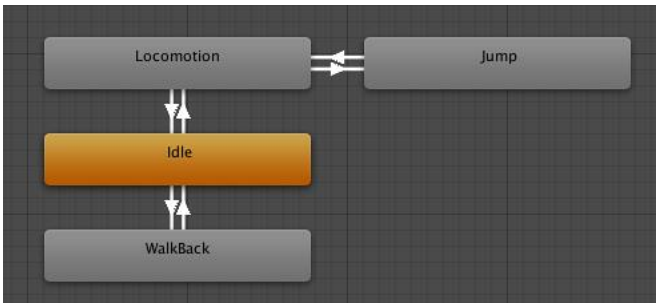
Animation State Machines

Ένας χαρακτήρας μπορεί να έχει πολλές διαφορετικές κινήσεις που αντιστοιχούν σε διαφορετικές ενέργειες που μπορεί να εκτελέσει στο παιχνίδι, όπως να αναπνεύει ή να ταλαντεύονται ελαφρά ενώ είναι σε αδράνεια, να περπατάει, να σηκώνει τα χέρια του σε κατάσταση πανικού καθώς πέφτει από μια πλατφόρμα. Ο έλεγχος, όταν αναπαράγονται αυτές οι κινήσεις, είναι μια αρκετά πολύπλοκη εργασία scripting. Το Mecanim χρησιμοποιεί μηχανές καταστάσεων για την απλοποίηση του ελέγχου και της αλληλουχίας των animations ενός χαρακτήρα.

Ένας χαρακτήρας έχει ένα συγκεκριμένο είδος δράσης σε κάθε δεδομένη στιγμή. Οι διαθέσιμες ενέργειες θα εξαρτηθούν από το είδος του παιχνιδιού, αλλά οι τυπικές ενέργειες περιλαμβάνουν την αδράνεια, το περπάτημα, το τρέξιμο, τα άλματα, κλπ. Οι δράσεις αυτές αναφέρονται ως καταστάσεις, και ο χαρακτήρας βρίσκεται σε μια "κατάσταση " περπατήματος ή οτιδήποτε άλλο. Υπάρχουν περιορισμοί σχετικά με την επόμενη κατάσταση που μπορεί να πάει ο χαρακτήρας και δεν είναι σε θέση να στραφεί αμέσως από μια κατάσταση σε οποιαδήποτε άλλη. Για παράδειγμα, ο χαρακτήρας μπορεί να πηδήξει όταν τρέχει μόνο όταν βρίσκεται ήδη σε εξέλιξη και όχι όταν είναι σε ακινησία, γι 'αυτό δεν μπορεί να μεταβεί κατ' ευθείαν από την κατάσταση αναμονής στην κατάσταση άλματος. Οι επιλογές για την επόμενη κατάσταση στην οποία ένας χαρακτήρας μπορεί να πάει από την τρέχουσα κατάσταση αναφέρονται ως μεταβάσεις. Για να θυμόμαστε την τωρινή κατάσταση της στατικής μηχανής συγκρατούμε το σύνολο των καταστάσεων, το σύνολο των μεταβάσεων και τις μεταβλητές.

Οι καταστάσεις και οι μεταβάσεις μιας στατικής μηχανής μπορούν να αναπαρασταθούν χρησιμοποιώντας ένα γραφικό διάγραμμα, όπου οι κόμβοι αναπαριστούν στις καταστάσεις και τα τόξα (βέλη μεταξύ των κόμβων) αντιπροσωπεύουν τις μεταβάσεις. Η τρέχουσα κατάσταση μπορεί να θεωρηθεί ως ένας δείκτης που τοποθετείται σε έναν από τους κόμβους και μπορεί στη συνέχεια να μεταβεί μόνο σε άλλο κόμβο κατά μήκος ενός από τα βέλη.



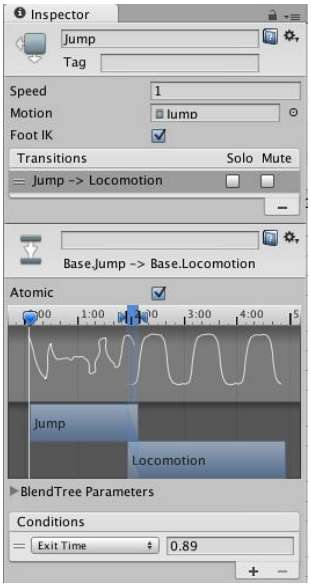


Με τις στατικές μηχανές τα κινούμενα σχέδια μπορούν να σχεδιάζονται και να ενημερώνονται αρκετά εύκολα με σχετικά μικρή κωδικοποίηση. Κάθε κατάσταση έχει μια κίνηση που συνδέεται με αυτό που θα συμβεί κάθε φορά που το μηχάνημα βρίσκεται σε αυτή την κατάσταση. Αυτό επιτρέπει σε έναν animator ή σχεδιαστή να καθορίσει τις πιθανές ακολουθίες ενεργειών του χαρακτήρα και των κινουμένων σχεδίων, χωρίς να ανησυχεί για το πώς θα δουλεύει ο κώδικας. Τα Animation State Machines του Mecanim παρέχουν έναν τρόπο για να εποπτεύουν όλα τα κλιπ κινουμένων σχεδίων που σχετίζονται με ένα συγκεκριμένο χαρακτήρα και επιτρέπουν στις διάφορες εκδηλώσεις του παιχνιδιού να προκαλέσουν διαφορετικά animations.

Animation States

Οι καταστάσεις των animation (Animation States) είναι τα βασικά δομικά στοιχεία μιας μηχανής καταστάσεων Animation. Κάθε κατάσταση περιέχει μια μεμονωμένη σειρά κινουμένων σχεδίων (ή

blend tree), η οποία θα παίξει ενώ ο χαρακτήρας είναι σε αυτή την κατάσταση. Όταν ένα γεγονός στο παιχνίδι ενεργοποιεί μια μεταβατική κατάσταση, ο χαρακτήρας θα πρέπει να αφεθεί σε μια νέα κατάσταση και σε ένα νέο animation.



Animation Transitions

Οι μεταβάσεις των Animations ορίζουν τι συμβαίνει όταν αλλάζουμε από τη μια Animation κατάσταση στην άλλη. Μπορεί να υπάρχει μόνο μία ενεργή μετάβαση σε κάθε δεδομένη στιγμή. Μια κατάσταση αποτελείται από μια παράμετρο του γεγονότος ή έναν χρόνο εξόδου, προσδιορίζοντας έναν αριθμό που αντιπροσωπεύει το κανονικοποιημένο χρόνο της κατάσταση της πηγής (π.χ. 0,95 σημαίνει ότι η μετάβαση θα πραγματοποιηθεί όταν έχουμε παίξει το 95% του κλιπ της πηγής), έναν όρο αν χρειάζεται (για παράδειγμα Λιγότερο/Περισσότερο για float μεταβλητές), και μια τιμή της παραμέτρου (αν χρειάζεται).

Animation Parameters

Οι παράμετροι του Animation είναι οι μεταβλητές που έχουν οριστεί στο πλαίσιο του συστήματος κινουμένων σχεδίων, αλλά μπορούν επίσης να προσεγγιστούν καθορισμένες τιμές από τον κώδικα. Για παράδειγμα, η τιμή μιας παραμέτρου μπορεί να ενημερώνεται από μια καμπύλη animation και στη συνέχεια να προσεγγιστεί από ένα κώδικα, όπως ο τόνος ενός ήχου μπορεί να μεταβάλλεται σαν να ήταν ένα κομμάτι του animation. Ομοίως ένας κώδικας ορίζει τις τιμές των παραμέτρων που πρέπει να διαβαστούν από το Mecanim, όπως μπορεί να ρυθμίσει μια παράμετρο για τον έλεγχο ένός Blend Tree. Η μεταβλητή μιας παραμέτρου μπορεί να είναι ένας Vector για ένα σημείο στο διάστημα, ένας ακέραιος αριθμός Int, ένας αριθμός Float με ένα κλασματικό μέρος, ή μια Bool με τιμή true ή false.

Blend Trees

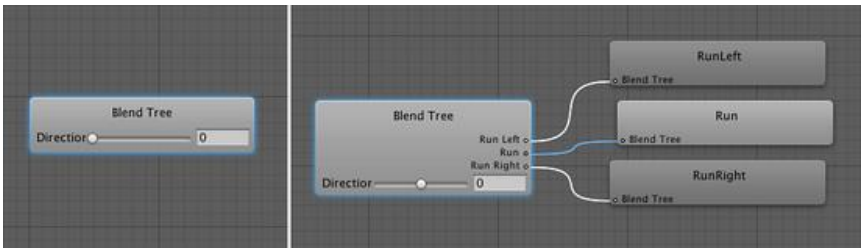
Σε ένα παιχνίδι συνήθως συνδυάζονται μεταξύ τους δύο ή περισσότερες παρόμοιες κινήσεις, όπως η ανάμειξη του περπατήματος και του τρεξίματος των κινουμένων σχεδίων ανάλογα με την ταχύτητα του χαρακτήρα, ή την κλίση του ήρωα προς τα αριστερά ή προς τα δεξιά καθώς γυρίζει ενώ τρέχει. Οι μεταβάσεις και τα Blend δέντρα χρησιμοποιούνται για τη δημιουργία ομαλής κίνησης αλλά για διαφορετικά είδη καταστάσεων.

Οι μεταβάσεις χρησιμοποιούνται για την ομαλή μετάβαση από τη μια κατάσταση του Animation σε μια άλλη σε ένα συγκεκριμένο χρονικό διάστημα. Οι μεταβάσεις ορίζονται ως μέρος μιας μηχανής καταστάσεων του Animation. Η μετάβαση από τη μία κίνηση σε μια εντελώς διαφορετική κίνηση είναι συνήθως καλή, αν η μετάβαση είναι γρήγορη.

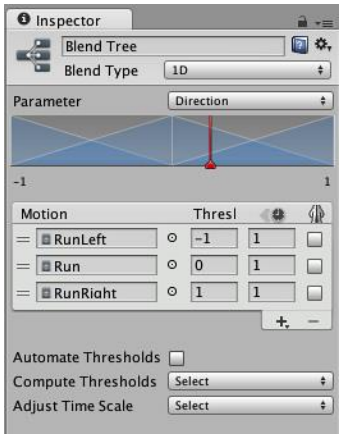
Τα Blend δέντρα χρησιμοποιούνται για να επιτρέπουν την ομαλή ανάμιξη πολλαπλών κινήσεων με ενσωματωμένα τμήματά τους σε διάφορους βαθμούς. Το πόσο συμβάλλει η κάθε

μία από τις κινήσεις στο τελικό αποτέλεσμα ελέγχεται με μια ανάμειξη παραμέτρων, η οποία είναι μόλις μία από τις πολλές παραμέτρους κίνησης που σχετίζονται με τον Animator ελεγκτή. Για για να έχει νόημα η μικτή κίνηση, οι κινήσεις που είναι αναμεμειγμένες πρέπει να είναι παρόμοιας φύσης και χρονοδιαγράμματος. Τα Blend δέντρα είναι ένας ειδικός τύπος μιας κατάστασης σε μια μηχανή καταστάσεων του Animation.

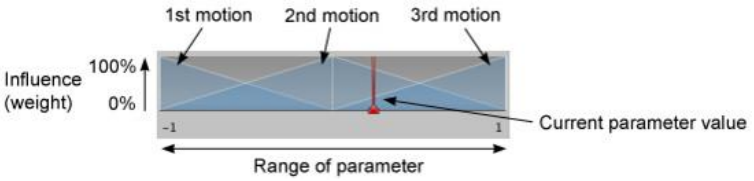
Για να λειτουργεί καλά η ανάμειξη των animations, οι μεταβολές των κλιπς πρέπει να γίνονται στα ίδια σημεία σε κανονικοποιημένο χρόνο. Για παράδειγμα, το περπάτημα και το τρέξιμο μπορούν να ευθυγραμμιστούν έτσι ώστε οι στιγμές της επαφής του ποδιού στο πάτωμα να λαμβάνουν χώρα στα ίδια σημεία σε κανονικοποιημένο χρόνο (π.χ. το αριστερό πόδι να ακουμπάει στο 0,0 και το δεξί πόδι στο 0,5). Δεδομένου ότι χρησιμοποιείται κανονικοποιημένος χρόνος, δεν έχει σημασία αν τα κλιπς έχουν διαφορετικό μήκος.



Τα είδη της ανάμειξης των κινουμένων σχεδίων χωρίζονται σε δύο κατηγορίες, σε 1D και 2D. Το 1D Blending συνδυάζει τις κινήσεις του παιδιού, σύμφωνα με μία μόνο παράμετρο, ενώ το 2D Blending συνδυάζει τις κινήσεις του παιδιού, σύμφωνα με δύο παραμέτρους. Μετά τη ρύθμιση του Blend Τύπου, πρέπει να επιλέξουμε την παράμετρο Animation που θα ελέγξει αυτό το Blend δέντρο. Σε αυτό το παράδειγμα, η παράμετρος είναι κατεύθυνση που κυμαίνεται μεταξύ -1.0 (αριστερά) και 1.0 (δεξιά), με 0,0 δηλώνει μια ευθεία κίνηση, χωρίς να κλίνει. Μπορούμε να προσθέσουμε ξεχωριστά κινούμενες εικόνες.



Το διάγραμμα στην κορυφή του Inspector δείχνει την επίδραση καθεμιάς από τις κινήσεις των παιδιών, καθώς η παράμετρος ποικίλει μεταξύ της ελάχιστης και της μέγιστης τιμής της. Κάθε κίνηση εμφανίζεται ως μια μικρή μπλε πυραμίδα (η πρώτη και η τελευταία εμφανίζονται μόνο κατά το ήμισυ), η κορυφή της κάθε πυραμίδας καθορίζει την τιμή της παραμέτρου. Όπου η κίνηση έχει πλήρη επίδραση το βάρος του animation είναι 1 και οι άλλες κινήσεις έχουν βάρος 0. Αυτό ονομάζεται επίσης το threshold της κίνησης.

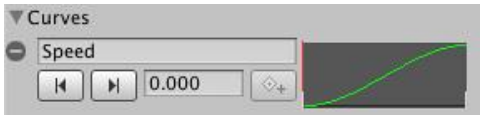


Animation Curves στο Mecanim

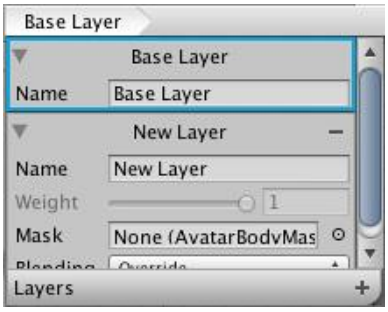
Οι Animation καμπύλες μπορούν να συνδεθούν με animation κλιπ στις ρυθμίσεις εισαγωγής. Ο x-άξονας της καμπύλης παριστάνει τον κανονικοποιημένο χρόνο και κυμαίνεται

πάντα μεταξύ 0,0 και 1,0 , που αντιστοιχεί στην αρχή και το τέλος του κλιπ κινουμένων σχεδίων, αντιστοίχως και ανεξάρτητα από τη διάρκεια του.

Μπορούμε να προσθέσουμε κλειδιά στην καμπύλη, τα οποία είναι τα σημεία κατά μήκος του χρόνου της καμπύλης όπου έχει οριστεί μια τιμή από τον animator. Τα κλειδιά είναι πολύ χρήσιμα για τη σήμανση σημαντικών σημείων κατά μήκος της λωρίδας χρόνου του animation. Για παράδειγμα, σε ένα animation περπατήματος μπορούμε να βάλουμε ως κλειδιά τα σημεία όπου είναι στο έδαφος το αριστερό πόδι, και τα δύο πόδια, το δεξί πόδι, κλπ.



Animation Layers



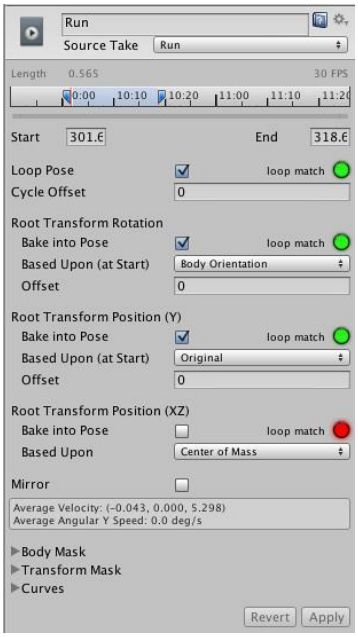
Τα Animation Layers χρησιμοποιούνται για τη διαχείριση σύνθετων στατικών μηχανών για διάφορα μέρη του σώματος, πχ ένα στρώμα με το κάτω-σώμα για περπάτημα και άλμα, και ένα στρώμα άνω μέρους του σώματος για ρίψη αντικειμένων/πυροβολισμού.

Target Matching

Συχνά σε παιχνίδια προκύπτει μια κατάσταση όπου ένας χαρακτήρας πρέπει να κινηθεί με τέτοιο τρόπο που το ένα χέρι ή πόδι προσγειώνεται σε ορισμένο μέρος σε ένα ορισμένο χρονικό διάστημα. Για το χειρισμό μιας τέτοιας κατάστασης μπορούμε να χρησιμοποιήσουμε τη λειτουργία Animator.MatchTarget.

Root Motion

Το Body Transform είναι το κέντρο μάζας του χαρακτήρα, χρησιμοποιείται στη μηχανή επαναστόχευσης του Mecanim και παρέχει το πιο σταθερό μοντέλο μετατόπισης. Ο προσανατολισμός του σώματος είναι ο μέσος όρος του κατώτερου και ανώτερου προσανατολισμού του σώματος σε σχέση με την T-Pose του Avatar. Τα Body Μετασχηματισμού και προσανατολισμού αποθηκεύονται στο Animation Clip χρησιμοποιώντας τους ορισμούς των μυών του Avatar. Είναι οι μόνες καμπύλες που αποθηκεύονται στο Animation Clip, ενώ όλες οι καμπύλες μυών και οι στόχοι IK (χέρια και πόδια) είναι αποθηκευμένες στο body transform.



Η Root Transform είναι μια προβολή στο επίπεδο Y του Body Transform και υπολογίζεται κατά το χρόνο εκτέλεσης. Σε κάθε πλαίσιο υπολογίζεται μια αλλαγή στη ρίζα Transform και εφαρμόζεται στο αντικείμενο για να κινηθεί το GameObject.

Οι ρυθμίσεις του Animation Clip Editor, Root Μετασχηματισμός Περιστροφής (Root Transform Rotation), (Root Transform Position)

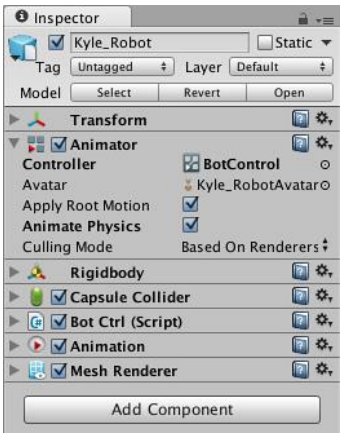
θέσης (Y) και θέσης (XZ), μας επιτρέπουν να ελέγχουμε το Root Transform προβολής από το Body Transform. Ανάλογα με τις ρυθμίσεις αυτές ορισμένα μέρη του σώματος μετασχηματισμού μπορούν να μεταφέρουν Root Transform.

- Bake into Pose: Ο προσανατολισμός θα παραμείνει στο σώμα μετασχηματισμού (ή Pose). Ο προσανατολισμός Root θα είναι συνεχής και ο Προσανατολισμός θα είναι η ταυτότητά του. Αυτό σημαίνει ότι το αντικείμενο δεν θα περιστραφεί καθόλου από αυτό το AnimationClip. Μόνο τα AnimationClips που έχουν παρόμοια αρχή και τέλος του Root Προσανατολισμού χρησιμοποιούν αυτή την επιλογή. Ένα πράσινο φως στο UI μας ενημερώνει ότι το AnimationClip είναι κατάλληλο, που σημαίνει ότι περπατάει ή τρέχει σε ευθεία γραμμή.
- Based Upon: Αυτό μας επιτρέπει να ορίσουμε τον προσανατολισμό του κλιπ. Χρησιμοποιώντας το Προσανατολισμό του σώματος, το κλιπ προσανατολίζεται να ακολουθεί τον μπροστινό φορέα του σώματος. Αυτή η προεπιλεγμένη ρύθμιση λειτουργεί καλά για τα περισσότερα δεδομένα Motion Capture (Mocap), όπως περίπατος, τρέξιμο, και άλματα, αλλά θα αποτύχει με κίνηση κάθετη προς τον μπροστινό φορέα του σώματος . Σε αυτές τις περιπτώσεις , μπορούμε να ρυθμίσουμε με το χέρι τον προσανατολισμό χρησιμοποιώντας την τιμή Offset . Έχουμε την επίλογή Original που προσθέτει αυτόματα offset που βρέθηκε στο εισαγόμενο κλιπ. Συνήθως χρησιμοποιείται με Keyframed δεδομένα για τη διατήρηση του προσανατολισμού που ορίστηκε από τον καλλιτέχνη.

Animation Clip

Τα Animation Clip είναι τα μικρότερα δομικά στοιχεία του animation στην Unity. Αντιπροσωπεύουν ένα απομονωμένο κομμάτι της κίνησης, όπως RunLeft, Jump, ή Crawl, και μπορούμε να τα χειριστούμε και να τα συνδυάσουμε με διάφορους τρόπους για να παράγουν ζωντανά τελικά αποτελεσμάτα, με τη βοήθεια των Animation State Machines, Animator ελεγκτή, ή των Blend δέντρων. Τα Animation κλιπ μπορούν να επιλεχθούν από τα εισαγόμενα δεδομένα FBX, τα οποία διαθέτουν ένα σύνολο ιδιοτήτων.

Animator Component



Κάθε GameObject που έχει ένα avatar θα έχει επίσης ένα συστατικό Animator, το οποίο είναι η σύνδεση μεταξύ του χαρακτήρα και της συμπεριφοράς του. Αναφέρεται σε έναν Animator Controller, ο οποίος χρησιμοποιείται για τη δημιουργία της συμπεριφοράς του χαρακτήρα. Αυτό περιλαμβάνει τη ρύθμιση των στατικών μηχανών, των Blend δέντρων, και των γεγονότων που πρέπει να ελέγχονται από τον κώδικα.

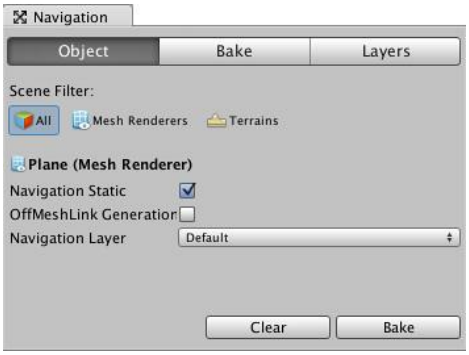
Animator Controller

Ο Animator Controller είναι το κύριο συστατικό, με το οποίο η συμπεριφορά animation προστίθεται σε ένα αντικείμενο.

3.3.19 Πλοήγηση και Pathfinding

Ένας χαρακτήρας πρέπει να είναι σε θέση να ταξιδέψει αυτόματα από την τρέχουσα θέση του σε έναν επιθυμητό προορισμό. Είναι απαραίτητη κάποια λογική ελέγχου για να επιτρέψει στο χαρακτήρα να λαμβάνει αποφάσεις σε κάθε σημείο κατά μήκος της διαδρομής του για να επιλέξει το επόμενο βήμα του. Η διαδικασία της επιλογής των κινήσεων ενός χαρακτήρα σε όλο τον κόσμο το παιχνίδι είναι γνωστή ως πλοήγηση. Πρέπει να φαίνεται ότι ο χαρακτήρας σχεδιάζει την πορεία του με έξυπνο τρόπο, λαμβάνοντας την πιο άμεση και συμφέρουσα διαδρομή, που αυτό απαιτεί ένα κομμάτι της τεχνητής νοημοσύνης (AI) για να καθορίσει τον καλύτερο τρόπο που θα πάει από το σημείο εκκίνησης στο σημείο προορισμού. Αυτό το έργο στο AI παιχνίδι είναι γνωστό ως pathfinding.

Πλοήγηση Πλεγμάτων (Navigation Meshes)

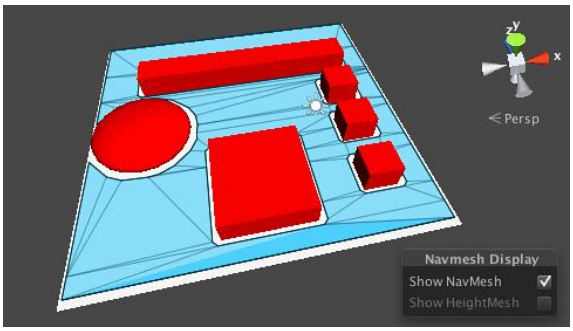


Ένα σύστημα pathfinding χρειάζεται έναν τρόπο για να αντιπροσωπεύει ποιά μέρη του κόσμου του παιχνιδιού μπορούν να προσεγιστούν από έναν χαρακτήρα, αλλά και έναν τρόπο για να καθορίσει τις πιθανές διαδρομές μεταξύ δύο οποιωνδήποτε σημείων και να επιλέξει το καλύτερο. Ένα πλέγμα (παρόμοιο με αυτό που χρησιμοποιούνται για 3D γραφικά) είναι μια καλή λύση και στα δύο προβλήματα. Η προσβάσιμη περιοχή ορίζεται από τα πολύγωνα του πλέγματος, ενώ οι πιθανές διαδρομές μέσα από το πλέγμα εύκολα αναλύονται σε hops μεταξύ γειτονικών πολυγώνων. Θεωρητικά το ίδιο πλέγμα που χρησιμοποιείται για τα γραφικά δαπέδου μπορεί επίσης να χρησιμοποιηθεί για την πλοήγηση. Στην πράξη όμως, το γραφικό πλέγμα είναι συνήθως πολύ λεπτομερές και κατά συνέπεια αναποτελεσματικό για την πλοήγηση. Γι 'αυτό ένα σύστημα AI συμπληρώνει συνήθως το πλέγμα δαπέδου με ένα απλούστερο, αόρατο πλέγμα γνωστό ως ένα πλέγμα πλοήγησης ή navmesh. Υπάρχουν τεχνικές για τη δημιουργία ενός αποτελεσματικού navmesh αυτόματα από ένα γραφικό πλέγμα δαπέδου και μερικών αριθμητικών παραμέτρων. Η διαδικασία του υπολογισμού ενός navmesh με αυτόν τον τρόπο είναι γνωστή ως baking.

Οι επιλογές του φίλτρου της σκηνή απλώς περιορίζουν την επιλογή των αντικειμένων που εμφανίζονται στη σκηνή και στο Hierarchy. Οι άλλες επιλογές εφαρμόζονται στο επιλεγμένο αντικείμενο(α), και μας επιτρέπουν να συμπεριλάβουμε ή να εξαιρέσουμε ένα αντικείμενο εντός της περιοχή βάδισης του navmesh, το οποίο ονομάζεται Static. Μπορούμε να ρυθμίσουμε το Layer πλοήγησης (Navigation Layer) ως Default εάν θέλουμε η περιοχή να

είναι προσπελάσιμη, ή διαφορετικά Walkable. Οι προσβάσιμες περιοχές αντιστοιχούν σε δάπεδα, ράμπες και γέφυρες στον κόσμο του παιχνιδιού, ενώ οι μη-προσβάσιμες περιοχές αντιστοιχούν σε τοίχους και άλλα εμπόδια αδιάβατα.

Στις Γενικές ρυθμίσεις η Ακτίνα καθορίζει πόσο κοντά ένας πλοηγούμενος χαρακτήρας μπορεί να φτάσει σε ένα τοίχο και το πλάτος του στενότερου χάσματος μεταξύ των δύο τοίχων που μπορεί περάσει. Το Ύψος αναφέρεται στο ύψος του «ταβανιού» πάνω από την επιφάνεια navmesh, καθώς μια περιοχή μπορεί να μην είναι προσβάσιμη μόνο και μόνο επειδή δεν υπάρχει αρκετός χώρος για το κεφάλι του χαρακτήρα. Η παράμετρος της Μέγιστης κλίσης (Max Slope) θέτει το όριο της κλίση όπου ένα κεκλιμένο επίπεδο γίνεται τοίχος, ενώ το Ύψος του βήματος (Step Height) είναι το μέγιστο ύψος χτυπήματος ή βήματος προς την επιφάνεια του δαπέδου που αγνοείται από το χαρακτήρα (κάθε βήμα μεγαλύτερο από αυτό το ύψος έχει ως αποτελέσματα την αποσυνδεδεμένη walkable περιοχή). Με αυτές τις παραμέτρους και επιλέγοντας το Bake υπολογίζουμε το navmesh, το οποίο δημιουργείται πολύ γρήγορα και το προκύπτον navmesh εμφανίζεται στην σκηνή ως επικάλυψη στη γεωμετρία του επιπέδου.



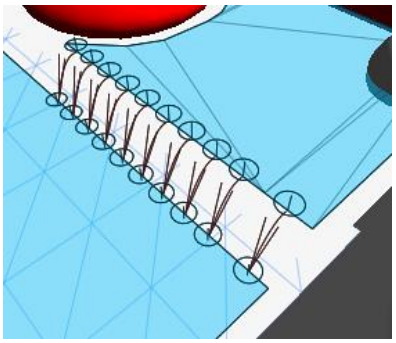
Αφού δημιουργηθεί το navmesh, πρέπει οι χαρακτήρες να εξοπλιστούν κατάλληλα ώστε να μπορούν να το χρησιμοποιήσουν. Το Nav Mesh Agent component χειρίζεται το pathfinding και τον έλεγχο κίνησης ενός χαρακτήρα. Σε scripts γίνεται η ρύθμιση του επιθυμητού σημείου προορισμού της πλοήγησης μέσω του Nav Mesh Agent.

Off-mesh Links

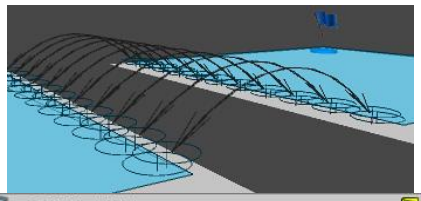
Η βάδιση στην επιφάνεια ενός κόσμου παιχνιδιού δεν περιορίζεται σε ένα μόνο πλέγμα. Συχνά χωρίζεται μια επιφάνεια δαπέδου σε έναν αριθμό ξεχωριστών τμημάτων, ακόμη και αν το δάπεδο είναι συνεχές. Η Unity μας επιτρέπει να συνδέσουμε δύο navmeshes μαζί χρησιμοποιώντας συνδέσεις off-mesh, ώστε να μπορούμε να επιτρέψουμε σε ένα χαρακτήρα να περπατήσει μεταξύ δύο σημείων σε ένα χώρο, αλλά μόνο περνώντας μέσα από ένα χαρακτηριστικό «σημείο συμφόρησης» (π.χ. διέλευση ενός ποταμού μόνο από μια στενή γέφυρα).

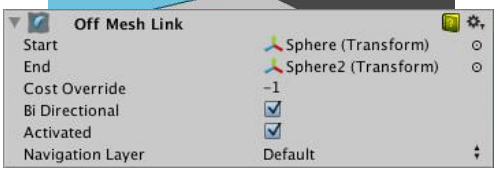
Ένας σύνδεσμος off-πλέγματος έχει δύο ακραία σημεία, ένα σε κάθε μία από τα δύο πλέγματα που πρόκειται να συνδεθούν. Όταν ένας πράκτορας πρέπει να περάσει μεταξύ δύο συνδεδεμένων navmeshes, θα κατευθυνθεί προς το τελικό σημείο που είναι πλησιέστερο μεταξύ των δύο. Στη συνέχεια θα διασχίσει τη σύνδεση με το τελικό σημείο στο άλλο navmesh. Στη συνέχεια συνεχίζει το ταξίδι του προς τον προορισμό κανονικά. Οι off-mesh συνδέσεις μετατοπίζονται αυτόματα, αλλά μπορούμε να ενεργοποιήσουμε το script να ανιχνεύσει πότε ένας πράκτορας φτάνει σε μια σύνδεση και δεν περνάει από αυτήν.

Σύνδεσμοι μπορούν εύκολα να προστεθούν αυτόματα κατά τη διαδικασία baking. Μπορούμε να ελέγξουμε το ύψος της πτώσης και την απόσταση του άλματος, τα οποία αντιπροσωπεύουν την μέγιστη κατακόρυφη και οριζόντια απόσταση που μπορεί να καλυφθεί από μια σύνδεση πλέγματος. Τα πλέγματα που βρίσκονται στο επιτρεπόμενο ύψος πτώσης και στο ύψος άλματος συνδέονται με μαύρα τόξα, τα οποία αντιπροσωπεύουν τις σχέσεις μεταξύ των πλεγμάτων. Ένας navmesh agent πάνω στο πλέγμα μπορεί να φτάσει το σημείο-στόχο σε ένα ξεχωριστό πλέγμα ακολουθόντας τις συνδέσεις μεταξύ των δύο.



Δύο navmeshes πρέπει να συνδέονται οπουδήποτε επικαλύπτονται τα άκρα τους. Αυτό συνήθως συμβαίνει όταν τα δύο πλέγματα αντιπροσωπεύουν μια ενιαία επιφάνεια που περιλαμβάνει ένα βήμα ή κένο. Οι συνδέσεις μπορούν να τοποθετηθούν έτσι ώστε να επιτρέπουν μια κατάσταση στο παιχνίδι, όπως μία μόνο σύνδεση για να αντιπροσωπεύει μια πόρτα, γέφυρα ή άλλο στενό σημείο επαφής ανάμεσα σε δύο ξεχωριστά τμήματα του αγωνιστικού χώρου. Αυτό βοηθάει στην απόδοση του pathfinding σε ένα μεγάλο κόσμο παιχνιδιού, αν αυτός ο κόσμος είναι κατανεμημένος σε χωριστά navmeshes με λίγα μόνο περιορισμένα σημεία επαφής μεταξύ τους. Η αυτόματη δημιουργία συνδέσμου δεν θα είναι σε θέση να τοποθετήσει τους συνδέσμους με σύνεση, αλλά μπορούμε να προσθέσουμε δικές μας συνδέσεις μεταξύ πλεγμάτων χειροκίνητα χρησιμοποιώντας το στοιχείο Off Mesh Link component .





Είναι πιθανό η navmesh στατική γεωμετρία στη σκηνή να έχει αποσυνδεθεί, καθιστώντας έτσι αδύνατη στους πράκτορες για να πάνε από το ένα μέρος του κόσμου στο άλλο. Για να αντιμετωπιστεί αυτή η κατάσταση, η Unity έχει ένα σύστημα συνδέσμων Off-mesh.

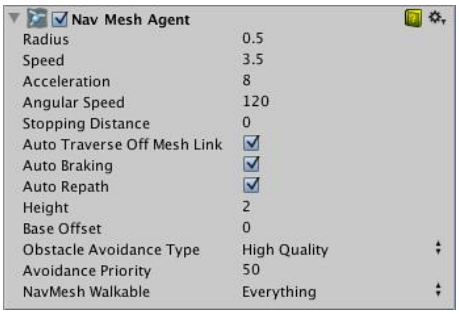
Επίπεδα πλοήγησης και κόστος (NavMesh Layers and Costs)

Για να φαίνονται οι πράκτορες έξυπνοι, σχεδιάζουμε το σύστημα ώστε να αναζητήσουν την βέλτιστη διαδρομή από το σημείο εκκίνησης στο σημείο προορισμού. Η συντομότερη διαδρομή θεωρείται βέλτιστη, αλλά υπάρχουν πολλές καταστάσεις στην πραγματική ζωή όπου ο υπολογισμός δεν είναι τόσο απλός και μπορεί να είναι πιο δύσκολο να προχωρήσουμε μέσα ορισμένες περιοχές από ό,τι άλλες και μπορεί να υπάρχουν ανησυχίες εκτός από το χρόνο. Κάθε βήμα κατά μήκος μιας διαδρομής έχει ένα συγκεκριμένο κόστος που συνδέεται με αυτό. Το κόστος μπορεί να αποτελεί τη δυσκολία του εδάφους, του κινδύνου ή πολλούς άλλους παράγοντες. Η βέλτιστη διαδρομή είναι αυτή με το χαμηλότερο συνολικό κόστος και αυτή δεν θα είναι πάντα η συντομότερη. Κάθε τμήμα navmesh μπορεί να συσταθεί με την αξία του

κόστους μέσω της χρήσης των στρωμάτων Navmesh (Navmesh Layers). Κάθε γεωμετρία που χαρακτηρίζεται ως Navmesh Static θα ανήκει σε ένα Navmesh Layer. Το κόστος ενός μονοπατιού είναι μια πολύ αφηρημένη έννοια και θα μπορούσε να περιλαμβάνει πολλούς διαφορετικούς παράγοντες, όπως το κίνδυνο, την ορατότητα, τον προσανατολισμό, την αποφυγή βλαβερών αντικειμένων ή σημείων, και τη δυσκολία ή σκοπιμότητα του εδάφους.

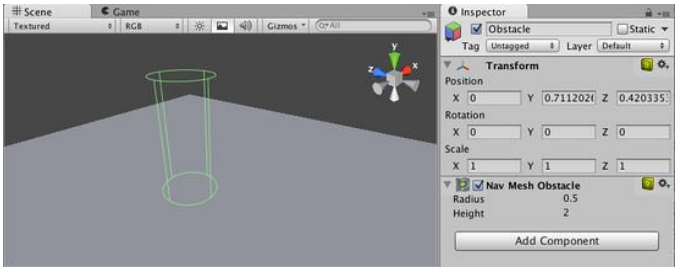
Κατά τη διάρκεια του pathfinding, αντί να συγκρίνονται τα μήκη των πιθανών τμημάτων διαδρομής, υπολογίζεται το κόστος του κάθε τμήματος. Αυτό γίνεται με την κλιμάκωση του κόστους του navmesh layer μήκους του κάθε τμήματος για αυτό το συγκεκριμένο τμήμα. Όλα τα κόστη είναι ορισμένα σε 1 όταν η βέλτιστη διαδρομή είναι ισοδύναμη με τη συντομότερη διαδρομή.

NavMesh Agent



Το NavMesh Agent component χρησιμοποιείται σε συνδυασμό με το pathfinding, και ορίζουμε σε αυτό τις πληροφορίες σχετικά με το πώς ο πράκτορας θα περιηγείται την NavMesh.

Navmesh Εμπόδιο



Τα σταθερά εμπόδια στο navmesh μπορούν να συσταθούν ως μέρος της διαδικασίας baking. Είναι επίσης δυνατόν να έχουμε δυναμικά εμπόδια σε μία σκηνή, τα οποία θα πρέπει να αποφεύγονται από τους πράκτορες καθώς κινούνται γύρω. Τέτοια δυναμικά εμπόδια μπορούν να προσδιορίζονται με τη χρήση του Navmesh Obstacle component, το οποίο μπορεί να προστεθεί σε οποιοδήποτε GameObject και θα κινηθεί όπως κινείται ένα αντικείμενο.

3.3.20 Asset Components

Τα περιουσιακά στοιχεία είναι τα μοντέλα, οι υφές, οι ήχοι και όλα τα άλλα περιεχόμενα του αρχείου από τα οποία μπορούμε να φτιάξουμε το παιχνίδι μας.

Εισαγωγή αρχείων Maya

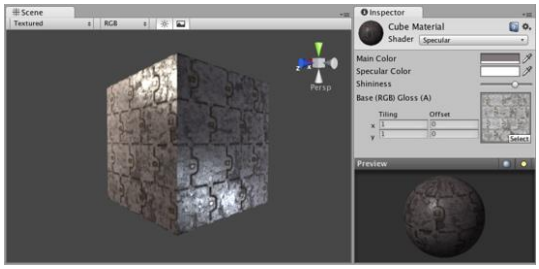
Η Unity αναγνωρίζει τα αρχεία Maya στην κανονική τους μορφή .Mb , .Ma . Όταν εισάγουμε ένα αρχείο Maya στη Unity, το αρχείο μπορεί να περιλαμβάνει όλους τους κόμβους με τη θέση, την περιστροφή και την κλίμακά τους, τα σημεία του άξονα και τα ονόματα, τα πλέγματα με τα χρώματα κορυφών, τα Normals και μέχρι 2 UV σετ, τα υλικά με υφή, το χρώμα, και πολλαπλά υλικά ανά πλέγμα, τα Animations FK & IK, τα Bone-based Animations, και Blend Σχήματα.

Γραμματοσειρά (Font)

Οι γραμματοσειρές μπορούν να δημιουργηθούν ή να εισαχθούν με τη χρήση είτε του GUI κειμένου ή του κειμένου Mesh Components. Οι υποστηριζόμενες μορφές γραμματοσειρών είναι οι γραμματοσειρές TrueType (.Ttf ή .Dfont αρχεία) και OpenType (.Otf αρχεία) και υποστηρίζεται η δυναμική απεικόνιση γραμματοσειρών. Μπορούμε να αλλάξουμε το μέγεθος, το χρώμα και το υλικό της γραμματοσειράς, να ρυθμίσουμε το κείμενο που βασίζεται σε Unicode μέσω scripting. Το αποτέλεσμα εμφανίζεται σε ένα UnityGUI ή σε ένα GUIText.

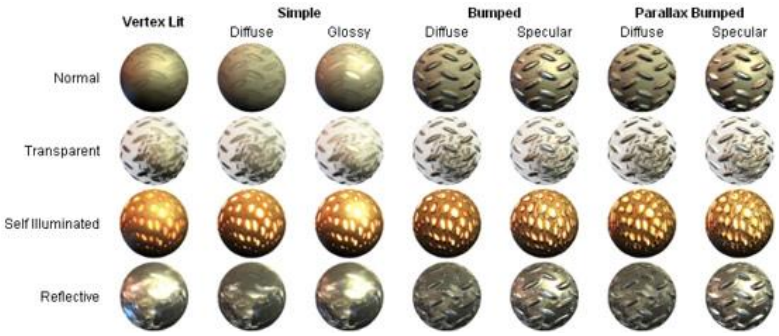
Materials και Shaders

Τα Υλικά χρησιμοποιούνται σε συνδυασμό με το πλέγμα ή τα Συστήματα Σωματιδίων που επισυνάπτονται στο GameObject. Παίζουν σημαντικό ρόλο στον καθορισμό του τρόπου εμφάνισης του αντικειμένου. Τα υλικά συνδέονται με τον Shader που χρησιμοποιείται για να καταστήσει το πλέγμα ή τα σωματίδια, έτσι ώστε αυτά τα συστατικά δεν μπορούν να εμφανιστούν χωρίς κάποιο είδος του υλικού.



Τα Υλικά χρησιμοποιούνται για να τοποθετήσουμε υφές σε GameObjects. Δεν μπορούμε να προσθέσουμε μια υφή άμεσα χωρίς υλικό, καθώς η κάθε τοποθέτηση μιας υφής δημιουργεί έμμεσα ένα νέο υλικό. Η σωστή ροή εργασίας είναι η δημιουργία ενός υλικού, η επιλογή ενός Shader, και η επιλογή της υφής αντικειμένου για να εμφανιστεί μαζί με αυτό. Υπάρχει στενή σχέση μεταξύ υλικών και Shaders. Οι Shaders περιέχουν κώδικα ο οποίος καθορίζει τι είδους ιδιότητες και αντικείμενα να χρησιμοποιήσουμε. Τα υλικά μας επιτρέπουν να προσαρμόσουμε τις ιδιότητες και να εκχωρήσουμε τα αντικείμενα. Μπορούμε να επιλέξουμε ποιο Shader θα χρησιμοποιεί το κάθε υλικό, καθώς και να αλλάξουμε τις ιδιότητές του. Οι ιδιότητες μπορεί να είναι τα χρώματα, οι υφές, οι αριθμοί, ή τα διανύσματα.

Διατίθονται πάνω από ογδόντα ενσωματωμένοι Shaders. Ένας Shader ουσιαστικά ορίζει τη σκίαση στο παιχνίδι και είναι μια σειρά από ιδιότητες (συνήθως υφές). Οι shaders υλοποιούνται μέσω υλικών, τα οποία συνδέονται άμεσα με GameObjects. Δημιουργούνται από προγραμματιστές γραφικών, χρησιμοποιώντας τη γλώσσα ShaderLab η οποία είναι αρκετά απλή. Η ομαλή λειτουργία ενός shader σε μια ποικιλία καρτών γραφικών είναι μια περίπλοκη εργασία και απαιτεί ολοκληρωμένη γνώση της λειτουργίας των καρτών γραφικών.



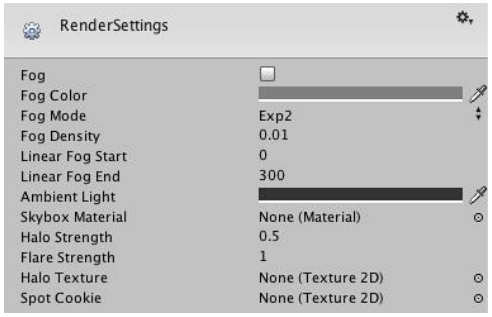
Meshes

Τα Πλέγματα αποτελούν ένα μεγάλο μέρος του 3D κόσμου. Εκτός από μερικά plugins η Unity δεν περιλαμβάνει εργαλεία μοντελοποίησης. Ωστόσο υπάρχει μεγάλη αλληλεπίδραση με τα περισσότερα 3D modeling προγράμματα. Υποστηρίζει τριγωνικά ή Quadrangulated πολυγωνικά πλέγματα. Οι Nurbs, NURMS, Subdiv επιφάνειες πρέπει να μετατραπούν σε πολύγωνα. Η εισαγωγή πλεγμάτων μπορεί να επιτευχθεί από τις εξαγόμενες 3D μορφές αρχείων, όπως .FBX ή .OBJ, ή από 3D μορφές αρχείων 3D προγραμμάτων, όπως .Max και .Blend από το 3D Studio Max ή Blender, καθώς επίσης και μέσα από τη Unity.

3.3.21 Settings Managers

Render Settings

Οι Render ρυθμίσεις περιέχουν προκαθορισμένες τιμές για μια σειρά από οπτικά στοιχεία στη σκηνή, όπως φώτα και skyboxes. Χρησιμοποιείται για να καθορίσει κάποιες βασικές οπτικές ομοιότητες του κάθε ατόμου σκηνή του παιχνιδιού. Μπορούμε να έχουμε δύο επίπεδα στο ίδιο περιβάλλον για τη μέρα και τη νύχτα χρησιμοποιώντας το Ambient Light για να ελέγξουμε το φωτισμό.

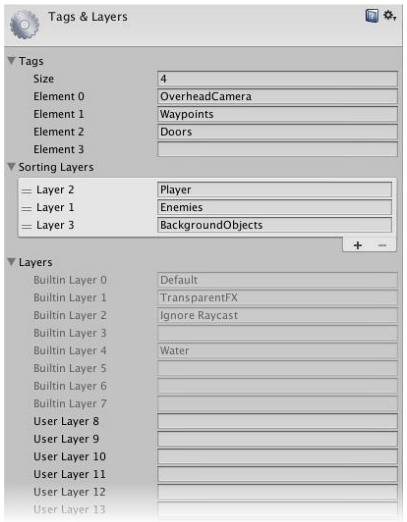


Ομίχλη

Μπορούμε να ενεργοποιήσουμε ομίχλη στην σκηνή μας, και να ρυθμίσουμε την εμφάνιση, την πυκνότητα και το χρώμα της. Η προσθήκη ομίχλης συχνά χρησιμοποιείται για τη βελτιστοποίηση της απόδοσης, με τη διασφάλιση ότι τα μακριά αντικείμενα ξεθωριάζουν και δεν έχουν συνταχθεί, ρυθμίζοντας κατάλληλα την κάμερα.

Ετικέτες και Επίπεδα (Tags and Layers)

Οι ετικέτες και τα στρώματα μας επιτρέπουν να ορίσουμε Layers, Ετικέτες και την ταξινόμηση των στρωμάτων. Οι ετικέτες είναι δείκτες που μπορούν να χρησιμοποιηθούν για τον εντοπισμό αντικειμένων στο έργο μας. Μπορούμε να προσθέσουμε και να αφαιρέσουμε όσες ετικέτες θέλουμε και να τις ορίσουμε στα GameObjects που επιθυμούμε να βρούμε. Τα Layers χρησιμοποιούνται για τη δημιουργία ομάδων των αντικειμένων που μοιράζονται συγκεκριμένα χαρακτηριστικά. Τα Layers χρησιμοποιούνται κυρίως για να περιοριστούν εργασίες όπως raycasting ή απόδοσης, έτσι ώστε να εφαρμόζονται μόνο για τις ομάδες των αντικειμένων που είναι σχετικές. Τα πρώτα οκτώ στρώματα είναι προεπιλεγμένα και δεν είναι επεξεργάσιμα αλλά τα στρώματα 8-31



μπορούν να υποστούν επεξεργασία, όμως σε αντίθεση με ετικέτες ο αριθμός των στρωμάτων δεν μπορεί να αυξηθεί.

Time Manager

Fixed Timestep

Το Fixed TimeStep είναι ένα καρέ που δηλώνει πότε πραγματοποιούνται οι μετρήσεις φυσικής και γεγονότα FixedUpdate( ). Η σταθερά του χρόνου του βηματισμού είναι πολύ σημανική για τη σταθερή προσομοίωση της φυσικής. Δεν είναι όλοι οι υπολογιστές ή τα κινητά ίσα μεταξύ τους, και διαφορετικές διαμορφώσεις hardware θα τρέξουν τα παιχνίδια με διαφορετικές επιδόσεις. Ως εκ τούτου, η φυσική πρέπει να υπολογιστεί ανεξάρτητα από το frame rate του παιχνιδιού. Οι υπολογισμοί της φυσικής, όπως η ανίχνευση σύγκρουσης και η Rigidbody κίνηση πραγματοποιείται με διακριτά βήματα καθορισμένου χρόνου που δεν εξαρτώνται από το frame rate. Το γεγονός αυτό καθιστά την προσομοίωση πιο συνεπής σε διαφορετικούς υπολογιστές και διαφορετικά κινητά ή όταν συμβαίνουν οι αλλαγές στο ρυθμό καρέ. Για παράδειγμα, ο ρυθμός καρέ μπορεί να μειωθεί λόγω της εμφάνισης πολλών παιχνιδιών στην οθόνη του, είτε επειδή ο χρήστης ξεκίνησε μια άλλη εφαρμογή στο παρασκήνιο.

Πριν εμφανιστεί το κάθε πλαίσιο στην οθόνη, η Unity υπολογίζει το fixed delta time και εκτελεί τους υπολογισμούς φυσικής μέχρι να φτάσει στην τρέχουσα στιγμή. Όσο μικρότερη είναι η τιμή του Fixed TimeStep, τόσο πιο συχνά υπολογίζεται η φυσική. Ο αριθμός των σταθερών καρέ ανά δευτερόλεπτο μπορεί να υπολογιστεί διαιρώντας το 1 από το σταθερό Timestep. Έτσι έχουμε 1 / 0,02 = 50 σταθερά καρέ ανά δευτερόλεπτο και 1 / 0,05 = 20 σταθερά πλαίσια ανά δευτερόλεπτο. Μια μικρότερη σταθερή τιμή οδηγεί σε πιο ακριβή προσομοίωση φυσικής, αλλά είναι βαρύτερο για την CPU .

Maximum Allowed Timestep

Το Maximum Allowed Timestep είναι ένα καρέ όταν το frame-rate είναι χαμηλό. Υπολογισμοί Φυσικής και FixedUpdate ( ) γεγονότα δεν θα πραγματοποιηθούν για μεγαλύτερο χρονικό διάστημα από ό, τι ορίζεται. Η σταθερά χρόνου εξασφαλίζει τη σταθερή προσομοίωση φυσικής, αλλά μπορεί να προκαλέσει αρνητικές επιπτώσεις στην απόδοση, αν το παιχνίδι είναι βαρύ για τη φυσική και ήδη τρέχει αργά ή περιστασιακά βυθίζεται σε χαμηλό ρυθμό καρέ. Όσο περισσότερο χρόνο χρειάζεται το πλαίσιο για να επεξεργαστεί, τόσα περισσότερα σταθερά βήματα ενημέρωσης θα πρέπει να εκτελεστούν για το επόμενο πλαίσιο. Αυτό οδηγεί σε υποβάθμιση της απόδοσης. Για να αποφευχθεί ένα τέτοιο σενάριο το Maximum Allowed Timestep εξασφαλίζει ότι οι υπολογισμοί της φυσικής δεν θα τρέξουν περισσότερο από το καθορισμένο όριο.

Εάν το πλαίσιο χρειάζεται περισσότερο χρόνο για να επεξεργαστεί από το χρόνο που ορίζεται, τότε η φυσική " προσποιείται " ότι το πλαίσιο χρειάστηκε τα μέγιστα δευτερόλεπτα. Με άλλα λόγια, αν το frame rate πέφτει κάτω από κάποιο όριο, τότε τα στερεά σωμάτα θα επιβραδύνουν λίγο για να καλύψει η CPU τη διαφορά. Το Maximum Allowed Timestep επηρεάζει τον υπολογισμό της φυσικής και τα FixedUpdate() γεγονότα. Το μέγιστο καθορίζεται σε δευτερόλεπτα, όπως η Σταθερή Timestep. Συνεπώς το 0.1 θα κάνει τη φυσική και τα FixedUpdate() γεγονότα να επιβραδύνουν, αν το frame rate κατέβει κάτω από το 1/0,1 = 10 καρέ ανά δευτερόλεπτο.

Time Scale

Το Time Scale είναι η ταχύτητα του χρόνου. Η τιμή 1 σημαίνει πραγματικός χρόνος, η τιμή 0,5 σημαίνει μισή ταχύτητα, ενώ η τιμή 2 είναι διπλάσια ταχύτητα.

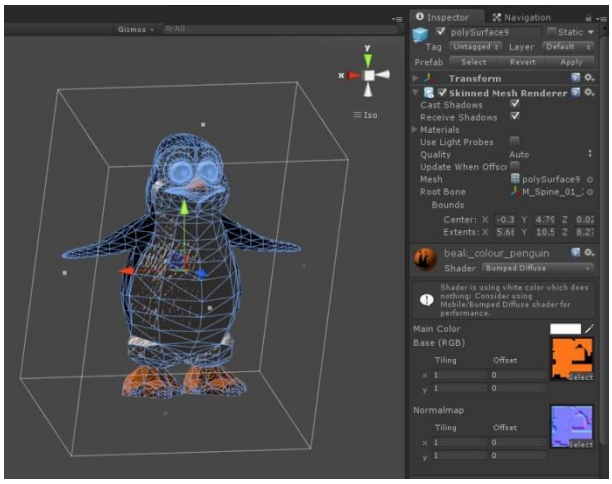
3.3.22 Mesh Components

Mesh Renderer

Το Mesh Renderer παίρνει τη γεωμετρία από το φίλτρο του πλέγματος (Mesh Filter) και την τοποθετεί στη θέση που ορίζεται από το Transform του αντικειμένου. Τα πλέγματα εισάγονται από 3D προγράμματα και μπορούν να χρησιμοποιούν πολλαπλά υλικά. Κάθε submesh θα χρησιμοποιήσει ένα υλικό από τη λίστα υλικών. Εάν υπάρχουν περισσότερα υλικά στο Mesh Renderer από όσα submeshes υπάρχουν στο πλέγμα, το πρώτο submesh θα αποδοθεί με καθένα από τα υπόλοιπα υλικά. Με κόστος των επιδόσεων, αυτό θα να δημιουργήσει multi-pass απόδοση σε αυτό το submesh.

Skinned Mesh Renderer

Ο Skinned Mesh Renderer προστίθεται αυτόματα στα εισαγόμενα πλέγματα, όταν το εισαγόμενο πλέγμα έχει δέρμα. Τα Skinned πλεγμάτα χρησιμοποιούνται για την απόδοση χαρακτήρων. Οι χαρακτήρες κινουμένων σχεδίων χρησιμοποιούν τα οστά, και το κάθε οστό επηρεάζει ένα μέρος του πλέγματος. Τα πολλαπλά οστά μπορεί να επηρεάσούν την ίδια κορυφή. Το animation χωρίς κόκαλα είναι η κύρια τεχνική που χρησιμοποιείται για να παραμορφώσουμε το σχήμα ενός πλέγματος, έτσι ώστε τα άκρα να μπορούν να λυγίσουν στις αρθρώσεις. Το Rigidbody και ο χαρακτήρας μπορούν να συνδέονται με τα οστά, έτσι ώστε να κινούνται ρεαλιστικά και με φυσικό τρόπο, το οποίο είναι γνωστό ως "ragdoll" effect.



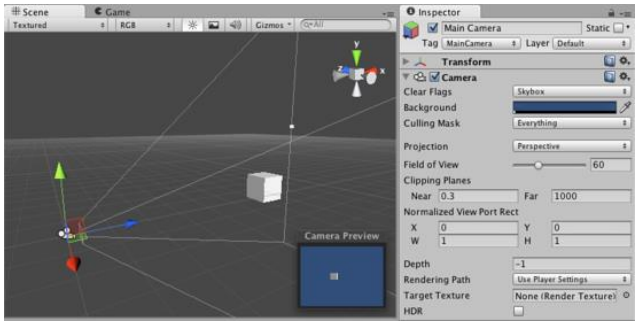
Text Mesh

Το Mesh Κείμενο δημιουργεί 3D γεωμετρία που εμφανίζει συμβολοσειρές κειμένου. Το πλέγμα κειμένου τοποθετεί κείμενο στη 3D σκηνή. Για να δημιουργήσουμε κείμενο για 2D γραφικά περιβάλλοντα, χρησιμοποιούμε ένα στοιχείο GUI κειμένου.

3.3.23 Rendering Components

Κάμερα

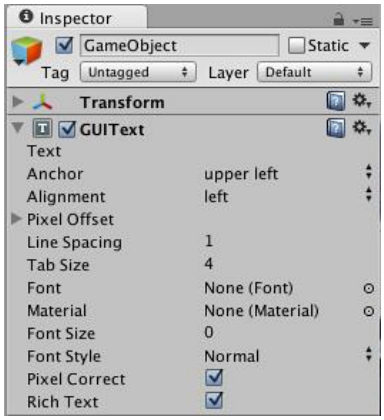
Οι κάμερες είναι οι συσκευές που συλλαμβάνουν και εμφανίζουν τον κόσμο στον παίκτη. Μπορούμε να έχουμε απεριόριστο αριθμό των καμερών σε μια σκηνή, να προσαρμόσουμε κώδικα σε αυτές, μπορούν να ρυθμιστούν σε οποιαδήποτε σειρά, σε οποιαδήποτε θέση στην οθόνη, ή μόνο σε ορισμένα τμήματα της οθόνης. Επίσης μπορούμε να δημιουργήσουμε μια σχέση γονέα-παιδιού όπου το παιδί θα είναι η κάμερα ώστε να ακολουθεί παντού τον γονεά στο παιχνίδι.



Οι κάμερες χρησιμοποιούνται για την παρουσίαση του παιχνιδιού στη συσκευή αναπαραγωγής. Μπορούν να προσαρμοστούν, να ελεγχθούν από τον κώδικα, ή να αποτελούν το παιδί σε μια pareting σχέση με ένα GameObjetc. Για ένα first-person shooter παιχνίδι, ο χαρακτήρας του παίκτη αποτελεί το γονέα και η κάμερα το παιδί ώστε να τον ακολουθεί παντού και τοποθετείται στο επίπεδο των ματιών του χαρακτήρα, ενώ σε ένα third-person shooter θα τοποθετητεί από πίσω του όπως σε ένα παιχνίδι αγώνων η κάμερα ακολουθεί το όχημα του παίκτη. Μπορούμε να δημιουργήσουμε πολλαπλές κάμερες και να εκχωρήσουμε κάθε μία σε διαφορετικό βάθος. Η προτεραιότητα της κάθε κάμερα ορίζεται αναλόγως το βάθος που της έχουμε ορίσει, η κάμερα με μεγαλύτερο βάθος επικαλύπτει αυτές με μικρότερο βάθος. Μπορούμε να ρυθμίσουμε το μέγεθος και τη θέση προβολή στην οθόνη της κάμερας αλλάζοντας τις τιμές τους.

GUI Layer

Ένα GUI Layer component συνδέεται με μια κάμερα για να καταστήσει δυνατή την απόδοση των 2D γραφικών περιβάλλοντων. Όταν ένα GUI Layer είναι συνδεδεμένο με μια κάμερα προβάλονται όλες οι υφές GUI και τα GUI Κείμενα στη σκηνή. Τα GUI στρώματα δεν επηρεάζουν την UnityGUI.

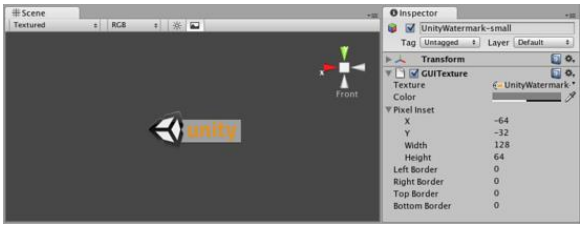


GUI Text

Το GUI Κείμενο εμφανίζει κείμενο κάθε γραμματοσειράς που εισάγουμε σε συντεταγμένες της οθόνης. Χρησιμοποιείται για την εκτύπωση κειμένου πάνω στην οθόνη σε 2D. Η κάμερα πρέπει να έχει

ένα GUI Layer προκειμένου να καταστεί το κείμενο, το οποίο το έχει από προεπιλογή. Τα GUI Κείμενα τοποθετούνται χρησιμοποιώντας μόνο τους X και Y άξονες, και τοποθετούνται σε συντεταγμένες οθόνης, όπου (0,0) είναι το κάτω-αριστερά και (1,1) είναι η κορυφαία δεξιά γωνία της οθόνης.

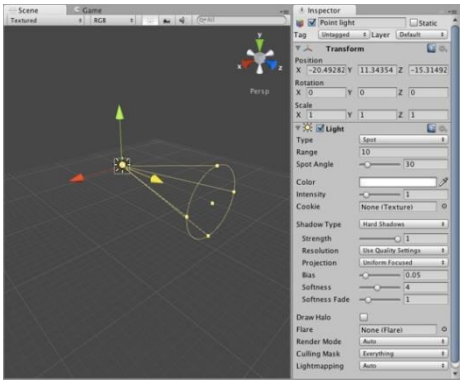
GUI Texture



Οι GUI υφές εμφανίζονται ως επίπεδες εικόνες σε 2D. Είναι φτιαγμένες ειδικά για τα στοιχεία διεπαφής του χρήστη, τα κουμπιά ή τις διακοσμήσεις. Η τοποθέτηση και η κλιμάκωση τους γίνεται κατά μήκος του x και y άξονες μόνο, και μετρώνται σε συντεταγμένες οθόνης αντί σε World συντεταγμένες.

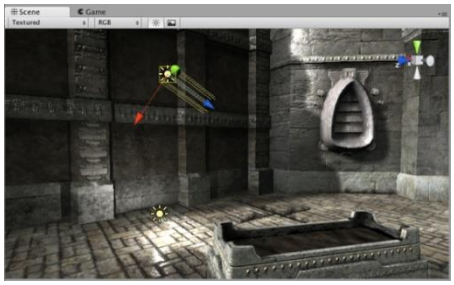
Οι GUI υφές είναι τέλειες για την παρουσίαση υπόβαθρων interface του παιχνιδιού, κουμπιά ή άλλων στοιχείων για τον παίκτη. Μέσω scripting μπορούμε να παρέχουμε εύκολα οπτική ανάδραση για διαφορετικές καταστάσεις της υφής, όπως όταν το ποντίκι πλανάται πάνω από την υφή, ή πατάμε με το ποντίκι ένα σημείο.

Φωτισμός



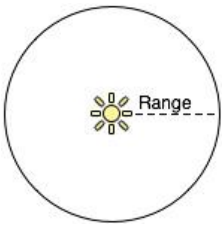
Τα φώτα είναι ένα ουσιαστικό μέρος της κάθε σκηνής. Ενώ τα meshes και οι υφές καθορίζουν το σχήμα και την εμφάνιση μιας σκηνής, τα φώτα καθορίζουν το χρώμα και τη διάθεση του 3D περιβάλλοντος, φωτίζουν τις σκηνές και τα αντικείμενα για να δημιουργήσουν την τέλεια οπτική διάθεση. Είναι δυνατή η τοποθέτηση περισσότερων από μία φωτεινή πηγή σε κάθε σκηνή. Τα φώτα μπορούν να χρησιμοποιηθούν για να προσομοιώσουν τον ήλιο, τα φανάρια, το φακό, τη λάμψη από πυροβολισμό, ή εκρήξεις, καθώς επίσης μπορούν να αλλάξουν το κλίμα της σκηνής αλλάζοντας το χρώμα του φωτισμού.

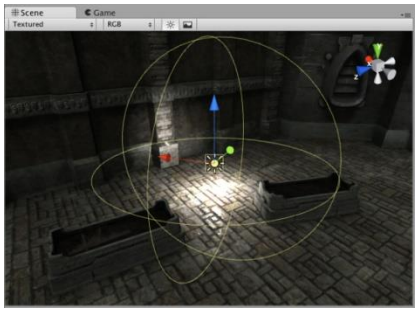
Υπάρχουν τέσσερις τύποι φώτων:



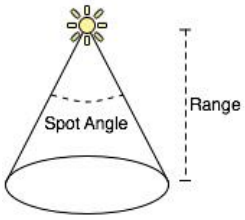
- Τα φώτα κατεύθυνσης (Directional lights) που τοποθετούνται απείρως μακριά και επηρεάζουν τα πάντα στη σκηνή, σαν τον ήλιο. Χρησιμοποιούνται κυρίως σε εξωτερικές σκηνές για τον ήλιο και το φως του φεγγαριού. Μπορούν να επηρεάσουν όλες τις επιφάνειες των αντικειμένων στη σκηνή. Είναι το λιγότερο ακριβό σχετικά με τον επεξεργαστή γραφικών.

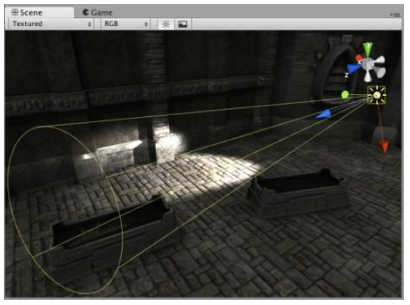
- Σημείο φωτός (Point lights) που λάμπει από μια τοποθεσία ενός σημείου σε όλες τις κατευθύνσεις εξίσου, όπως μια λάμπα. Είναι τα πιο κοινά φώτα σε παιχνίδια υπολογιστή, χρησιμοποιούνται συνήθως για εκρήξεις, λαμπτήρες, κ.λπ. Έχουν ένα μέσο κόστος για τον επεξεργαστή γραφικών (αν και οι σκιές του point light είναι το πιο ακριβό).





- Spot lights που λάμπει από ένα σημείο σε μια κατεύθυνση και φωτίζει μόνο τα αντικείμενα μέσα σε ένα κώνο. Είναι ιδανικό για φακούς, προβολείς των αυτοκινήτων ή φανοστάτες. Είναι το πιο ακριβό στον επεξεργαστή γραφικών.





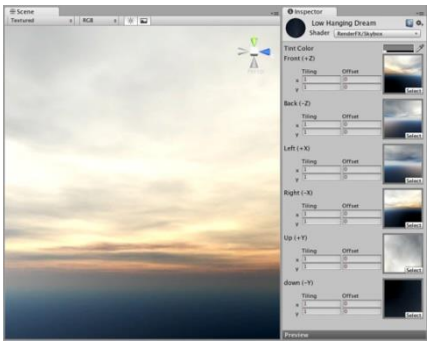
- Φώτα περιοχής (Area lights), διαθέσιμα μόνο για τη δημιουργία lightmap , λάμπουν σε όλες τις κατευθύνσεις προς τη μία πλευρά ενός ορθογώνιου τμήματος της περιοχής. Ρίχνει φως σε όλα τα αντικείμενα εντός της εμβέλειας του φωτός. Το μέγεθος του ορθογωνίου καθορίζεται από τις πλάτος και το ύψος του πλαισίου, δηλαδή την πλευρά στην οποία ρίχνει το φως, η οποία είναι η ίδια με τη θετική κατεύθυνση Z του φωτός. Το φως εκπέμπεται από όλη την επιφάνεια του ορθογωνίου, έτσι η σκίαση και οι σκιές από τα αντικείμενα είναι πιο ομαλές από ό,τι με τις πηγές point ή directional φωτισμού. Δεδομένου ότι ο υπολογισμός φωτισμού είναι εντατικός, τα φώτα περιοχής δεν είναι διαθέσιμα κατά το χρόνο εκτέλεσης και μπορεί να υπάρξουν μόνο σε lightmaps.

Τα φώτα μπορούν να έχουν σκιές και οι ιδιότητές τους μπορούν να ρυθμιστούν με βάση το κάθε είδος φωτισμού. Τα φώτα που δημιουργούμε είναι πραγματικού χρόνου, ο φωτισμός τους υπολογίζεται κάθε καρέ ενώ το παιχνίδι βρίσκεται σε λειτουργία. Αν γνωρίζουμε ότι το φως δεν θα αλλάξει, μπορούμε να κάνουμε το παιχνίδι πιο γρήγορο και να φαίνεται καλύτερο με τη χρήση Lightmapping. Υποστηρίζονται διαφορετικά μονοπάτια Rendering, τα οποία επηρεάζουν κυρίως τα Φώτα και τις σκιές, επιλέγοντας έτσι το σωστό μονοπάτι απόδοσης ανάλογα με τις απαιτήσεις του παιχνιδιού μπορεί να υπάρξει βελτιώσει στην απόδοση του έργου.

Τα φώτα μπορούν να αποδοθούν με μία από τις δύο μεθόδους: φωτισμός κορυφών και φωτισμού pixel. Ο Vertex φωτισμός υπολογίζει μόνο το φωτισμό στις κορυφές των μοντέλων παιχνιδιών και παρεμβάλλει το φωτισμό πάνω από τις επιφάνειες των μοντέλων. Ο φωτισμός των Pixel υπολογίζεται σε κάθε pixel της οθόνης, και ως εκ τούτου είναι πολύ πιο ακριβός. Ενώ ο φωτισμός pixel είναι πιο αργός, επιτρέπει κάποια εφέ που δεν είναι δυνατά με το φωτισμό κορυφών. Τα σχήματα Spotlight και τα σημεία Point light είναι πολύ καλύτερα όταν εμφανίζονται με pixel.

Τα φώτα έχουν μεγάλο αντίκτυπο στην απόδοση της ταχύτητας, ως εκ τούτου πρέπει να γίνει μια ανταλλαγή μεταξύ της ποιότητας του φωτισμού και την ταχύτητα του παιχνιδιού. Αφού ο φωτισμός των pixel είναι πολύ πιο ακριβός από το φωτισμό κορυφών, η Unity θα αποδώσει μόνο τα φωτεινότερα ανά-pixel ποιότητας. Από προεπιλογή το φως αποδίδεται αυτόματα με βάση το πόσο το αντικείμενο επηρεάζεται από το φως.

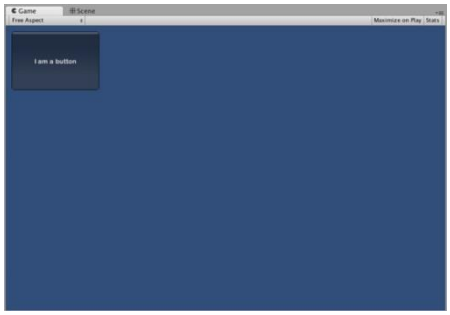
Ουρανός (Skybox)



Τα Skyboxes είναι ένα περίβλημα γύρω από όλη τη σκηνή και πέρα από τον κόσμο που εμφανίζει τον ουρανό. Το υλικό που χρησιμοποιείται για να αποδόση τον ουρανό περιέχει έξι υφές, χρησιμοποιώντας το Skybox Shader.

Τα Skyboxes αποδίδονται πριν από οτιδήποτε άλλο στη σκηνή, ώστε να δωθεί η εντύπωση ενός πολύπλοκου τοπίου στον ορίζοντα. Αποτελεί ένα κουτί από έξι υφές, μία για κάθε πλευρά του. Έχουμε δύο επιλογές εφαρμογής των skyboxes. Μπορούμε να τα προσθέσουμε σε μια κάμερα, συνήθως στην κύρια κάμερα, ή να δημιουργήσουμε ένα προεπιλεγμένο Skybox στις ρυθμίσεις του Render υλικών του Skybox. Οι Render Settings είναι πιο χρήσιμες αν θέλουμε όλες τις κάμερες στη σκηνή μας να μοιράζονται το ίδιο Skybox. Η προσθήκη του Component Skybox σε μια κάμερα παρακάμπτει το προεπιλεγμένο Skybox στα Render Settings, και έτσι μπορεί η κάθε κάμερα στο παιχνίδι να έχει ένα διαφορετικό Skybox.

Στοιχεία Διεπαφής Παιχνιδιού



Η Unity παρέχει μια σειρά από επιλογές για τη δημιουργία των γραφικών διεπαφών χρήστη του παιχνιδιού μας (GUI). Μπορούμε να χρησιμοποιήσουμε τα αντικείμενα GUI κειμένα (GUI Text) και GUI Υφές (GUI Texture) στη σκηνή, ή να δημιουργήσουμε το περιβάλλον με scripts χρησιμοποιώντας τη UnityGUI.

Η UnityGUI μας επιτρέπει να δημιουργήσουμε μια μεγάλη ποικιλία από εξαιρετικά λειτουργικά GUIs πολύ γρήγορα και εύκολα. Η δημιουργία, ο έλεγχος και ο χειρισμός τους μπορεί να γίνει μέσω κώδικα καλώντας τη συνάρτηση GUI.

3.3.24 UnityGUI Group

Το UnityGUI είναι ένα ενσωματωμένο σύστημα δημιουργίας GUI. Αποτελείται από τη δημιουργία διαφορετικών ελέγχων, και τον καθορισμό του περιεχομένου και της εμφάνισης των εν λόγω ελέγχων. Τα components του UnityGUI, GUI Skin και GUI Style, μας επιτρέπουν να ορίσουμε την εμφάνιση των ελέγχων. Επιτρέπει τη δημιουργία μιας μεγάλης ποικιλίας από εξαιρετικά λειτουργικά GUIs πολύ γρήγορα και εύκολα με λίγες γραμμές κώδικα. Ο κώδικας παράγει GUI Controls που ελέγχονται με την συνάρτηση OnGUI().

Controls

Υπάρχει μια σειρά από διαφορετικούς ελέγχους GUI που μπορούμε να δημιουργήσουμε. Ο λειτουργικός έλεγχος είναι απαραίτητος για το παιχνίδι, και η εμφάνιση αυτών των ελέγχων είναι πολύ σημαντική για την αισθητική του παιχνιδιού μας. Στην UnityGUI μπορούμε να ρυθμίσουμε την εμφάνιση των ελέγχων με πολλές λεπτομέρειες. Μπορούμε να ελέξγουμε το περιεχόμενο, το ύφος και την εμφάνιση. Οι εμφανίσεις ελέγχου υπαγορεύονται με GUIStyles. Όταν έχουμε ένα μεγάλο αριθμό διαφορετικών GUIStyles, μπορούμε να τα καθορίσουμε σε ένα ενιαίο GUISkin. Ένα GUISkin δεν είναι τίποτα περισσότερο από μια συλλογή GUIStyles. Τα Styles καθορίζουν την εμφάνιση του GUI ελέγχου. Δεν χρειάζεται να χρησιμοποιήσουμε ένα Skin αν θέλουμε να χρησιμοποιήσουμε ένα στυλ.

Label

Η ετικέτα είναι μη διαδραστική και είναι μόνο για επίδειξη. Δεν μπορούμε να την επιλέξουμε ή να μετακινηθεί. Είναι καλύτερη για την εμφάνιση μόνο πληροφοριών.

Button

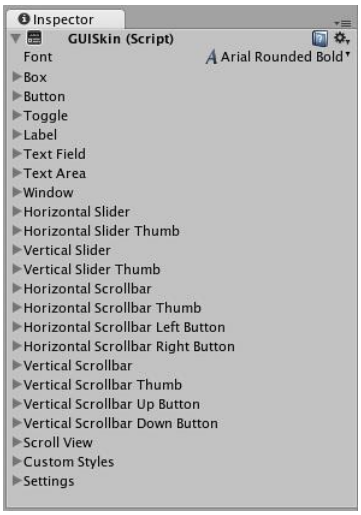
Το Button είναι ένα τυπικό διαδραστικό πλήκτρο. Θα ανταποκριθεί όταν επιλεχθεί, και δεν έχει σημασία πόσο χρόνο παραμένει το ποντίκι σε αυτό. Η αντίδρασή του εμφανίζεται μόλις το κουμπί του ποντικιού απελευθερώνεται. Θα επιστρέψει true όταν επιλεχθεί, και στον κώδικα χρησιμοποιούμαι την κλάση GUI.Button για να το χειριστούμε.

RepeatButton

Το RepeatButton είναι μια παραλλαγή του κανονικού κουμπιού. Η διαφορά είναι ότι το RepeatButton ανταποκρίνεται όσο το ποντίκι παραμένει πατημένο σε αυτό. Επιστρέφει true για κάθε frame που είναι επιλεγμένο και στον κώδικα χρησιμοποιούμε την κλάση GUI.RepeatButton.

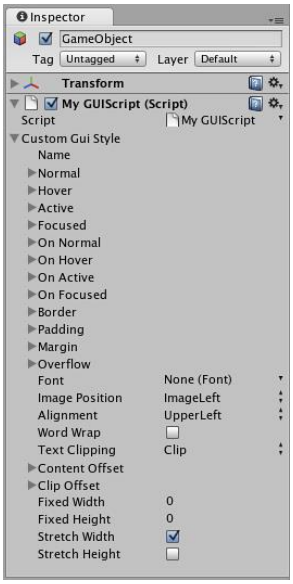
GUISkin

Τα GUISkins είναι μια συλλογή από GUIStyles που μπορούν να εφαρμοστούν στο GUI. Ο κάθε τύπος ελέγχου έχει τον δικό του ορισμό του στυλ. Με τα Skins εφαρμόζουμε στυλ σε ένα ολόκληρο UI, και αποτελούν μέρος του συστήματος UnityGUI.



Όταν δημιουργούμε ένα ολόκληρο GUI για το παιχνίδι, θα χρειαστεί πιθανότατα να κάνουμε παραμετροποίηση για κάθε διαφορετικό τύπο ελέγχου. Σε πολλά διαφορετικά είδη παιχνιδιών, όπως στρατηγικής σε πραγματικό χρόνο ή ρόλων, αυτό πρέπει να γίνεται για σχεδόν κάθε είδος ελέγχου. Επειδή κάθε Control χρησιμοποιεί ένα συγκεκριμένο στυλ, δεν έχει νόημα να δημιουργήσουμε πολλά διαφορετικά Styles. Με τη δημιουργία των GUI Skins έχουμε μια προκαθορισμένη συλλογή από Styles για το κάθε Control. Μπορούμε να εφαρμόσουμε το Skin με κώδικα, έτσι ώστε να καθορίζουμε το Style του κάθε Control αυτόματα.

GUI Style

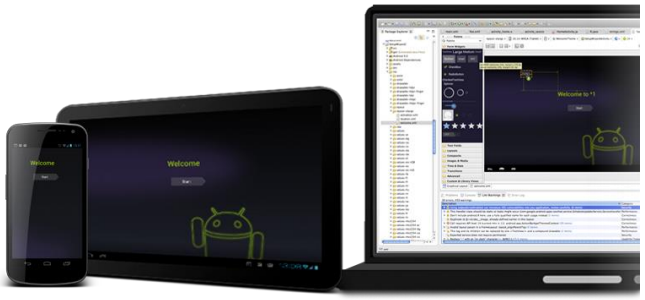


Το στυλ GUI είναι μια συλλογή από προσαρμοσμένα χαρακτηριστικά για τη χρήση του UnityGUI. Ένα GUI Style καθορίζει την εμφάνιση ενός UnityGUI Control. Αν θέλουμε να προσθέσουμε το στυλ σε περισσότερους από έναν έλεγχους, χρησιμοποιούμε ένα GUI Skin αντί για ένα GUI Style.

Τα GUIStyles δηλώνονται και τροποποιούνται μέσω κώδικα. Αν θέλουμε να χρησιμοποιήσουμε μόνο ένα ή μερικά Controls με ένα προσαρμοσμένο στυλ, μπορούμε να δηλώσουμε το Style στο κώδικα ως argument της συνάρτησης Control και έτσι τα στοιχεία ελέγχου θα εμφανίζονται με το στυλ που ορίσαμε.

3.4 Developer Tools

Το Android Developer Tools (ADT) plugin για το Eclipse παρέχει επαγγελματικού επιπέδου περιβάλλον ανάπτυξης για την οικοδόμηση Android apps. Είναι ένα πλήρες Java IDE με προηγμένα χαρακτηριστικά για να μας βοηθήσει στη δημιουργία, δοκιμή, αποσφαλμάτωση και τη συσκευασία των Android εφαρμογών μας. Είναι δωρεάν, ανοιχτού κώδικα, και τρέχει στις περισσότερες μεγάλες πλατφόρμες OS. Για να ξεκινήσουμε την ανάπτυξη μιας Android εφαρμογής, κατεβάζουμε το Android SDK.



3.4.1 Full Java IDE

- Android-specific refactoring, γρήγορες λύσεις, ενσωματωμένη πλοήγηση μεταξύ Java και των XML πόρων.
- Ενισχυμένοι XML συντάκτες για τους Android XML resources.
- Εργαλεία στατικής ανάλυσης για να εντοπίζει την απόδοση, τη χρηστικότητα, και τα προβλήματα ορθότητας.
- Κατασκευάστηκε υποστήριξη σύνθετων έργων, υποστήριξη της γραμμής εντολών για CI μέσω Ant. Περιλαμβάνει ProGuard και app-signing.
- Template-based wizard για τη δημιουργία πρότυπων έργων Android και των συστατικών.

3.4.2 Graphical UI Builders

- Κατασκευάστηκαν πλούσια Android UI με drag and drop.
- Οπτικοποίηση των UI για tablets, τηλέφωνα και άλλες συσκευές. Εναλλαγή θεμάτων, τοπικών ρυθμίσεων, ακόμη και εκδόσεων πλατφόρμας αμέσως, χωρίς κατασκευή.
- Visual refactoring μας επιτρέπει να εξάγουμε διάταξη για την ένταξη, μετατρέπει σχεδιαγράμματα, εξάγει στυλ.
- Υποστήριξη Editor για την εργασία με custom UI components.

3.4.3 On-device Developer Options

- Ενεργοποίηση εντοπισμού σφαλμάτων μέσω USB.
- Γρήγορη καταγραφή αναφορών σφαλμάτων στη συσκευή.
- Εμφάνιση χρήση της CPU στην οθόνη.
- Σχεδιασμός πληροφοριών εντοπισμού σφαλμάτων που εμφανίζονται στην οθόνη, όπως όρια διάταξης, ενημερώσεις σχετικά με τις GPU όψεις και τα στρώματα του hardware, καθώς και άλλες πληροφορίες.
- Προσθήκη πολλών περισσότερων επιλογών για την προσομοίωση app stresses ή την ενεργοποίηση των επιλογών debugging.

Απόκτηση πρόσβασης σε αυτές τις ρυθμίσεις έχουμε ανοίγοντας τις επιλογές Developer στις ρυθμίσεις του συστήματος.

3.4.4 Ανάπτυξη σε συσκευές Hardware

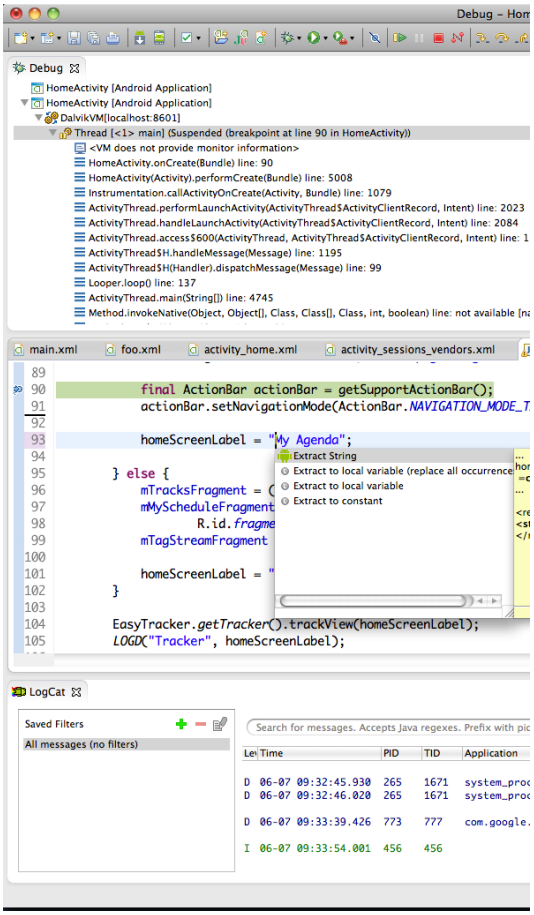
- Χρησιμοποιούμε οποιαδήποτε εμπορική Android hardware συσκευή ή πολλαπλές συσκευές.
- Ανάπτυξη της εφαρμογής μας για τις συνδεδεμένες συσκευές απευθείας από τον IDE.
- Ζωντανά, εντοπισμός σφαλμάτων σε συσκευή, έλεγχος, και profiling.

3.4.5 Ανάπτυξη για Virtual Συσκευές

- Εξομοίωση οποιασδήποτε συσκευής. Χρησιμοποιούμε προσαρμοσμένα μεγέθη οθόνης, πληκτρολόγια και άλλα μηχανικά εξαρτήματα.
- Advanced hardware emulation, συμπεριλαμβανομένων κάμερα, αισθητήρες, multitouch, τηλεφωνία.
- Ανάπτυξη και δοκιμή για ευρεία συμβατότητα της συσκευής.

3.4.6 Powerful Debugging

- Πλήρες πρόγραμμα εντοπισμού σφαλμάτων Java με on-device debugging και ειδικά Android εργαλεία.
- Ενσωματωμένη μνήμη ανάλυσης, απόδοση/CPU προφίλ, OpenGL ES εντοπισμό.
- Γραφικά εργαλεία για τον εντοπισμό σφαλμάτων και βελτιστοποίηση UI, runtime inspecton της δομής UI και των επιδόσεων.
- Runtime γραφική ανάλυση της χρήσης του εύρους ζώνης (bandwidth) του δικτύου της εφαρμογής μας.



3.4.7 Testing

- Πλήρως instrumentated, scriptable περιβάλλον δοκιμής.
- Ολοκληρωμένες εκθέσεις με τη χρήση τυποποιημένων δοκιμών UI.
- Δημιουργία και εκτέλεση δοκιμών μονάδας σε συσκευές hardware ή εξομοιωτή (emulator).

3.4.8 Native Development

- Υποστήριξη για compiling και packaging του υφιστάμενου κώδικα γραμμένο σε C ή C++.
- Υποστήριξη για packaging πολλών αρχιτεκτονικών σε ένα ενιαίο δυαδικό, για ευρεία συμβατότητα.

3.5 Android SDK

Το Android SDK παρέχει τις βιβλιοθήκες API και εργαλεία που είναι απαραίτητες για την κατασκευή, τη δοκιμή και τον εντοπισμό σφαλμάτων για τις εφαρμογές Android.

Η ADT Bundle περιλαμβάνει τα βασικά συστατικά του Android SDK για την ανάπτυξη εφαρμογών, συμπεριλαμβανομένης μια έκδοση του Eclipse IDE με ενσωματωμένο ADT (Android Developer Tools) για την ανάπτυξη της Android εφαρμογής: Eclipse + ADT plugin, Android SDK Tools, Android Platform-tools, την πιο πρόσφατη πλατφόρμα Android, και την τελευταία Android εικόνα του συστήματος για τον εξομοιωτή.

Χρειάζεται συχνά να εγκατασταθούν πρόσθετες εκδόσεις του Android για τον εξομοιωτή και άλλα πακέτα όπως η βιβλιοθήκη για το Google Play In-app Billing. Για την εγκατάσταση των πακέτων χρησιμοποιείται ο SDK Manager.

Μπορεί να χρησιμοποιηθεί μια υπάρχουσα έκδοση του Eclipse ή άλλο IDE για την ανάπτυξη του Android app, χρησιμοποιώντας τα SDK Tools και εγκαθιστώντας πακέτα Android SDK (όπως η πλατφόρμα Android και την εικόνα του συστήματος). Στην υπάρχουσα έκδοση του Eclipse μπορεί να προσθεθεί το ADT plugin.

Το πακέτο SDK Tools δεν είναι το πλήρες περιβάλλον SDK. Περιλαμβάνει μόνο τα βασικά εργαλεία SDK, τα οποία μπορούν να χρησιμοποιηθούν για να κατεβάσουμε τα υπόλοιπα πακέτα SDK (όπως την τελευταία εικόνα του συστήματος).

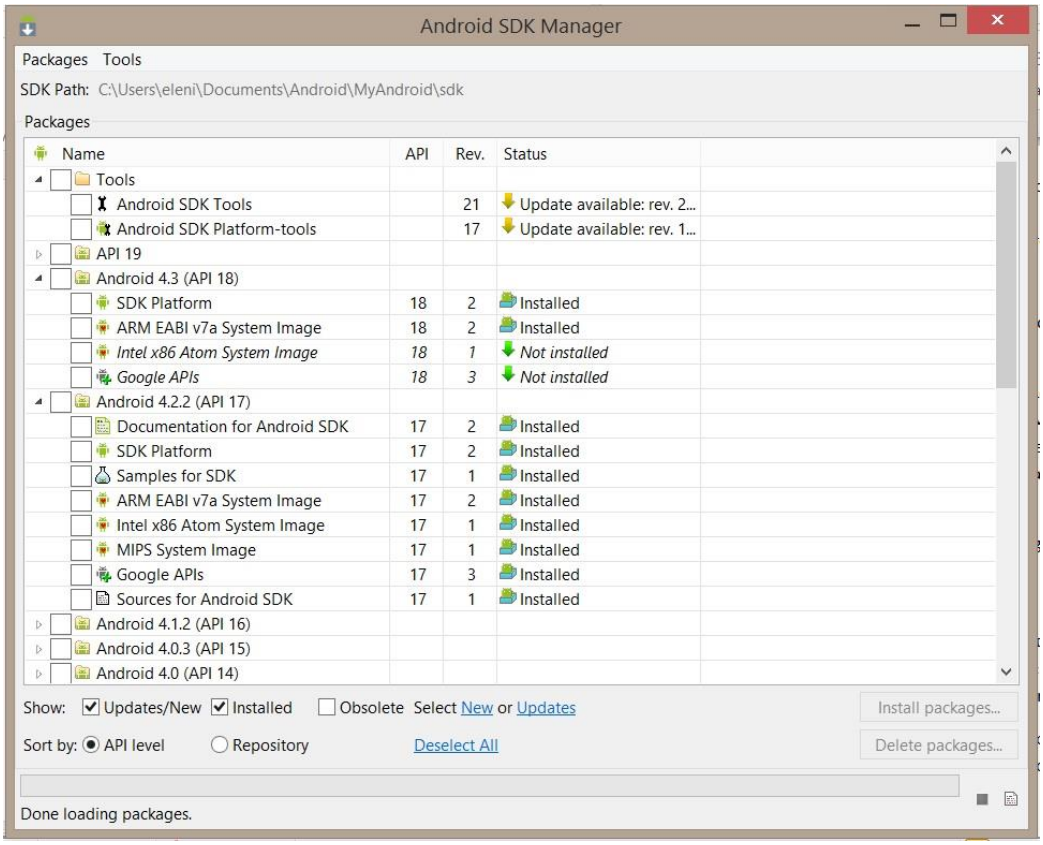
Το πακέτο λήψης είναι ένα εκτελέσιμο αρχείο που ξεκινά ένα πρόγραμμα εγκατάστασης. Το πρόγραμμα εγκατάστασης ελέγχει τον υπολογιστή μας για τα απαιτούμενα εργαλεία, όπως η ορθή Java SE Development Kit (JDK) και την εγκαθιστά, αν είναι απαραίτητο. Το πρόγραμμα εγκατάστασης αποθηκεύει τα Android SDK Tools σε μια προεπιλεγμένη θέση (ή μπορούμε να καθορίσουμε τη θέση). Τη θέση αυτή πρέπει να τη συνδέσουμε με την Unity για να μπορέσει να μετατραπεί το παιχνίδι μας σε εφαρμογή.

Το Android προσφέρει μια προσαρμοσμένη plugin για το Eclipse IDE, που ονομάζεται Android Development Tools (ADT). Αυτό το plugin παρέχει ένα ισχυρό, ολοκληρωμένο περιβάλλον για την ανάπτυξη των Android apps. Επεκτείνει τις δυνατότητες του Eclipse για να μας αφήσει να ρυθμίσουμε γρήγορα νέα Android projects, την οικοδόμηση μιας UI app, την εκσφαλμάτωση (debug) της app μας, και την εξαγωγή πακέτων app (apks) για τη διανομή.

Εμείς χρησιμοποιήσαμε το MonoDevelop της Unity οπότε δεν χρειάστηκε να εγκαταστήσουμε το Eclipse, αλλά χρησιμοποιήσαμε άμεσα τα εργαλεία SDK για την κατασκευή και τον εντοπισμό σφαλμάτων της εφαρμογής μας. Αφού ρυθμίσουμε το ADT plugin, πρέπει να προσθέσουμε τα πιο σύγχρονα εργαλεία της πλατφόρμας SDK και μια Android πλατφόρμα για το περιβάλλον μας.

3.5.1 SDK Manager

Το Android SDK χωρίζει εργαλεία, πλατφόρμες, και άλλα συστατικά σε πακέτα που μπορούμε να κατεβάσουμε χρησιμοποιώντας το Android SDK Manager. Το αρχικό πακέτο SDK που κατεβάζουμε εξαρχής περιλαμβάνει μόνο τα εργαλεία SDK. Για την ανάπτυξη μιας Android εφαρμογής, θα πρέπει επίσης να κατεβάσουμε τουλάχιστον μία Android πλατφόρμα και τα τελευταία SDK Platform-tools.



Ο SDK Manager εμφανίζει όλα τα διαθέσιμα πακέτα SDK για να τα προσθέσουμε στο Android SDK μας. Δείχνει τα πακέτα SDK που είναι διαθέσιμα, που έχουν ήδη εγκατασταθεί, ή για τα οποία είναι διαθέσιμη μια ενημέρωση. Πρέπει να εγκαταστήσουμε τα τελευταία πακέτα Tools, την τελευταία έκδοση του Android, και τη Βιβλιοθήκη Υποστήριξης Android (Android Support Library). Επιλέγοντας τα πακέτα ο Android SDK Manager τα εγκαθιστά στο περιβάλλον Android SDK. Με τα πακέτα αυτά εγκατεστημένα, είμαστε έτοιμοι να ξεκινήσουμε να αναπτύξουμε την Android εφαρμογή.

Το Android SDK αποτελείται από αρθρωτά πακέτα που μπορούμε να κατεβάσουμε ξεχωριστά χρησιμοποιώντας το Android SDK Manager. Όταν έχει κυκλοφορήσει μια ενημέρωση ή μια νέα έκδοση της πλατφόρμας Android για κάποιο πακέτο, μπορούμε να χρησιμοποιήσετε τον SDK Manager για να τα κατεβάσουμε γρήγορα στο περιβάλλον μας.

Υπάρχουν πολλά διαφορετικά διαθέσιμα πακέτα για το Android SDK. Παρακάτω περιγράφουμε τα περισσότερα από τα διαθέσιμα:

- **SDK Tools**: Υποχρεωτικό. Περιέχει εργαλεία για τον εντοπισμό σφαλμάτων και τις δοκιμές (debugging και testing), καθώς και άλλα βοηθητικά προγράμματα που απαιτούνται για την ανάπτυξη μιας εφαρμογής. Εάν έχουμε εγκαταστήσει το starter πακέτο SDK, τότε έχουμε ήδη εγκαταστήσει την τελευταία έκδοση αυτού του πακέτου. Πρέπει να το διατηρούμε ενημερωμένο.
- **SDK Platform-tools**: Υποχρεωτικό. Περιέχει platform-dependent tools για την ανάπτυξη και τον εντοπισμό σφαλμάτων της εφαρμογής μας. Αυτά τα εργαλεία υποστηρίζουν τις πιο πρόσφατες δυνατότητες της πλατφόρμας Android και συνήθως ενημερώνεται μόνο όταν μια νέα πλατφόρμα είναι διαθέσιμη. Τα εργαλεία αυτά είναι πάντα συμβατά με παλαιότερες πλατφόρμες, αλλά θα πρέπει να βεβαιωθούμε ότι έχουμε την πιο πρόσφατη έκδοση αυτών των εργαλείων, όταν εγκαθιστάμε μια νέα πλατφόρμα SDK. Πρέπει να εγκαταστήσουμε αυτό το πακέτο κατά την εγκατάσταση του SDK την πρώτη φορά.
- Documentation: Ένα offline αντίγραφο της τελευταίας τεκμηρίωσης για το Android APIs της πλατφόρμας.
- **SDK Platform**: Υποχρεωτικό. Υπάρχει διαθέσιμο ένα Platform SDK για κάθε έκδοση του Android. Περιλαμβάνει ένα android.jar αρχείο πλήρως συμβατό με Android βιβλιοθήκη. Για να φτιάξουμε μια Android εφαρμογή, πρέπει κατεβάσουμε τουλάχιστον μία πλατφόρμα SDK στο περιβάλλον μας για να μπορούμε να κάνουμε compile και να την ορίσουμε ως στόχο της κατασκευής μας. Προκειμένου να παρέχει την καλύτερη εμπειρία του χρήστη σχετικά με τις τελευταίες συσκευές, είναι καλύτερο να χρησιμοποιούμε την πιο πρόσφατη έκδοση της πλατφόρμας ως στόχο της κατασκευή μας. Θα εξακολουθεί να είναι σε θέση να τρέξει την εφαρμογή μας σε παλαιότερες εκδόσεις, αλλά θα πρέπει να την φτιάξουμε στην τελευταία έκδοση προκειμένου να χρησιμοποιήσει τις νέες δυνατότητες όταν τρέχει σε συσκευές με την τελευταία έκδοση του Android.
- System Images: Συνιστάται. Αν και μπορεί να έχουμε μία ή περισσότερες συσκευές με λογισμικό Android στην οποία θα δοκιμαστεί η εφαρμογή μας, είναι απίθανο να έχουμε μια συσκευή για κάθε έκδοση Android που υποστηρίζει η εφαρμογή μας. Είναι μια καλή πρακτική για να κατεβάσουμε τις εικόνες του συστήματος για όλες τις εκδόσεις του Android που υποστηρίζει η εφαρμογή μας και να δοκιμάσουμε την εφαρμογή μας να τρέχει σε αυτά με το Android emulator. Κάθε έκδοση της πλατφόρμας προσφέρει μία ή περισσότερες διαφορετικές εικόνες του συστήματος (όπως για ARM και x86). Το Android emulator απαιτεί ένα σύστημα εικόνας για να λειτουργήσει. Πρέπει πάντα να δοκιμάζουμε την εφαρμογή μας με την τελευταία έκδοση του Android και η χρήση του εξομοιωτή με το πιο πρόσφατο system image είναι ένας καλός τρόπος για να το πετύχουμε.

- Sources for Android SDK: Ένα αντίγραφο της πλατφόρμας Android πηγαίου κώδικα που είναι χρήσιμα για την ενίσχυση μέσω του κώδικα κατά τον εντοπισμό σφαλμάτων της εφαρμογής μας.
- Samples for SDK: Συνιστάται. Μια συλλογή δειγμάτων εφαρμογών που επιδεικνύουν μια ποικιλία των APIs της πλατφόρμας. Αυτά είναι ένας μεγάλος πόρος για να περιηγηθούμε κώδικα της Android app. Η εφαρμογή API Demos παρέχει έναν τεράστιο αριθμό μικρών demos. Τα δείγματα δίνουν τον πηγαίο κώδικα που μπορούμε να τον χρησιμοποιήσουμε για να μάθουμε το Android, το φορτώνουμε ως ένα έργο και το τρέχουμε, ή το επαναχρησιμοποιούμε στη δική μας εφαρμογή. Πολλαπλά δείγματα πακέτων είναι διαθέσιμα - ένα για κάθε έκδοση Android της πλατφόρμας. Όταν επιλέγουμε ένα πακέτο δείγμα για να κατεβάσουμε, επιλέγουμε εκείνο του οποίου η API Level ταιριάζει με το επίπεδο API της πλατφόρμας Android που σκοπεύουμε να χρησιμοποιήσουμε.
- Google APIs: Μια SDK add-on που παρέχει μια πλατφόρμα που μπορούμε να χρησιμοποιήσουμε για να αναπτύξουμε μια εφαρμογή με τη χρήση ειδικών Google APIs και ένα system image για τον εξομοιωτή ώστε να μπορούμε να δοκιμάσουμε την εφαρμογή μας χρησιμοποιώντας τα APIs της Google.
- Android Support: Συνιστάται. Περιλαμβάνει μια στατική βιβλιοθήκη που μας επιτρέπει να χρησιμοποιήσουμε και να συμπεριλάβουμε σε πηγές της εφαρμογής μας, ορισμένες από τις τελευταίες και ισχυρές Android APIs (όπως fragments, καθώς και άλλες που δεν περιλαμβάνονται στο πλαίσιο καθόλου και δεν είναι διαθέσιμα στη βασική πλατφόρμα) για συσκευές που τρέχουν μια έκδοση της πλατφόρμας τόσο παλιά όσο και το Android 1.6. Για παράδειγμα, η βιβλιοθήκη υποστήριξης περιέχει εκδόσεις της κλάσης Fragment που είναι συμβατή με Android 1.6 και άνω (η κατηγορία εισήχθη αρχικά σε Android 3.0) και τα APIs ViewPager που μας επιτρέπουν να δημιουργήσουμε εύκολα ένα side-swipeable UI. Όλα τα διαθέσιμα πρότυπα δραστηριότητας όταν δημιουργούμε ένα νέο έργο με την ADT Plugin το απαιτούν.
- Google Play Billing: Παρέχει τις στατικές βιβλιοθήκες και τα δείγματα που μας επιτρέπουν να ενσωματώσουμε τις υπηρεσίες τιμολόγησης στην εφαρμογή μας με το Google Play.
- Google Play Licensing: Παρέχει τις στατικές βιβλιοθήκες και τα δείγματα που μας επιτρέπουν να εκτελέσουμε επαλήθευση της άδειας για την εφαρμογή μας κατά τη διανομή με το Google Play.

Τα παραπάνω πακέτα δεν είναι πλήρη και μπορούμε να προσθέσουμε νέες τοποθεσίες για να κατεβάσουμε επιπλέον πακέτα από τρίτους.

Σε ορισμένες περιπτώσεις, ένα πακέτο SDK μπορεί να απαιτήσει μια συγκεκριμένη ελάχιστη αναθεώρηση ενός άλλου εργαλείου πακέτου ή SDK. Για παράδειγμα, μπορεί να υπάρχει εξάρτηση μεταξύ του ADT Plugin για το Eclipse και του πακέτου SDK Tools. Όταν εγκαταστήσουμε το πακέτο SDK Tools, θα πρέπει επίσης να αναβαθμίσουμε την απαιτούμενη

έκδοση του ADT (αν χρησιμοποιούμε το Eclipse). Σε αυτή την περίπτωση, ο μεγαλύτερος αριθμός έκδοσης για το ADT plugin μας θα πρέπει να ταιριάζει πάντα τον αριθμό αναθεώρησης των εργαλείων SDK μας (για παράδειγμα, ADT 8.x απαιτεί SDK Tools r8).

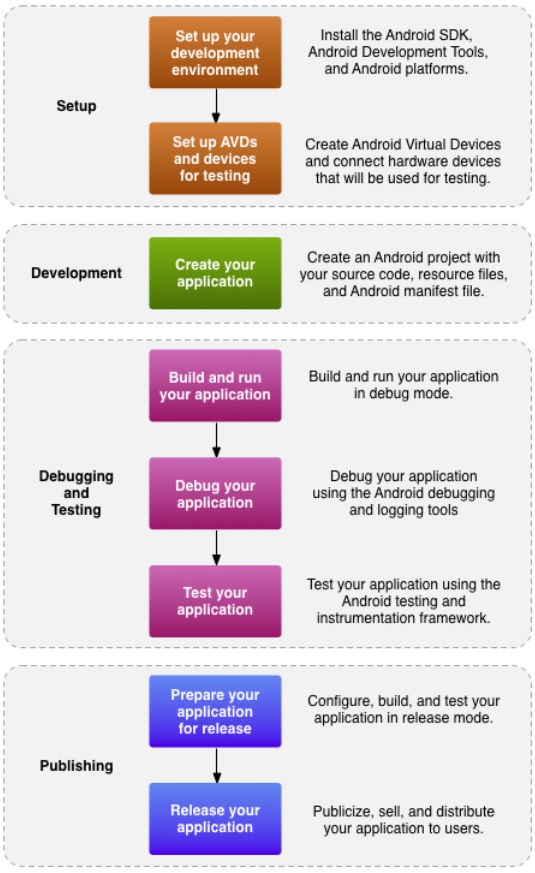
Τα εργαλεία ανάπτυξης ειδοποιούν με debug προειδοποιήσεις αν υπάρχει εξάρτηση, η οποία θα πρέπει να αντιμετωπιστεί. Ο Android SDK Manager επιβάλλει επίσης εξαρτήσεις απαιτώντας να κατεβάσουμε όλα τα πακέτα που είναι απαραίτητα από εκείνα που έχουμε επιλέξει.

3.5.2 Workflow

Για την ανάπτυξη εφαρμογών για τις Android συσκευές, μπορούμε να χρησιμοποιήσουμε μια σειρά από εργαλεία που περιλαμβάνονται στο Android SDK. Κατεβάζοντας και εγκαθιστώντας το SDK, μπορούμε να αποκτήσουμε πρόσβαση σε αυτά τα εργαλεία από το Eclipse IDE, μέσα από το plugin ADT, ή από τη γραμμή εντολών. Η ανάπτυξη με το Eclipse είναι η προτιμώμενη μέθοδος, επειδή μπορούμε να επικαλέσουμε απευθείας τα εργαλεία που χρειαζόμαστε κατά την ανάπτυξη εφαρμογών.

Ωστόσο, μπορούμε να επιλέξουμε να αναπτύξουμε με άλλο IDE ή έναν απλό επεξεργαστή κειμένου και να επικαλούνται τα εργαλεία της γραμμής εντολών ή scripts. Αυτός είναι ένας λιγότερο ορθολογικός τρόπος, επειδή θα πρέπει μερικές φορές να καλέσουμε τα εργαλεία της γραμμής εντολών με το χέρι, αλλά θα έχουμε πρόσβαση στον ίδιο αριθμό χαρακτηριστικών που θα είχαμε στο Eclipse.

Τα βασικά βήματα για την ανάπτυξη εφαρμογών (με ή χωρίς Eclipse) φαίνεται στην εικόνα που ακολουθεί. Τα στάδια ανάπτυξης είναι τέσσερα, τα οποία περιλαμβάνουν:



Εγκατάσταση

Κατά τη διάρκεια αυτής της φάσης μπορούμε να εγκαταστήσουμε και να δημιουργήσουμε το περιβάλλον ανάπτυξης. Μπορούμε επίσης να δημιουργήσουμε το Android Virtual Devices (AVDs) και να συνδέσουμε τις συσκευές hardware στις οποίες μπορούμε να εγκαταστήσουμε τις εφαρμογές μας.

Development

Κατά τη διάρκεια αυτής της φάσης δημιουργούμε και αναπτύσσουμε το Android project μας, το οποίο περιέχει το σύνολο του πηγαίου κώδικα και τα αρχεία των πόρων για την εφαρμογή μας.

Debugging and Testing

Κατά τη διάρκεια αυτής της φάσης μπορούμε να δημιουργήσουμε το πρόγραμμά μας σε ένα πακέτο debuggable .apk που μπορούμε να εγκαταστήσουμε και να εκτελέσουμε τον εξομοιωτή ή μια Android-powered συσκευή. Εάν χρησιμοποιούμε Eclipse, δημιουργούνται builds κάθε φορά που αποθηκεύεται το project. Εάν χρησιμοποιούμε κάποιο άλλο IDE, μπορούμε να δημιουργήσουμε το project μας χρησιμοποιώντας Ant και να το εγκαταστήσουμε σε μια συσκευή που χρησιμοποιεί adb.

Στη συνέχεια, μπορούμε να διορθώσουμε την εφαρμογή μας χρησιμοποιώντας ένα JDWP συμβατό πρόγραμμα εντοπισμού σφαλμάτων μαζί με τα εργαλεία εντοπισμού σφαλμάτων και καταγραφής που παρέχονται με το Android SDK. Το Eclipse διαθέτει ένα συμβατό πρόγραμμα εντοπισμού σφαλμάτων.

Τέλος, δοκιμάζουμε την εφαρμογή μας χρησιμοποιώντας διάφορα εργαλεία ελέγχου του Android SDK.

Publishing

Κατά τη διάρκεια αυτής της φάσης μπορούμε να ρυθμίσουμε και να δημιουργήσουμε την αίτησή μας για την απελευθέρωση και διανομή της εφαρμογής μας για τους χρήστες.

3.5.3 Managing Virtual Devices

Μια Android Virtual Device (AVD) είναι μια διαμόρφωση του εξομοιωτή που μας επιτρέπει να μοντελοποιήσουμε μια πραγματική συσκευή ορίζοντας τις hardware και software επιλογές που μπορούν να προσομοιωθούν από το Android Emulator.

Ο ευκολότερος τρόπος για να δημιουργήσουμε μια AVD είναι να χρησιμοποιήσουμε τη γραφική AVD Manager, το οποίο θα ξεκινήσει από το Eclipse (Window> AVD Manager). Μπορούμε επίσης να ξεκινήσουμε τον AVD Manager από τη γραμμή εντολών με την κλήση του android εργαλείου με τις AVD επιλογές, από την <sdk>/tools/ directory. Μπορούμε ακόμη να δημιουργήσουμε AVDs στη γραμμή εντολών με το πέρασμα των εργαλείων των android επιλογών.

Μία AVD αποτελείται από:

- Ένα hardware profile: Καθορίζει τα χαρακτηριστικά του hardware της εικονικής συσκευής. Για παράδειγμα, μπορούμε να ορίσουμε αν η συσκευή διαθέτει φωτογραφική μηχανή, είτε χρησιμοποιεί ένα φυσικό πληκτρολόγιο QWERTY ή ένα πληκτρολόγιο κλήσης, πόση μνήμη έχει, και ούτω καθεξής.
- Μια χαρτογράφηση σε ένα system image: Μπορούμε να καθορίσουμε ποια έκδοση της πλατφόρμας Android θα τρέξει στην εικονική συσκευή. Μπορούμε να επιλέξουμε μια

έκδοση της standard Android πλατφόρμας ή της εικόνας του συστήματος μαζί με μια SDK add-on.

- Άλλες επιλογές: Μπορούμε να καθορίσουμε το δέρμα εξομοιωτή που θέλουμε να χρησιμοποιήσουμε με την AVD, το οποίο μας επιτρέπει να ελέγχουμε τις διαστάσεις της οθόνης, την εμφάνιση, και ούτω καθεξής. Μπορουμε επίσης να καθορίσουμε την προσομοίωση της κάρτας SD για χρήση με την AVD.
- Ένας ειδικός χώρος αποθήκευσης για την μηχανή ανάπτυξής μας: τα δεδομένα του χρήστη της συσκευής (εγκατεστημένες εφαρμογές, ρυθμίσεις, και ούτω καθεξής) και κάρτα SD προσομοίωσης αποθηκεύονται σε αυτόν τον τομέα.

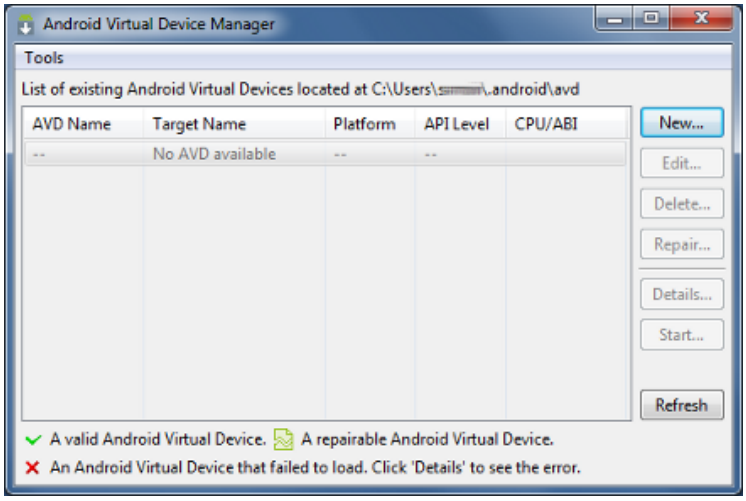
Μπορούμε να δημιουργήσουμε όσες AVDs χρειαζόμαστε, με βάση τους τύπους της συσκευής που θέλουμε να μοντελοποιήσουμε. Για να ελέγξουμε προσεκτικά την εφαρμογή μας, θα πρέπει να δημιουργήσουμε μία AVD για κάθε γενική διαμόρφωση της συσκευής (για παράδειγμα, σε διαφορετικά μεγέθη οθόνης και εκδόσεις της πλατφόρμας), με τα οποία η εφαρμογή μας είναι συμβατή και να ελέγξουμε την εφαρμογή για την κάθε ένα.

Όταν διαλέγουμε ένα system image στόχο για την AVD μας πρέπει να λαμβάνουμε υπόψιν μας:

- Το επίπεδο API του στόχου είναι σημαντικό, επειδή η εφαρμογή μας δεν θα είναι σε θέση να τρέξει σε ένα σύστημα του οποίου η εικόνα API Level είναι μικρότερη από αυτή που απαιτείται από την εφαρμογή μας, όπως ορίζεται στο minSdkVersion χαρακτηριστικό στο αρχείο δήλωσης της εφαρμογής.
- Θα πρέπει να δημιουργήσουμε τουλάχιστον μία AVD που χρησιμοποιεί έναν στόχο του οποίου το API Level είναι μεγαλύτερο από αυτό που απαιτείται από την εφαρμογή μας, επειδή μας επιτρέπει να ελέγξουμε την εμπρός συμβατότητα της εφαρμογής μας. Δοκιμή της forward-compatibility εξασφαλίζει ότι, όταν οι χρήστες που έχουν κατεβάσει την εφαρμογή μας λάβουν μια ενημερωμένη έκδοση του συστήματος, η εφαρμογή θα συνεχίσει να λειτουργούν κανονικά.
- Εάν η εφαρμογή μας δηλώνει ένα στοιχείο χρήσης - βιβλιοθήκης στο αρχείο δήλωσης της, η εφαρμογή μπορεί να τρέξει μόνο σε μια εικόνα του συστήματος στην οποία η εξωτερική βιβλιοθήκη είναι παρούσα. Εάν θέλουμε να εκτελέσουμε την εφαρμογή μας σε έναν εξομοιωτή, δημιουργούμε μία AVD που περιλαμβάνει την απαιτούμενη βιβλιοθήκη. Συνήθως, θα πρέπει να δημιουργήσουμε μια τέτοια AVD χρησιμοποιώντας ένα Add-on συστατικό για την πλατφόρμα της AVD (για παράδειγμα, τα APIs Google Add-on περιέχουν τη βιβλιοθήκη του Google Maps).

3.5.4 Managing AVDs με AVD Manager

Ο AVD Manager είναι ένα εύκολο στη χρήση περιβάλλον εργασίας (user interface) για τη διαχείριση των AVD (Android Virtual Device) διαμορφώσεων. Μια AVD είναι μια διαμόρφωση της συσκευής για το Android emulator που μας επιτρέπει να μοντελοποιήσουμε διαφορετικές διαμορφώσεις των Android-powered συσκευές. Όταν ξεκινάμε τον AVD Manager στο Eclipse ή πλοηγηθούμε στο SDK's tools/directory και εκτελέσουμε το Android avd, θα δούμε τον AVD Manager όπως φαίνεται στην εικόνα.

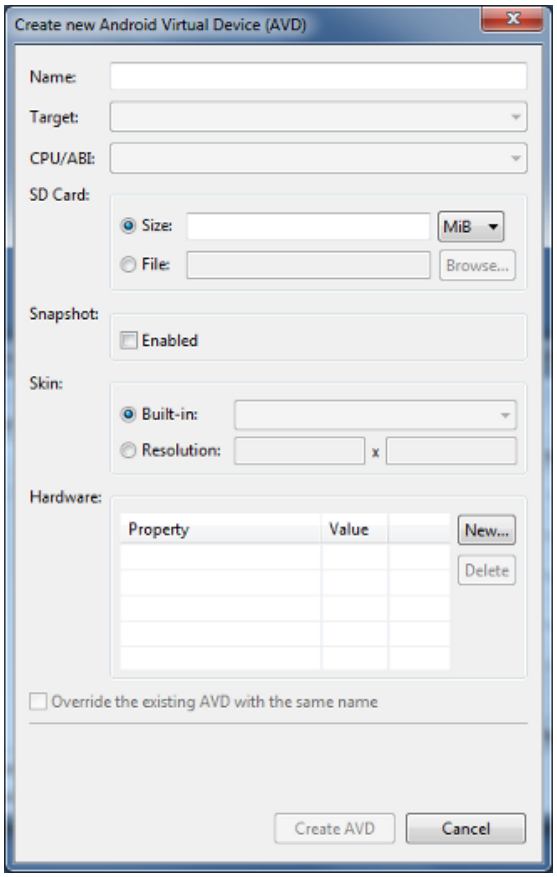


Από την κύρια οθόνη, μπορούμε να δημιουργήσουμε, να διαγράψουμε, να διορθώσουμε και να αρχίσουμε τις AVDs, καθώς και να δούμε τις λεπτομέρειες της κάθε AVD.

Δημιουργώντας μία AVD

Μπορούμε να δημιουργήσουμε όσες AVDs θέλουμε να δοκιμάσουμε. Συνιστάται να ελέγξουμε τις εφαρμογές μας σε όλα τα επίπεδα API υψηλότερα από το επίπεδο-στόχο API για την εφαρμογή μας.

Στον πίνακα Virtual Devices, υπάρχει μια λίστα με τις υπάρχουσες AVDs, από τις οποίες μπορούμε να διαλέξουμε μία ή να δημιουργήσουμε μία νέα AVD. Συμπληρώνουμε τα στοιχεία για την AVD. Δίνουμε ένα όνομα, ένα στόχο πλατφόρμας, ένα μέγεθος κάρτας SD, και το δέρμα (HVGA είναι προεπιλογή). Μπορούμε επίσης να προσθέσουμε ειδικά χαρακτηριστικά του hardware της προσομοιωμένης συσκευής επιλέγοντας τα χαρακτηριστικά. Παρακάτω αναφέρεται μια λίστα με τα χαρακτηριστικά του hardware. Πρέπει να καθορίσουμε ένα στόχο για την AVD που ικανοποιεί το Build Target της εφαρμογής μας (ο στόχος της πλατφόρμας AVD πρέπει να έχει ένα επίπεδο API ίσο ή μεγαλύτερο από το επίπεδο API για το οποίο φτιάχνεται η εφαρμογής μας). Όταν η AVD μας είναι πλέον έτοιμη, μπορούμε να ξεκινήσουμε τον εξομοιωτή με την AVD επιλέγοντας μία συσκευή.



Εμείς για να δοκιμάσουμε το παιχνίδι μας, φτιάξαμε αρκετές διαφορετικές AVD προσομοιώνοντας διαφορετικές συσκευές σε διαφορετικές εκδόσεις πλατφόρμων Android, διαφορετικό hardware, και διαφορετικό μέγεθος οθόνης. Το παιχνίδι αναγνωρίζεται μέχρι και την 4.3 έκδοση Android. Στην καινούργια έκδοση 4.4 Kit Kat δεν υπήρχε ακόμη συμβατότητα από τη Unity.

Εκτός από τη δημιουργία AVDs με τη διεπαφή χρήστη του AVD Manager, μπορούμε επίσης να τις δημιουργήσουμε περνώντας τα arguments στη γραμμή εντολών στο android εργαλείο. Ανοίγουμε ένα terminal window και αλλάζουμε το <sdk>/tools/ directory, αν χρειάζεται. Για τη δημιουργία της κάθε AVD, δίνουμε την εντολή `android create avd`, με επιλογές που να καθορίζουν ένα όνομα για τη νέα AVD και την εικόνα του συστήματος που θέλουμε να εκτελείται στον εξομοιωτή, όταν χρησιμοποιείται η AVD. Μπορούμε επίσης να καθορίσουμε άλλες επιλογές στη γραμμή εντολών, όπως το μέγεθος της προσομοιωμένης κάρτας SD, το δέρμα του εξομοιωτή, ή μια προσαρμοσμένη θέση για τα αρχεία δεδομένων του χρήστη.

Hardware options

Κατά τη δημιουργία μιας νέας AVD, μπορούμε να καθορίσουμε τις εξής επιλογές hardware για να προσομοιώσει το AVD:

- Device ram size: Το ποσό της φυσικής μνήμης RAM της συσκευής, σε megabyte.
- Touch-screen support: Αν υπάρχει μια οθόνη αφής ή όχι στη συσκευή.
- Trackball support: Αν υπάρχει ένα trackball στη συσκευή.
- Keyboard support: Αν η συσκευή έχει ένα πληκτρολόγιο QWERTY.
- DPad support: Αν η συσκευή διαθέτει πλήκτρα Dpad.
- GSM modem support: Αν υπάρχει ένα μόντεμ GSM στη συσκευή.
- Camera support: Αν η συσκευή διαθέτει κάμερα.
- Μέγιστα οριζόντια camera pixels: Προκαθορισμένη τιμή είναι " 640 ".
- Μέγιστα κάθετα camera pixels: Προκαθορισμένη τιμή είναι " 480 ".
- GPS support: Αν υπάρχει ένα GPS στη συσκευή.
- Battery support: Αν η συσκευή μπορεί να τρέξει με μια μπαταρία.
- Accelerometer: Αν υπάρχει ένα επιταχυνσιόμετρο στη συσκευή.
- Audio recording support: Αν η συσκευή μπορεί να καταγράφει ήχο.
- Audio playback support: Αν η συσκευή μπορεί να παίξει ήχο.
- SD Card support: Αν η συσκευή υποστηρίζει την εισαγωγή/αφαίρεση της εικονικής κάρτας SD.
- Cache partition support: Αν χρησιμοποιούμε ένα χώρισμα cache στη συσκευή.
- Cache partition size: Προεπιλεγμένη τιμή είναι "66MB".
- Abstracted LCD density: Ρυθμίζει τη γενικευμένη χαρακτηριστική πυκνότητα που χρησιμοποιείται από την οθόνη του AVD του.

3.5.5 Χρησιμοποιώντας τον Emulator

Το Android SDK περιλαμβάνει μια εικονική κινητή συσκευή emulator που τρέχει στον υπολογιστή μας. Ο εξομοιωτής επιτρέπει την ανάπτυξη και δοκιμή των εφαρμογών Android χωρίς τη χρήση μιας φυσικής συσκευής.

Ο Android emulator προσομοιώνει όλα τα hardware και software χαρακτηριστικά μιας τυπικής κινητής συσκευής, εκτός από το ότι δεν μπορούμε να τοποθετήσουμε πραγματικές τηλεφωνικές κλήσεις. Παρέχει μια ποικιλία πλοήγησης και πλήκτρα ελέγχου, τα οποία μπορούμε να "πατήσουμε" χρησιμοποιώντας το ποντίκι ή το πληκτρολόγιο για να παράγει γεγονότα για την εφαρμογή μας. Παρέχει επίσης μια οθόνη στην οποία εμφανίζεται η εφαρμογή μας, μαζί με τυχόν άλλες ενεργές εφαρμογές Android.

Για να μπορέσουμε να μοντελοποιήσουμε και να ελέγξουμε την εφαρμογή μας πιο εύκολα, ο εξομοιωτής χρησιμοποιεί τις διαμορφώσεις της Android Virtual Device (AVD). Οι AVDs μας επιτρέπουν να ορίσουμε ορισμένες πτυχές του hardware του προσομοιωμένου τηλεφώνου μας και θα μας επιτρέψει να δημιουργήσουμε πολλές συνθέσεις για να δοκιμάσουμε πολλές πλατφόρμες Android και παραλλαγές του hardware. Μόλις η εφαρμογή μας λειτουργεί με τον εξομοιωτή, μπορεί να χρησιμοποιήσει τις υπηρεσίες της πλατφόρμας Android να επικαλεστεί άλλες εφαρμογές, πρόσβαση στο δίκτυο, αναπαραγωγή ήχου και βίντεο, να αποθηκεύει και να ανακτεί τα δεδομένα, να ειδοποιεί το χρήστη, και να καταστήσει γραφικές μεταβάσεις και θέματα.

Ο εξομοιωτής περιλαμβάνει επίσης μια ποικιλία από debug δυνατότητες, όπως μια κονσόλα από την οποία μπορούμε να συνδεθούμε στην έξοδο του πυρήνα, να προσομοιώσουμε τις διακοπές της εφαρμογής (όπως την άφιξη των μηνυμάτων SMS ή τηλεφωνικές κλήσεις) και την προσομοίωση των αποτελεσμάτων λανθάνουσας κατάστασης και απώλειες στο δίκτυο δεδομένων.

Overview

Ο Android emulator είναι μια εφαρμογή που παρέχει μια εικονική φορητή συσκευή στην οποία μπορούμε να τρέξουμε τις Android εφαρμογές μας. Τρέχει ένα πλήρες σύστημα Android στοίβας, μέχρι το επίπεδο του πυρήνα, που περιλαμβάνει ένα σύνολο προεγκατεστημένων εφαρμογών (όπως το dialer) στο οποίο μπορούμε να έχουμε πρόσβαση από τις εφαρμογές μας. Μπορούμε να επιλέξουμε ποια έκδοση του συστήματος Android θέλουμε να τρέξουμε στον εξομοιωτή με τη διαμόρφωση AVDs, και μπορούμε επίσης να προσαρμόσουμε το δέρμα κινητής συσκευής και τις key mappings. Κατά την έναρξη του εξομοιωτή και κατά το χρόνο εκτέλεσης, μπορούμε να χρησιμοποιήσουμε μια ποικιλία από εντολές και επιλογές για να ελέγξουμε τη συμπεριφορά του.

Οι εικόνες συστήματος Android που διατίθονται μέσω του Android SDK Manager περιέχουν κώδικα για τον πυρήνα Linux Android, τις εγγενείς βιβλιοθήκες, τη Dalvik VM, και των διαφόρων πακέτων Android (όπως το Android framework και τις προεγκατεστημένες εφαρμογές). Ο εξομοιωτής παρέχει δυναμική δυαδική μετάφραση του κώδικα μηχανής της συσκευής με το λειτουργικό σύστημα και επεξεργαστή αρχιτεκτονικής της μηχανής της εφαρμογής μας.

Ο Android emulator υποστηρίζει πολλά χαρακτηριστικά hardware που βρίσκονται σε κινητές συσκευές, όπως οι εξής:

- Μια CPU ARMv5 και η αντίστοιχη μονάδα διαχείρισης μνήμης (MMU).
- Μια οθόνη LCD 16-bit.
- Ένα ή περισσότερα πληκτρολόγια (ένα πληκτρολόγιο Qwerty, και τα αντίστοιχα κουμπιά Dpad / τηλέφωνο).
- Ένα chip ήχου με δυνατότητες εξόδου και εισόδου.
- Χωρίσματα μνήμης flash (προσομοιωμένα μέσα από τα αρχεία εικόνας δίσκου για την ανάπτυξη της μηχανής).
- Ένα μόντεμ GSM, συμπεριλαμβανομένου μιας προσομοιωμένης κάρτας SIM.
- Μια φωτογραφική μηχανή, χρησιμοποιώντας μια κάμερα συνδεδεμένη στον υπολογιστή μας.
- Αισθητήρες σαν ένα επιταχυνσιόμετρο, χρησιμοποιώντας δεδομένα από μια συνδεδεμένη συσκευή Android μέσω USB.

Android Virtual Devices και ο Emulator

Για να χρησιμοποιήσουμε τον εξομοιωτή, πρέπει πρώτα να δημιουργήσουμε μία ή περισσότερες διαμορφώσεις AVD. Σε κάθε διαμόρφωση, μπορούμε να καθορίσουμε μια πλατφόρμα Android να τρέχει στον εξομοιωτή και το σύνολο των επιλογών του hardware και του δέρματος εξομοιωτή που θέλουμε να χρησιμοποιήσουμε. Κατά την εκκίνηση του emulator, μπορούμε να καθορίσουμε τη διαμόρφωση AVD που θέλουμε να φορτώσουμε.

Κάθε AVD λειτουργεί ως ανεξάρτητο όργανο, με τη δική του ιδιωτική αποθηκευτοποίηση για τα δεδομένα των χρηστών, κάρτα SD, και ούτω καθεξής. Κατά την εκκίνηση του εξομοιωτή με μία διαμόρφωση AVD, φορτώνει αυτόματα τα δεδομένα του χρήστη και τα δεδομένα της κάρτας SD από τον κατάλογο AVD. Από προεπιλογή, ο εξομοιωτής αποθηκεύει τα δεδομένα των χρηστών, τα δεδομένα της κάρτας SD, και cache στον κατάλογο AVD. Για να δημιουργήσουμε και να διαχειριστούμε AVDs χρησιμοποιούμε τον AVD Manager UI ή το android εργαλείο που περιλαμβάνεται στο SDK.

Εκκίνηση και σταμάτημα του εξομοιωτή

Κατά την ανάπτυξη και τη δοκιμή της εφαρμογής μας, μπορούμε να εγκαταστήσουμε και να εκτελέσουμε την εφαρμογή στο Android emulator. Μπορούμε να εκκινήσουμε τον εξομοιωτή ως μια αυτόνομη εφαρμογή από μια γραμμή εντολών, ή μπορούμε να εκτελέσουμε μέσα από το περιβάλλον ανάπτυξης του Eclipse. Σε κάθε περίπτωση, μπορούμε να καθορίσουμε τη διαμόρφωση AVD να φορτώσει και τις επιλογές εκκίνησης που θέλουμε να χρησιμοποιήσουμε.

Μπορούμε να εκτελέσουμε την εφαρμογή μας σε ένα μόνο παράδειγμα του εξομοιωτή ή, ανάλογα με τις ανάγκες μας, μπορούμε να ξεκινήσουμε πολλαπλές περιπτώσεις εξομοιωτή και να εκτελέσουμε την εφαρμογή μας σε περισσότερες από μία προσομοιώσεις συσκευών. Μπορούμε να χρησιμοποιήσουμε ενσωματωμένες εντολές του εξομοιωτή για την προσομοίωση κλήσης τηλεφώνου GSM ή SMS μεταξύ των περιπτώσεων εξομοιωτή, και μπορούμε να ρυθμίσουμε την ανακατεύθυνση δικτύου που επιτρέπει emulators να στέλνουν δεδομένα σε έναν άλλο. Για να σταματήσουμε ένα παράδειγμα του εξομοιωτή, απλά κλείνουμε το παράθυρο του εξομοιωτή.

Χρησιμοποιώντας Hardware Acceleration

Για να τρέξει γρηγορότερα ο Android εξομοιωτής και να είναι πιο ευέλικτος, μπορούμε να τον ρυθμίσουμε να εκμεταλλεύεται την επιτάχυνση hardware, χρησιμοποιώντας ένα συνδυασμό επιλογών, συγκεκριμένων εικόνων συστήματος Android και τους hardware drivers.

3.5.6 Χρησιμοποιώντας συσκευές Hardware

Κατά τη δημιουργία μιας εφαρμογής κινητού, είναι σημαντικό πάντα να δοκιμάζουμε την εφαρμογή μας σε μια πραγματική συσκευή πριν από την δημοσίευσή της στους χρήστες.

Μπορούμε να χρησιμοποιήσουμε οποιαδήποτε συσκευή Android ως ένα περιβάλλον για την εκτέλεση, τον εντοπισμό σφαλμάτων και δοκιμή των εφαρμογών μας. Τα εργαλεία που περιλαμβάνονται στο SDK καθιστούν εύκολη την εγκατάσταση και την εκτέλεση της εφαρμογής μας στη συσκευή κάθε φορά που κάνουμε compile. Μπορούμε να εγκαταστήσουμε την εφαρμογή μας στη συσκευή απευθείας από το Eclipse ή από τη γραμμή εντολών με την ADB.

Κατά την ανάπτυξη σε μια συσκευή, πρέπει να εξακολουθούμε να χρησιμοποιούμε τον Android emulator για να ελέγξουμε την εφαρμογή μας για διαμορφώσεις που δεν είναι ισοδύναμες με αυτές της πραγματικής συσκευής μας. Παρά το γεγονός ότι ο εξομοιωτής δεν μας επιτρέπει να δοκιμάσουμε κάθε χαρακτηριστικό της συσκευής (όπως το επιταχυνσιόμετρο), μας επιτρέπει να βεβαιωθούμε ότι λειτουργεί σωστά η εφαρμογή μας σε διαφορετικές εκδόσεις της πλατφόρμας Android, σε διαφορετικά μεγέθη οθόνης και προσανατολισμούς, και πολλά άλλα.

Έτσι ήταν απαραίτητο να ελέγχουμε το παιχνίδι μας και σε πραγματικές συσκευές Android αλλά και στον emulator, ώστε να μπορέσουμε να εξετάσουμε αν όλα λειτουργούν σωστά. Αρχικά ελέγξαμε στον emulator αν δουλεύει σωστά σε διαφορετικά μεγέθη οθόνης και προσανατολισμούς. Στην πορεία όμως εγκαταλείψαμε τον emulator γιατί ήταν αρκετά βαρή πρόγραμμα παρουσιάζοντας έντονες καθυστερήσεις στον υπολογιστή, και επίσης επειδή το παιχνίδι βασιζόταν στο επιταχυνσιόμετρο έπρεπε να χρησιμοποιήσουμε συσκευές Android. Αφού ελέγξαμε ότι λειτουργεί σωστά σε διαφορετικά μεγέθη οθονών μέσω του emulator, χρησιμοποιούσαμε μόνο συσκευές Android για τους ελέγχους της συσκευής.

Ρύθμιση συσκευής για την Ανάπτυξη

Με μια Android-powered συσκευή, μπορούμε να αναπτύξουμε και να διορθώσουμε Android εφαρμογές, ακριβώς όπως θα κάναμε σε εξομοιωτή. Για να μπορέσουμε να ξεκινήσουμε, υπάρχουν μόνο μερικά πράγματα που πρέπει να κάνουμε:

1. Δηλώνουμε την εφαρμογή μας ως "debuggable" στο Android σας Manifest.
   Όταν χρησιμοποιούμε Eclipse, μπορούμε να παραλείψουμε αυτό το βήμα, επειδή η εκτέλεση της εφαρμογής μας απευθείας από το Eclipse IDE ενεργοποιεί αυτόματα τον εντοπισμό σφαλμάτων.
   Στο αρχείο AndroidManifest.xml, προσθέτουμε το `android:debuggable="true"` στο `<application>` στοιχείο.
   Εάν ενεργοποιήσουμε τον εντοπισμό σφαλμάτων στο αρχείο δήλωσης, να πρέπει να το απενεργοποιήσουμε πριν το δημοσιεύσουμε (η δημοσιευμένη εφαρμογή μας δεν θα πρέπει να είναι debuggable).
2. Ενεργοποίηση εντοπισμού σφαλμάτων USB στη συσκευή μας.

Στις περισσότερες συσκευές που τρέχουν Android 3.2 ή παλαιότερες εκδόσεις, μπορούμε να βρούμε την επιλογή αυτή στις Ρυθμίσεις> Εφαρμογές> Ανάπτυξη (Settings > Applications > Development).
Με το Android 4.0 και στις νεότερες εκδόσεις, είναι στις Settings > Developer options.

3. Ρύθμιση του συστήματός μας για να εντοπίσει τη συσκευή μας.
   Αν χρησιμοποιούμε Windows, θα πρέπει να εγκαταστήσουμε ένα USB driver για την adb, ενώ σε MAC λειτουργεί κανονικά.

OEM USB Drivers

Εάν αναπτύσσουμε σε Windows και θέλουμε να συνδέσουμε μια Android-powered συσκευή για να ελέγξουμε τις εφαρμογές μας, τότε θα πρέπει να εγκαταστήσουμε το κατάλληλο USB driver. Μπορούμε να βρούμε να κατεβάσουμε το κατάλληλο USB driver για τη συσκευή μας σε ιστοσελίδες για διάφορες κατασκευαστές πρωτότυπου εξοπλισμού (OEM). Ωστόσο, ο κατάλογος δεν είναι εξαντλητικός για όλες τις διαθέσιμες συσκευές με λογισμικό Android.

Εμείς εγκαταστήσαμε τους τελευταίους USB drivers για το Samsung Galasy S2 και το Sony neoV, ώστε να μπορέσουμε να τρέξουμε την εφαρμογή του παιχνιδιού μας σε αυτές τις δύο κινητές συσκευές. Βρήκαμε τον κατάλληλο driver για την κάθε συσκευή από την OEM drivers.

3.5.7 Managing Projects

Τα projects εμπεριέχουν κώδικα και αρχεία των πόρων. Τα εργαλεία SDK απαιτούν τα έργα μας να ακολουθούν μια συγκεκριμένη δομή, ώστε να μπορούν να καταρτίζουν και να συσκευάζουν την εφαρμογή μας σωστά, γι 'αυτό συνιστάται ιδιαίτερα να τα δημιουργήσουμε με το Eclipse και το ADT ή με το android εργαλείο στη γραμμή εντολών. Υπάρχουν τρεις τύποι έργων, και όλοι έχουν την ίδια γενική δομή, αλλά διαφέρουν σε:

- Android Projects: Ένα Android project είναι το εμπεριέχει τον πηγαίο κώδικα της εφαρμογής μας, αρχεία πόρων, και αρχεία όπως το Ant build και το αρχείο Android Manifest. Ένα project εφαρμογής είναι ο κύριος τύπος του έργου και τα περιεχόμενα είναι τελικά χτισμένα σε ένα αρχείο .apk που έχουμε εγκαταστήσει σε μια συσκευή.
- Test Projects: Τα projects αυτά περιέχουν κώδικα για να δοκιμάσουμε την εφαρμογή μας και είναι χτισμένα σε εφαρμογές που τρέχουν σε μια συσκευή.
- Library Projects: Τα έργα αυτά περιλαμβάνουν κοινόχρηστο τον πηγαίο κώδικα Android και τους πόρους που μπορούμε να αναφερθούμε σε έργα Android. Αυτό είναι χρήσιμο όταν έχουμε κοινό κώδικα που θέλουμε να χρησιμοποιήσουμε ξανά. Τα Library Projects δεν μπορούν να εγκατασταθούν σε μια συσκευή, όμως, είναι στο αρχείο .apk κατά το χρόνο κατασκευής.

Όταν χρησιμοποιούμε τα εργαλεία ανάπτυξης του Android για να δημιουργήσουμε ένα νέο έργο, θα δημιουργηθούν τα βασικά αρχεία και οι φάκελοι για εμάς. Υπάρχουν μόνο λίγα αρχεία και φακέλοι που δημιουργούνται για μας, και κάποια από αυτά εξαρτόνται από το αν χρησιμοποιούμε το plugin Eclipse ή το android εργαλείο για να δημιουργήσουμε το έργο μας. Δεδομένου ότι η εφαρμογή μας αυξάνεται σε πολυπλοκότητα, μπορεί να απαιτήσει νέα είδη πόρων, καταλόγων και αρχείων.

3.5.8 Building και Running

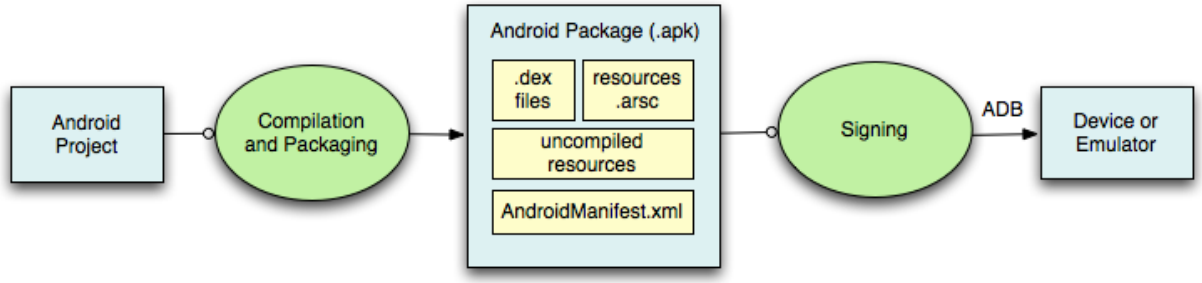
Κατά τη διάρκεια της διαδικασίας ανάπτυξης, τα Android projects συγκεντρώνονται και συσκευάζονται σε ένα αρχείο .apk , που περιέχει τη δυαδική εφαρμογή μας. Περιέχει όλες τις απαραίτητες πληροφορίες για να εκτελεστεί η εφαρμογή μας σε μια συσκευή ή εξομοιωτή, όπως καταρτίζεται σε .dex αρχεία (αρχεία .class μετατρέπονται σε Dalvik byte κώδικα), μια δυαδική έκδοση του αρχείου AndroidManifest.xml, συγκεντρωμένοι πόροι (resources.arsc) και uncompiled αρχεία πόρων για την εφαρμογή μας.

Το Eclipse και το ADT παρέχει ένα περιβάλλον όπου οι περισσότερες από τις λεπτομέρειες της διαδικασίας κατασκευής είναι κρυμμένες από εμάς. Εάν αναπτύσσουμε στο Eclipse, το plugin ADT φτιάχνει σταδιακά το έργο μας όπου η διαδικασία κατασκευής τρέχει συνεχώς στο παρασκήνιο, καθώς κάνουμε τις αλλαγές στον πηγαίο κώδικα. Το Eclipse εξάγει ένα αρχείο .apk αυτόματα στο φάκελο bin του έργου, έτσι ώστε να μην χρειάζεται να κάνουμε τίποτα επιπλέον για τη δημιουργία του .apk .

Αν δεν αναπτύσσουμε σε περιβάλλον Eclipse, μπορούμε να δημιουργήσουμε το έργο μας με το παραγόμενο build.xml Ant αρχείο που βρίσκεται στην διεύθυνση του έργου. Το αρχείο Ant καλεί τους στόχους που θέτουν αυτόματα τα εργαλεία δημιουργίας για μας.

Για να εκτελέσουμε μια εφαρμογή σε έναν εξομοιωτή ή τη συσκευή, η εφαρμογή πρέπει να υπογράφεται με debug ή με release λειτουργία. Συνήθως θέλουμε να υπογράψουμε την εφαρμογή μας σε κατάσταση εντοπισμού σφαλμάτων (debug mode) όταν αναπτύσσουμε και δοκιμάζουμε την εφαρμογή μας, επειδή τα εργαλεία κατασκευής χρησιμοποιούν ένα κλειδί debug με ένα γνωστό κωδικό πρόσβασης, ώστε να μην χρειάζεται να τον εισάγουμε κάθε φορά που την δημιουργούμε. Όταν είμαστε έτοιμοι να απελευθερώσουμε την εφαρμογή στο Google Play, θα πρέπει να υπογράψουμε την εφαρμογή στη λειτουργία απελευθέρωσης (release mode), χρησιμοποιώντας το δικό μας ιδιωτικό κλειδί.

Ευτυχώς, το Eclipse ή το Ant έχει έτοιμο κώδικα (build script) που υπογράφει την εφαρμογή για μας σε κατάσταση εντοπισμού σφαλμάτων κατά τη δημιουργία της εφαρμογής μας. Μπορούμε επίσης εύκολα να ρυθμίσουμε τον έτοιμο κώδικα του Eclipse ή του Ant για να υπογράψουμε την εφαρμογή μας σε λειτουργία απελευθέρωσης. Το παρακάτω διάγραμμα απεικονίζει τα συστατικά που εμπλέκονται στην κατασκευή και τη λειτουργία μιας εφαρμογής:

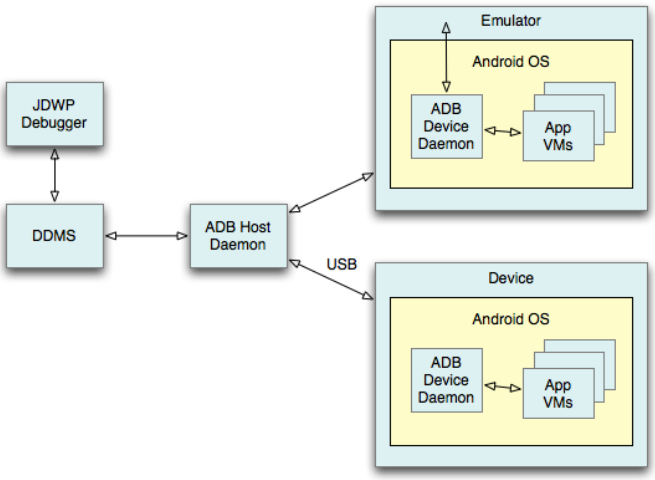


3.5.9 Testing

Το Android framework περιλαμβάνει ένα ολοκληρωμένο πλαίσιο ελέγχου που μας βοηθά να δοκιμάσουμε όλες τις πτυχές της εφαρμογής μας και τα εργαλεία SDK περιλαμβάνουν εργαλεία για τη δημιουργία και τη λειτουργία δοκιμή εφαρμογών. Είτε εργαζόμαστε σε Eclipse με ADT ή από τη γραμμή εντολών, τα εργαλεία SDK μας βοηθούν να ρυθμίσουμε και να εκτελέσουμε τις δοκιμές μας μέσα σε έναν εξομοιωτή ή τη συσκευή που στοχεύουμε.

3.5.10 Debugging

Το Android SDK παρέχει τα περισσότερα από τα εργαλεία που χρειαζόμαστε για να διορθώσουμε τις εφαρμογές μας. Χρειάζεται ένα JDWP συμβατό πρόγραμμα εντοπισμού σφαλμάτων, αν θέλουμε να είμαστε σε θέση να δουλέψουμε μέσα από τον κώδικα, να δούμε τιμές των μεταβλητών, και να διακόψουμε την εκτέλεση της εφαρμογής. Εάν χρησιμοποιούμε Eclipse, περιλαμβάνεται ήδη ένα JDWP συμβατό πρόγραμμα εντοπισμού σφαλμάτων και δεν απαιτείται εγκατάσταση. Εάν χρησιμοποιούμε κάποιο άλλο IDE, μπορούμε να χρησιμοποιήσουμε το πρόγραμμα εντοπισμού σφαλμάτων που έρχεται με αυτό και να επισυνάψουμε το πρόγραμμα εντοπισμού σφαλμάτων σε μια ειδική θύρα, ώστε να μπορεί να επικοινωνεί με την εφαρμογή VMs στις συσκευές μας. Η εικόνα που ακολουθεί δείχνει πως συνεργάζονται τα διάφορα εργαλεία εντοπισμού σφαλμάτων σε ένα τυπικό περιβάλλον εντοπισμού σφαλμάτων.



3.5.11 Publishing Overview

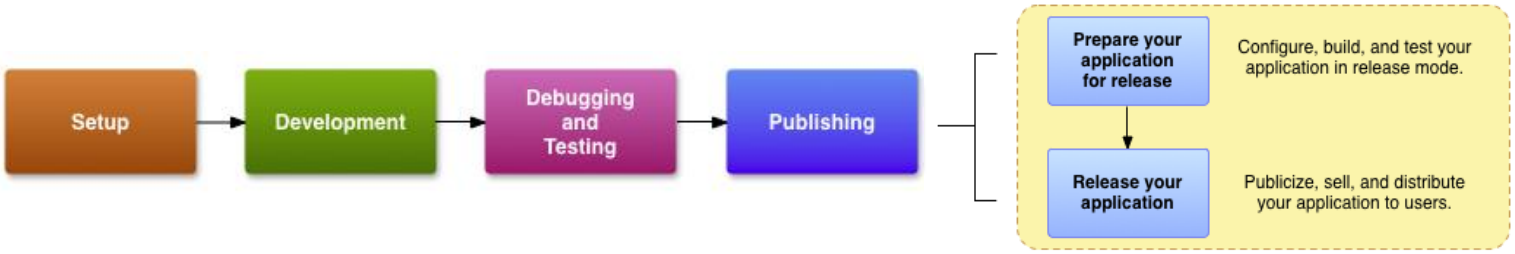
Η έκδοση (Publishing) είναι η γενική διαδικασία που κάνει τις Android εφαρμογές διαθέσιμες στους χρήστες. Όταν δημοσιεύουμε μία Android εφαρμογή, εκτελούμε δύο βασικά καθήκοντα :

- Προετοιμάζουμε την εφαρμογή για την απελευθέρωση: Κατά το στάδιο της προετοιμασίας δημιουργούμε μια έκδοση της εφαρμογής μας, την οποία οι χρήστες μπορούν να κατεβάσουν και να εγκαταστήσουν στις Android-powered συσκευές τους.

- Απελευθερώνουμε την εφαρμογή στους χρήστες: Κατά το στάδιο της απελευθέρωσης θα δημοσιοποιηθεί, πουληθεί και διανεμηθεί η έκδοση της εφαρμογής μας στους χρήστες.

Συνήθως, δημοσιεύουμε την εφαρμογή μας μέσω μιας αγοράς εφαρμογών, όπως το Google Play. Ωστόσο, μπορούμε επίσης να απελευθερώσουμε τις εφαρμογές στέλνοντάς τες απευθείας στους χρήστες ή αφήνοντας τους χρήστες να την κατεβάσουν από τη δική μας ιστοσελίδα.

Η παρακάτω εικόνα δείχνει πώς η εκδοτική διαδικασία εντάσσεται στη συνολική διαδικασία ανάπτυξης της Android εφαρμογής. Η διαδικασία έκδοσης συνήθως πραγματοποιείται αφού ολοκληρώσουμε τη δοκιμή της εφαρμογής μας σε ένα περιβάλλον εντοπισμού σφαλμάτων. Επίσης, ως βέλτιστη πρακτική, η εφαρμογή μας θα πρέπει να πληρεί όλα τα κριτήρια δημοσίευσης για τη λειτουργικότητα, την απόδοση και τη σταθερότητα πριν ξεκινήσουμε τη διαδικασία έκδοσης.



3.5.12 Support Library

Το πακέτο Android Support Library είναι ένα σύνολο από βιβλιοθήκες κώδικα που παρέχουν συμβατές εκδόσεις του Android APIs πλαισίου, καθώς και χαρακτηριστικά που είναι διαθέσιμα μόνο μέσω των APIs της βιβλιοθήκης. Κάθε Support Library είναι συμβατή με ένα συγκεκριμένο επίπεδο Android API. Αυτός ο σχεδιασμός σημαίνει ότι οι εφαρμογές μας μπορούν να χρησιμοποιούν τις δυνατότητες των βιβλιοθηκών και να εξακολουθούν να είναι συμβατές με τις συσκευές που τρέχουν Android (επίπεδο API 4) 1.6 και πάνω.

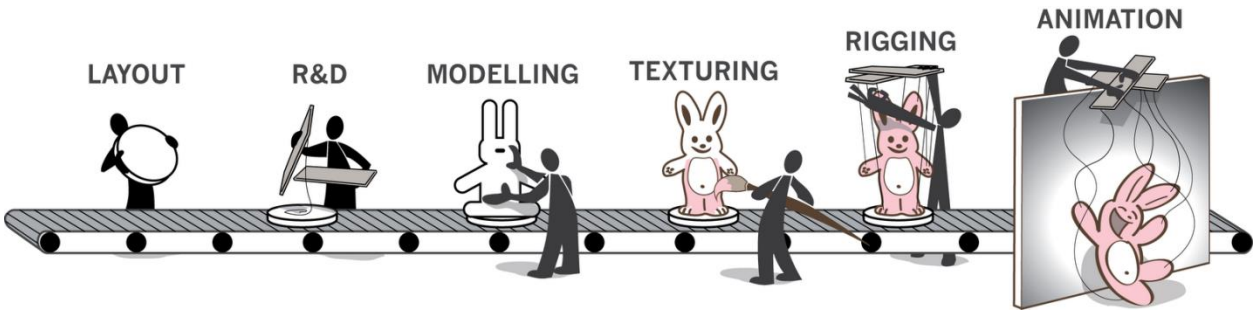
3.5.13 Εργαλεία

Το Android SDK περιλαμβάνει μια ποικιλία από εργαλεία που μας βοηθούν να αναπτύξουμε κινητές εφαρμογές για την πλατφόρμα Android. Τα εργαλεία κατατάσσονται σε δύο ομάδες: τα SDK tools και platform tools. Τα SDK tools είναι ανεξάρτητα από την πλατφόρμα και απαιτούνται ανεξάρτητα από την πλατφόρμα Android που αναπτύσσονται. Τα platform tools έχουν προσαρμοστεί για να υποστηρίζουν τα χαρακτηριστικά της τελευταίας πλατφόρμας Android.

3.6 Βασικά τεχνικά χαρακτηριστικά του Maya

3.6.1 Επισκόπηση

Το Maya είναι μια εφαρμογή που χρησιμοποιείται για τη δημιουργία 3D περιουσιακών στοιχείων για χρήση σε κινηματογράφο, τηλεόραση, ανάπτυξη παιχνιδιών και αρχιτεκτονική, ακολουθώντας μια σειρά από διαδικασίες μέχρι το τελικό στάδιο.



Οι χρήστες ορίζουν έναν εικονικό χώρο εργασίας (σκηνή) για να εφαρμόσουν και να επεξεργαστούν τα μέσα ενός συγκεκριμένου έργου. Οι σκηνές μπορούν να αποθηκευτούν σε μια ποικιλία μορφών, η προεπιλεγμένη μορφή αρχείου είναι σε .Mb (Maya Binary). Το Maya εκθέτει ένα κόμβο αρχιτεκτονικού γραφήματος. Τα στοιχεία της σκηνής είναι ο κόμβος-βάση, κάθε κόμβος έχει τη δική του ιδιότητα και παραμετροποίηση. Ως αποτέλεσμα, η οπτική αναπαράσταση μιας σκηνής βασίζεται εξ'ολοκλήρου σε ένα δίκτυο διασύνδεσης κόμβων, ανάλογα με τις πληροφορίες του άλλου. Για τη διευκόλυνση της προβολής αυτών των δικτύων, υπάρχει μια εξάρτηση και ένα κατευθυνόμενο άκυκλο γράφημα .

Οι χρήστες, που είναι φοιτητές ή καθηγητές, μπορούν να κατεβάσουν μια πλήρη εκπαιδευτική έκδοση από την κοινότητα Autodesk Εκπαίδευση. Οι εκδόσεις που διατίθενται στην κοινότητα έχουν άδεια μόνο για μη εμπορική χρήση, η οποία ενεργοποιείται με την πλήρη άδεια 36 μηνών του προϊόντος, και ορισμένα προϊόντα δημιουργούν υδατογραφήματα στην παραγωγή. Μόλις λήξει η άδεια, οι χρήστες μπορούν να συνδεθούν με την κοινότητα και να ζητήσουν νέα άδεια 36 μηνών και να κατεβάσουν το πιο πρόσφατο προϊόν της Autodesk .

Επιπλέον, μια αέναη μαθητική άδεια μπορεί να αγοραστεί για το Maya . Αυτή η άδεια δεν λήγει και η έκδοση των φοιτητών μπορεί να αναβαθμιστεί με την εμπορική έκδοση με σημαντική έκπτωση. Μπορεί να χρησιμοποιηθεί ακόμη και μετά την αποφοίτηση του μαθητή, ο μόνος περιορισμός είναι μη εμπορική χρήση. Δεν δημιουργούνται υδατογραφήματα κατά τη διάρκεια της παραγωγής, καθιστώντας την μαθητική έκδοση του Maya κατάλληλη για τη δημιουργία χαρτοφυλακίου. Τα αρχεία που αποθηκεύονται με αυτή την έκδοση αναγνωρίζονται από όλες τις εκδόσεις των Μάγια ως αρχεία που δημιουργούνται από μια έκδοση των φοιτητών. Η αέναη μαθητική άδεια επιτρέπει επίσης τη δημιουργία των μη

εμπορικών περιουσιακών στοιχείων για μη εμπορική χρήση σε μηχανές παιχνιδιού όπως το Unreal Development Kit . Η ελεύθερη φοιτητική άδεια δεν το επιτρέπει αυτό.

3.6.2 Components

Fluid Effects

Ένας ρεαλιστικός εξομοιωτής ρευστού που βασίζεται σε απλουστευμένες, ασυμπίεστες εξισώσεις Navier-Stokes για την προσομοίωση μη ελαστικών υγρών. Είναι αποτελεσματικό για τον καπνό, τη φωτιά, τα σύννεφα και τις εκρήξεις, καθώς και για εφέ πολύ παχού υγρού, όπως το νερό, το μάγμα ή τη λάσπη που προστέθηκε στο Maya 4.5.

Κλασική Ένδυση

Ένα δυναμικό εργαλείο προσομοίωσης ένδυσης του συστήματος nCloth, που χρησιμοποιεί ένα πρότυπο επίπεδο based workflow εμπνευσμένο από τη διαδικασία που χρησιμοποιείται για το σχεδιασμό πρότυπου ενδύματος πραγματικού κόσμου.

Γούνα

Η προσομοίωση γούνας είναι σχεδιασμένη για μεγάλη περιοχή κάλυψης με κοντές τρίχες και μαλλιά, όπως τα υλικά. Μπορεί να χρησιμοποιηθεί για την προσομοίωση κοντής γούνας αντικειμένων, όπως το γρασίδι, το χαλί, κλπ. Σε αντίθεση με το Maya Hair, η μονάδα Fur δεν κάνει καμία προσπάθεια να αποτρέψει συγκρούσεις μεταξύ των μαλλιών. Οι τρίχες είναι επίσης ανίκανες να αντιδράσουν δυναμικά σε φυσικές δυνάμεις με βάση τα μαλλιά. Τα αποτελέσματα φυσικής επιτυγχάνονται μέσω κοντινών τελεστών γούνας που προσεγγίζουν το αποτέλεσμα των φυσικών δυνάμεων κατά μέσο όρο πάνω από κοντινές follicles (ωοθυλακιες).

nHair

Ο προσομοιωτής μαλλιών είναι ικανός για προσομοίωση δυναμικών δυνάμεων που ενεργούν για μακριά μαλλιά και ανά συγκρούσεις μαλλιών. Συχνά χρησιμοποιείται για την προσομοίωση πολύπλοκων υπολογιστικών μορφων ανθρώπινης τρίχας, συμπεριλαμβανομένων ουρές πόνυ, περμανάντ και πλεξούδες. Η προσομοίωση χρησιμοποιεί καμπύλες NURBS ως βάση οι οποίες στη συνέχεια χρησιμοποιούνται ως κινήσεις για τις πινελιές των Paint Effects, δίνοντας έτσι στις καμπύλες το χρόνο απόδοσης (render) να αναπαραστήσουν μια επιφανεία, η οποία μπορεί να αλληλεπιδράσει με το φως και τη σκιά. Μια προσομοίωση μόνο από τις καμπύλες για άλλους σκοπούς εκτός από τα μαλλιά (όπως εύκαμπτους σωλήνες, καλώδια, σχοινιά, κλπ) είναι συχνά γνωστή απλά ως δυναμικές καμπύλες (Dynamic Curves).

nCloth

Είναι η πρώτη εφαρμογή των Maya Nucleus της Autodesk προσομοίωσης πλαισίου, γρήγορη και ευέλικτη, και αντικατέστησε το Maya Cloth. Το nCloth παρέχει καλλιτεχνία με λεπτομερή έλεγχο των υφασμάτων και των υλικών των προσομοιώσεων.

nParticle

Προστέθηκε στην έκδοση του 2009 , το nParticle είναι προσθήκη στα εργαλεία του Maya Nucleus. Το nParticle είναι να προσομοιώνει ένα ευρύ φάσμα πολύπλοκων 3D εφέ , συμπεριλαμβανομένων των υγρών , τα σύννεφα , τον καπνό , το σπρέι , και τη σκόνη . Το nParticles είναι πιο ευέλικτο από ό, τι το προηγούμενο σύστημα σωματιδίων του Μάγια και το μπορεί να χρησιμοποιηθεί για την προσομοίωση παχύρρευστων υγρών , καθώς και την υποστήριξη πραγματικών συγκρούσεων σωματιδίων. Το nParticles επίσης αλληλεπιδράει με το υπόλοιπο του πλαισίου προσομοίωσης Nucleus χωρίς την ανάγκη δαπανηρών επεμβάσεων που και scripting .

MatchMover

Επιτρέπει το compositing των στοιχείων CGI με τα δεδομένα κίνησης από ακολουθίες βίντεο και φιλμ, μια διαδικασία γνωστή ως Match moving ή εντοπισμός της κάμερας. Είναι ένα εξωτερικό πρόγραμμα, αλλά διατίθεται στην αγορά με το Μάγια.

Camera Sequencer

Η κάμερα Sequencer χρησιμοποιείται για τη διάταξη πολλαπλών λήψειων της κάμερας και τη διαχείρισή τους σε μια σειρά κινουμένων σχεδίων .

3.6.3 Maya Embedded Language

Παράλληλα με τις πιο αναγνωρισμένες οπτικές ροές εργασίας, το Maya είναι εξοπλισμένο με μια cross-platform scripting γλώσσα, που ονομάζεται Maya Embedded Language. Η MEL προβλέπεται για scripting και ως ένα μέσο για να προσαρμόσουμε τη βασική λειτουργικότητα του λογισμικού, δεδομένου ότι πολλά από τα εργαλεία και τις εντολές που χρησιμοποιούνται είναι γραμμένα σε αυτό. Ο κώδικας μπορεί να χρησιμοποιηθεί για την κατασκευή τροποποιήσεων, plug-ins ή να τον έλεγχο της εκτέλεσης. Η αλληλεπίδραση με το χρήστη καταγράφεται στο MEL, επιτρέποντας ακόμα και άπειρους χρήστες να εφαρμόζουν υπορουτίνες. Οι πληροφορίες σκηνικού μπορεί έτσι να επεκταθούν σε αρχείο .Ma, το οποίο είναι επεξεργάσιμο έξω από Maya σε οποιοδήποτε πρόγραμμα επεξεργασίας κειμένου.

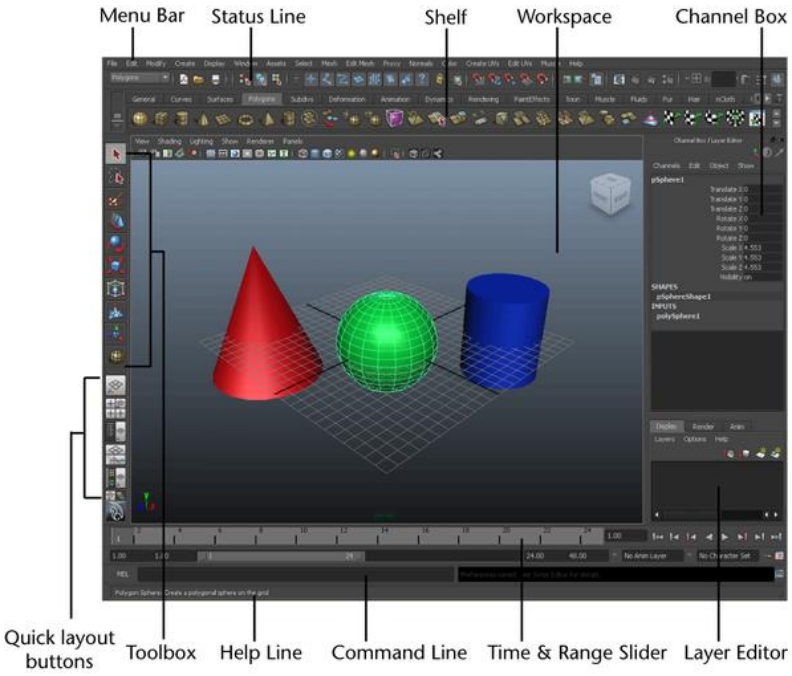
3.6.4 Απαιτήσεις συστήματος

Η Autodesk υποστηρίζει τις πλατφόρμες Windows (XP SP3 ή νεότερη έκδοση), Mac και Linux. Έχει δημοσιευθεί απαιτήσεις συστήματος για να τρέξει το Maya σε επαρκείς επιδόσεις. Οι προδιαγραφές είναι ίδιες και για τις δύο x86 και x64 πλατφόρμες.

Hardware	Spec
Processor	Intel Pentium 4 or higher, AMD Athlon 64, AMD Opteron processor, AMD Phenom processor
Video card	Qualified hardware-accelerated OpenGL graphics cards
Memory	2 GB, 4 GB for 64-bit OS
Hard drive	10 GB
Optical drive	DVD-ROM
Internet browser	Microsoft Internet Explorer 7.0 or higher, Apple Safari, or Mozilla Firefox, or Google Chrome

3.7 Η δομή του Maya

3.7.1 Interface



Το Maya έχει ένα πολύπλοκο interface το οποίο χωρίζεται σε πολλά διαφορετικά τμήματα διαφορετικών λειτουργιών. Το workspace είναι ο χώρος εργασίας του Maya στον οποίο διεξάγουμε το μεγαλύτερο μέρος της εργασίας, είναι το κεντρικό παράθυρο όπου εμφανίζονται τα αντικείμενα μας και τα περισσότερα editor πάνελ.

Εργαλεία και αντικείμενα είναι προσβάσιμα από το μενού που βρίσκεται στην κορυφή της διεπαφής χρήστη. Τα μενού ομαδοποιούνται σε σύνολα μενού, τα οποία είναι προσβάσιμα από το Menu Bar (Κυρίως Μενού). Το κάθε σύνολο μενού αντιστοιχεί σε μία μονάδα (module) του Μάγια: Animation, Polygons (πολύγωνα), Surfaces (επιφάνειες), Rendering, και Dynamics. Οι μονάδες είναι μια μέθοδος ομαδοποίησης σχετικών δυνατοτήτων και εργαλείων. Παραδείγματος χάριν, για τη δημιουργία ενός 3D αντικειμένου θα επιλέξουμε το Polygons μενού, ενώ για τη δημιουργία κίνησης επιλέγουμε το Animation μενού.

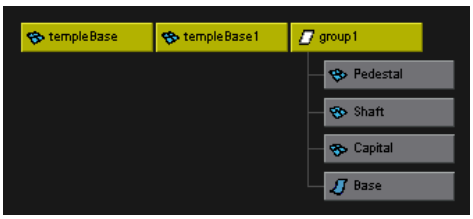
Το Status Line, που βρίσκεται ακριβώς κάτω το κύριο μενού, περιέχει μια ποικιλία από αντικείμενα, τα περισσότερα από τα οποία χρησιμοποιούνται κατά την μοντελοποίηση ή την εργασία με αντικείμενα στο Μάγια. Πολλά από τα αντικείμενα αντιπροσωπεύονται από ένα γραφικό εικονίδιο στο Status Line ώστε να έχουμε γρήγορη πρόσβαση σε αυτά.

ΤΟ Shelf, που βρίσκεται ακριβώς κάτω από το Status Line, περιλαμβάνει εργαλεία και αντικείμενα που χρησιμοποιούνται συχνά και οργανώνονται σε αντίστοιχες τοποθεσίες.

Στο Channel Box, στη δεξιά πλευρά της διεπαφής χρήστη, φαίνονται αριθμητικά στοιχεία. Οι πληροφορίες αυτές αφορούν τα X, Y, και Z, τη θέση, την περιστροφή, και την κλιμάκωση για το ενεργό αντικείμενο. Τα X, Y, και Z της θέσης ισούνται με 0 όταν το αντικείμενο είναι στην αρχική του θέση.

Το Toolbox περιέχει εργαλεία για βασικές λειτουργίες, όπως την επιλογή, την μετακίνηση, την περιστροφή και την κλιμάκωση του αντικειμένου. Τα Quick Layout buttons καθορίζουν την οπτική γωνία του χώρου εργασίας. Το Help Line ενημερώνει για τυχόν λάθη του έργου μας. Στο Command Line μπορούμε να γράψουμε κώδικα ο οποίος θα επιδράσει στο μοντέλο μας. Το Time & Range Slider είναι το διάγραμμα του χρόνου κατά μήκος του οποίου φτιάχνουμε την κίνηση, δηλαδή τα animations. Στον Layer Editor οργανώνουμε το αρχείο με το μοντέλο που φτιάχνουμε χωρίζοντάς το σε επίπεδα για την καλύτερη διαχείρισή του.

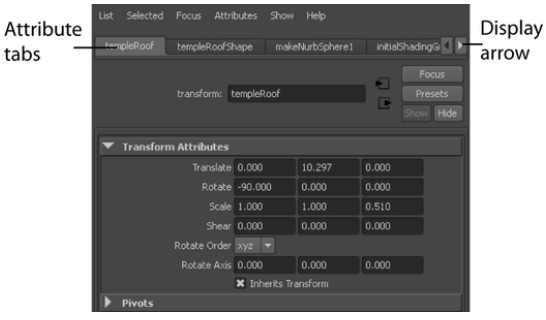
Το Hypergraph είναι ένα παράθυρο που δείχνει πώς οι κόμβοι και οι συνδέσεις τους οργανώνονται στη σκηνή μας. Μπορούμε να προβάλουμε τις ιεραρχίες του αντικειμένου, τις εξαρτήσεις και τις ομαδοποιήσεις του στο Hypergraph.



Ένα σημείο περιστροφής (pivot point) είναι μια συγκεκριμένη θέση στον 3D χώρο που χρησιμοποιείται ως αναφορά για τους μετασχηματισμούς των αντικειμένων. Όλα τα αντικείμενα (καμπύλες, επιφάνειες, ομάδες) έχουν σημεία περιστροφής. Όταν τοποθετούμε αντικείμενα σε μια ομάδα, ένας νέος κόμβος δημιουργείται και ονομάζεται κόμβος γονέας.

Όλα τα αντικείμενα στο Maya έχουν ένα Transform και ένα shape node. Τα γεωμετρικά σχήματα έχουν μικρότερα τμήματα που ονομάζονται components (συστατικά). Μερικά παραδείγματα των συστατικών στο Maya είναι οι control vertices (κορυφές ελέγχου), τα faces (όψεις) και οι hulls (φλοιοί). Τα Components μας επιτρέπουν να εργαζόμαστε με τα αντικείμενα σε ένα επίπεδο λεπτότερο, και να τα επεξεργαστούμε με δημιουργικούς τρόπους. Για να αλλάξουμε το σχήμα πέρα από τις βασικές μετατροπές κλίμακας, με έναν τρόπο που θέλουμε εμείς, θα πρέπει να τροποποιήσετε τα components που το απαρτίζουν.

Ο Attribute Editor (Επεξεργαστής Χαρακτηριστικών) παρέχει πληροφορίες σχετικά με τους διάφορους κόμβους, τα χαρακτηριστικά των αντικειμένων και τα υλικά στη σκηνή μας. Μπορούμε να προβάλουμε και να επεξεργαστούμε τις βασικές πληροφορίες μετασχηματισμού και πολλά άλλα keyable χαρακτηριστικά, καθώς παρέχει μια πιο λεπτομερή απεικόνιση όλων των χαρακτηριστικών για ένα επιλεγμένο αντικείμενο.



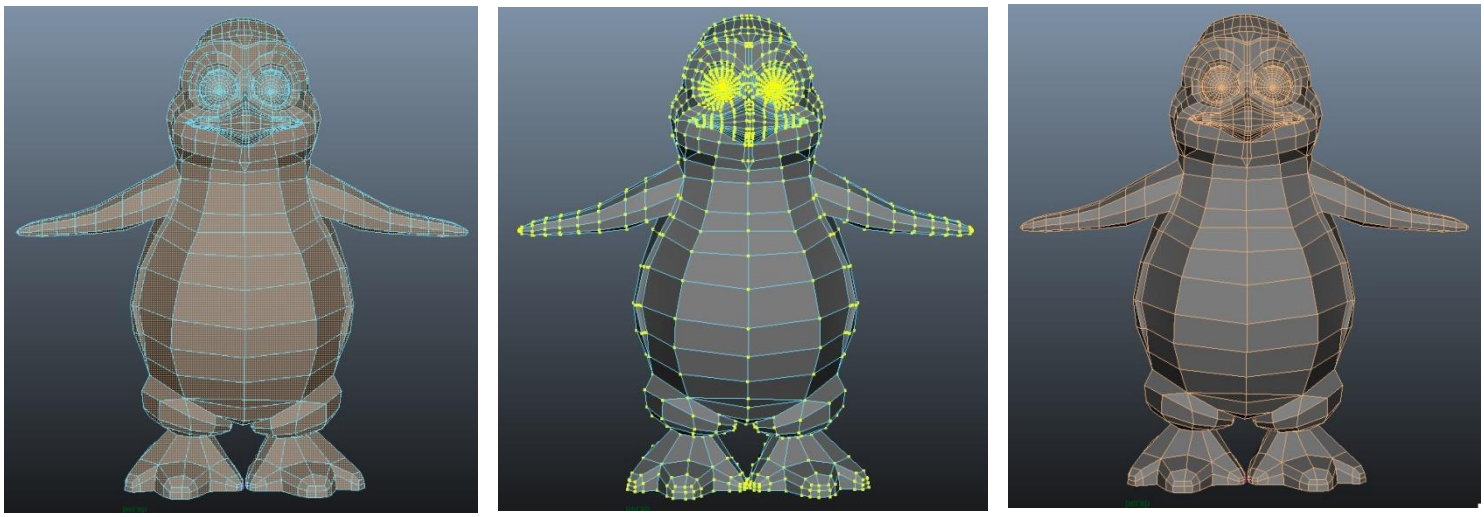
Τα χαρακτηριστικά χρώματος, λαμπρότητας, και ανακλαστικότητας ενός αντικειμένου ελέγχονται από το υλικό της επιφάνειάς του, που μερικές φορές αναφέρεται ως ένας shader, ή υλικό σκίασης. Τα χαρακτηριστικά του υλικού σχετίζονται με το πώς το αντικείμενο μιμείται μια φυσική αντίδραση στο φως του 3D κόσμου των υπολογιστών. Το Maya εκχωρεί ένα προεπιλεγμένο υλικό σκίασης σε όλα τα αντικείμενα όταν δημιουργούνται για πρώτη φορά.

3.7.2 Polygonal Modeling

Η μοντελοποίηση αναφέρεται στη διαδικασία δημιουργίας εικονικών 3D επιφανειών για τους χαρακτήρες και τα αντικείμενα στη σκηνή του Maya. Οι επιφάνειες είναι ζωτικής σημασίας για τη δημιουργία μιας πειστικής εικόνας 3D. Η διαδικασία μοντελοποίησης απαιτεί έντονες οπτικές ικανότητες και γνώση των εργαλείων μοντελοποίησης. Όσο πιο ακριβείς είμαστε κατά τη μοντελοποίηση της φόρμας μας, όσον αφορά το μέγεθος, το σχήμα, τη λεπτομέρεια, και την αναλογία, τόσο πιο πειστική θα είναι η τελική σκηνή.

Υπάρχουν τρεις τύποι μοντελοποίησης επιφάνειας στο Maya, και ο κάθε τύπος επιφάνειας έχει ιδιαίτερα χαρακτηριστικά και οφέλη: Polygons (πολύγωνα), NURBS, και Subdivision surfaces (επιφάνειες υποδιαίρεσης).

Οι επιφάνειες πολυγώνων είναι ένα δίκτυο από τρεις ή περισσότερες επίπεδες όψεις επιφανειών που ονομάζεται faces (όψεις) που συνδέονται μεταξύ τους για να δημιουργήσουν ένα πολυπλέγμα. Τα πολύγωνα των πλεγμάτων αποτελούνται από vertices (κορυφές), faces, και edges (ακμές). Οι γραμμές wireframe στο πλέγμα αντιπροσωπεύουν τις ακμές της κάθε όψης. Οι περιοχές που οριοθετούνται από τις ακμές είναι όψεις. Όπου οι ακμές τέμνονται μεταξύ τους είναι η θέση ενός σημείου που ονομάζεται κορυφή.



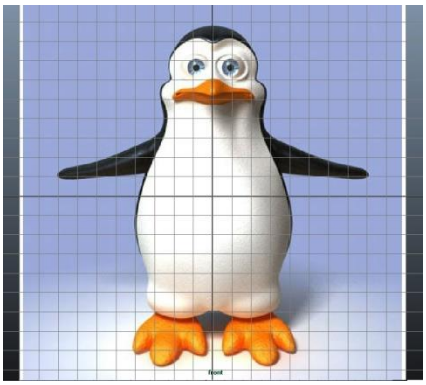
Όταν ένα πλέγμα πολυγώνου αποδίδεται, οι ακμές του μπορούν να ρυθμιστούν για να φαίνονται σκληρές ή λείες. Μπορούν εύκολα τα πολύγωνα να αντιπροσωπεύουν τόσο

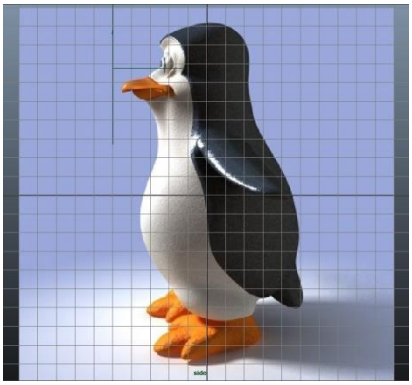
επίπεδες όσο και καμπύλες 3D μορφές. Κατά τη μοντελοποίηση πολυγώνων εργαζόμαστε με αυτά τα components συνεχώς.

Οι πολυγωνικές επιφάνειες έχουν ένα ευρύ φάσμα εφαρμογών και είναι ο προτιμώμενος τύπος επιφάνειας για πολλές εφαρμογές 3D, συμπεριλαμβανομένων διαδραστικών παιχνιδιών και την ανάπτυξη εφαρμογών web. Οι πολυγωνικές επιφάνειες μπορούν να περιγραφούν με το μικρότερο ποσό των δεδομένων όλων των τύπων 3D επιφανειών, για αυτό μπορούν να αποδοθούν γρήγορα, παρέχοντας αυξημένη ταχύτητα και διαδραστική απόδοση στον τελικό χρήστη σε παιχνίδια και άλλες εφαρμογές.

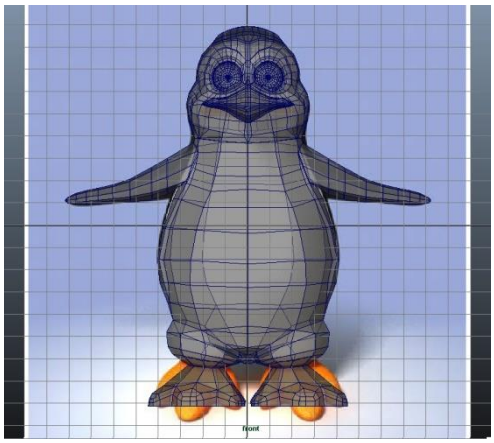
Η δημιουργία διαδραστικών χαρακτήρων ενός παιχνιδιού γίνεται χρησιμοποιώντας πολυγωνικές τεχνικές μοντελοποίησης επιφανειών μέσω των εργαλείων που προσφέρει το Maya. Χρησιμοποιούμε 2D επίπεδα εικόνων ως σημείο αναφοράς για την κατασκευή των 3D μοντέλων, και 3D primitives ως βάση για τη δημιουργία πιο σύνθετων μοντέλων. Δουλεύουμε με τα components του πλέγματος πολυγώνου (όψεις, ακμές και κορυφές), καθώς μπορούμε να τα επεξεργαστούμε με την τροποποιήση, κλιμάκωση, πρόσθεση, αφαίρεση, μετακίνηση, περιστροφή, ένωση και το διαχωρισμό. Στο τέλος ομαλοποιούμε το πλέγμα του πολυγώνου.

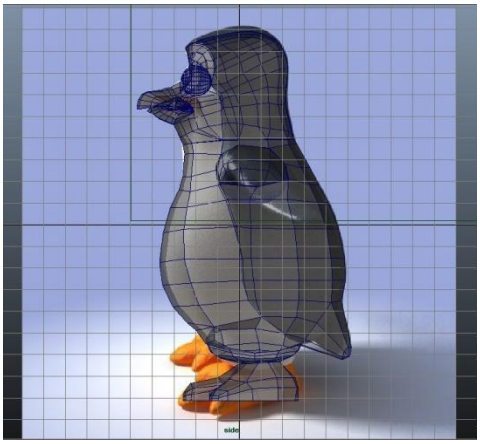
Μπορούμε να χρησιμοποιήσουμε διαφορετικές οπτικές γωνίες (front, side, και top views) των εικόνων, είτε είναι ζωγραφιές, ή σκίτσα ή φωτογραφίες, ώστε να απεικονίσουμε το 3D μοντέλο μας στο Maya. Οι εικόνες αυτές εισάγονται ως 2D αντικείμενα και συνδέονται με την κάμερα της σκηνής, και μπορούμε να τις επεξεργαστούμε. Επίσης μπορούν να χρησιμοποιηθούν για τη δημιουργία περιβάλλοντος background.





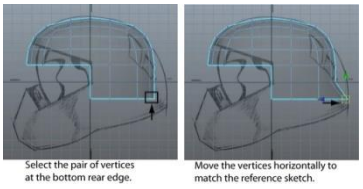
Ακολουθώντας τη βασική μεθοδολογία κατασκευής τρισδιάστατων χαρακτήρων, δημιουργήσαμε το κυρίως μέρος του πλέγματος του πιγκουίνου από έναν κύλινδρο primitive που παρείχε το Maya χρησιμοποιώντας τις εικόνες ως αναφορά. Τα primitives αντικείμενα είναι μια μέθοδος για την έναρξη δημιουργίας 3D πλεγμάτων, επειδή μπορούν να τροποποιηθούν για να δημιουργήσουμε άλλες μορφές. Μειώσαμε τις επιφάνειες του αρχικού αντικειμένου, ώστε να αποφύγουμε τις πολλές λεπτομέρειες, να κάνουμε πιο ελαφρύ το μοντέλο στην απόδοσή του, και να γίνει πιο απλή και εύκολη η τροποποίησή του. Το αρνητικό της απλοποίησης αυτής, είναι ότι προστέθηκαν πιο έντονες και απότομες γωνίες στο μοντέλο, οι οποίες μπορούν να αφαιρεθούν στο τέλος με τη διαδικασία της ομαλοποίησης.



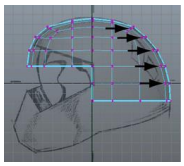


Κατά τη δημιουργία του μοντέλου μπορούμε να επωφεληθούμε την συμμετρία, δουλεύοντας μόνο στο μισό μέρος του μοντέλου και στην συνέχεια να το αντιγράψουμε και να το αντιστρέψουμε αντικατοπτρίζοντάς το ως προς τους άξονες YZ (mirror). Αυτό μπορεί να εξοικονομήσει χρόνο και προσπάθεια, και μας προσφέρει απόλυτη συμμετρία.

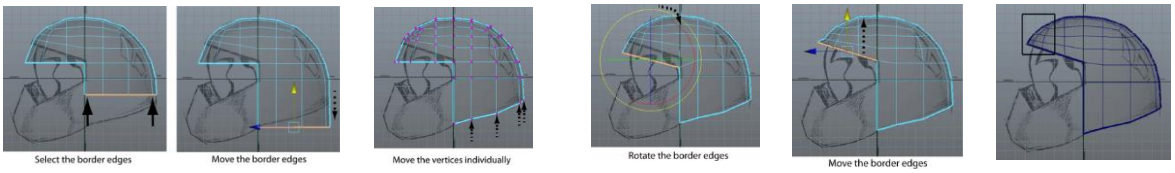
Η επιλογή των εξαρτημάτων(components) και ο μετασχηματισμός τους ώστε να ταιριάζουν με τις εικόνες αναφοράς είναι μια βασική μέθοδος για την επεξεργασία του σχήματος του πλέγματος πολυγώνου. Στην κάθε όψη ξεχωριστά, την πλαϊνή και την μπροστινή, επιλέγουμε τις κορυφές Vertex και



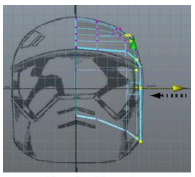
τις μετακινούμε ώστε να τις τοποθετούμε με τέτοιο τρόπο που να καλύπτουν το πλαίσιο της εικόνας αναφοράς και να ταιριάζουν με το σχήμα της. Αρχικά ξεκινάμε από την πλαϊνή όψη να δίνουμε σχήμα στο μοντέλο μας.



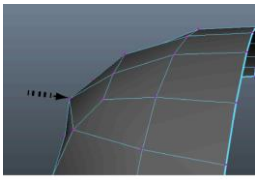
Μπορούμε επίσης να μετακινήσουμε και να περιστρέψουμε τις συνοριακές ακμές κατά μήκος του άκρου του πλέγματος και να προσαρμόσουμε πάλι τις κορυφές ώστε να δώσουμε το κατάλληλο σχήμα στο μοντέλο μας.



Ολοκληρώνοντας το σχήμα από την πλαϊνή όψη, συνεχίζουμε από την μπροστινή ακριβώς την ίδια διαδικασία, ακολουθώντας την εικόνα αναφοράς. Δουλεύουμε όμως στο μισό κομμάτι του μοντέλου από τη δεξιά μεριά καθώς μετά μπορούμε να το αντιγράψουμε και να το ενώσουμε κατάλληλα στο αρχικό κομμάτι. Μετακινώντας την κορυφή στο μισό κομμάτι του μοντέλου, μετακινείται αντίστοιχα και η συμμετρική της κορυφή κατά μήκος του άξονα συμμετρίας.

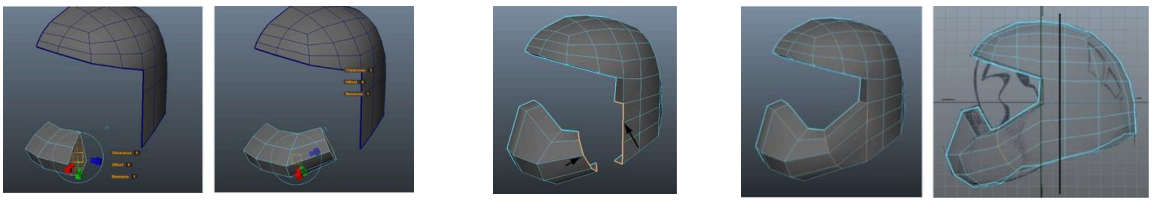


Στη συνέχεια δουλεύουμε στη προοπτική προβολή (perspective view) για να έχουμε μια πλήρη εικόνα και να ελέγξουμε τυχόν ατέλειες. Εκεί εξετάζουμε τις κορυφές και τις ακμές ώστε να είναι ομαλές για να δημιουργήσουν την καμπυλότητα του πλέγματος χωρίς ανεπιθύμητες αιχμές ή βυθίσεις και να έχουμε υψηλής ανάλυσης πλέγματος.

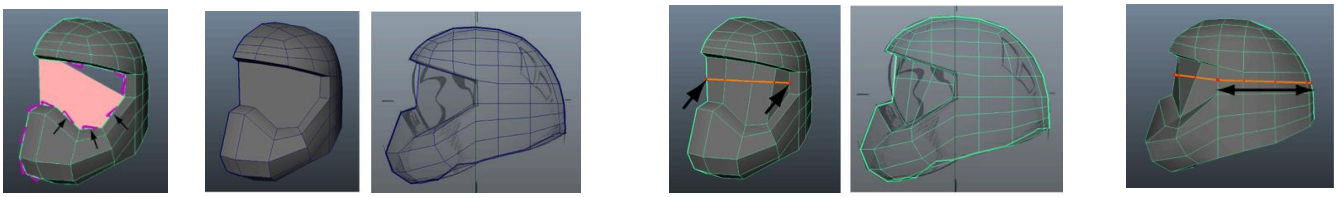


Εάν εξετάσουμε το μοντέλο στην προοπτική προβολή πλέγματος θα πρέπει να ταιριάζει στις εικόνες των σκίτσων αναφοράς. Το πλέγμα θα πρέπει επίσης να έχει μια σχετικά ομοιόμορφη κατανομή των όψεων πολυγώνου στο πλέγμα και οι βρόγχοι ακμών θα πρέπει να ρέουν ομαλά.

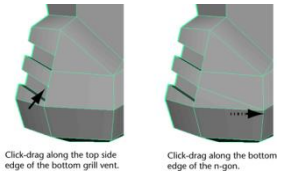
Μπορούμε να τοποθετήσουμε καινούργιες κορυφές στη περιοχή που θέλουμε ή μπορούμε ακόμα να δημιουργήσουμε νέα στοιχεία πολυγώνου από τα υπάρχοντα χρησιμοποιώντας τη δυνατότητα εξώθησης (Extrude). Όταν εξάγουμε ένα συστατικό πολυγώνου (για παράδειγμα, μια όψη, μια ακμή ή μια κορυφή), μπορούμε να δημιουργήσουμε πρόσθετα συστατικά πολυγώνου από αυτά που έχουμε επιλέξει. Επίσης μπορούμε να συνδέσουμε πλέγματα γεφυρώνοντας μία ή περισσότερες συνοριακές ακμές του πλέγματος με τη λειτουργία Bridge. Οι ακμές για να γεφυρωθούν πρέπει να βρίσκονται στο ίδιο πλέγμα πολυγώνου, δηλαδή θα πρέπει να ενώσουμε τα δύο πλέγματα σε ένα με τη λειτουργία Combine.



Για να καλύψουμε κενές επιφάνειες μπορούμε να δημιουργήσουμε όψεις τις οποίες θα χωρίσουμε με τέτοιο τρόπο ώστε να αποτελούνται από πολλαπλά πολύπλευρα πολύγωνα ανάλογα με την υπάρχουσα τετράπλευρη τοπολογία του πλέγματος για να ταιριάζουν. Ο χωρισμός γίνεται με το εργαλείο Split, με το οποίο σχεδιάζουμε γραμμές κατά μήκος των επιφανειών, που δείχνουν τη θέση της διάσπασης.

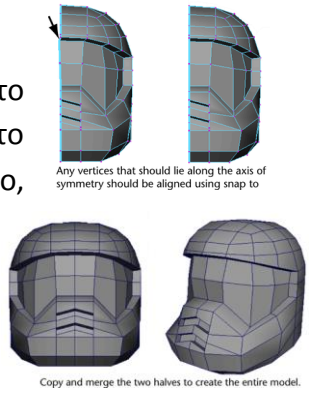


Όλες οι ακμές πρέπει να είναι αδιάσπαστες και συνεχόμενες σε ένα βρόγχο και να μην κόβονται.

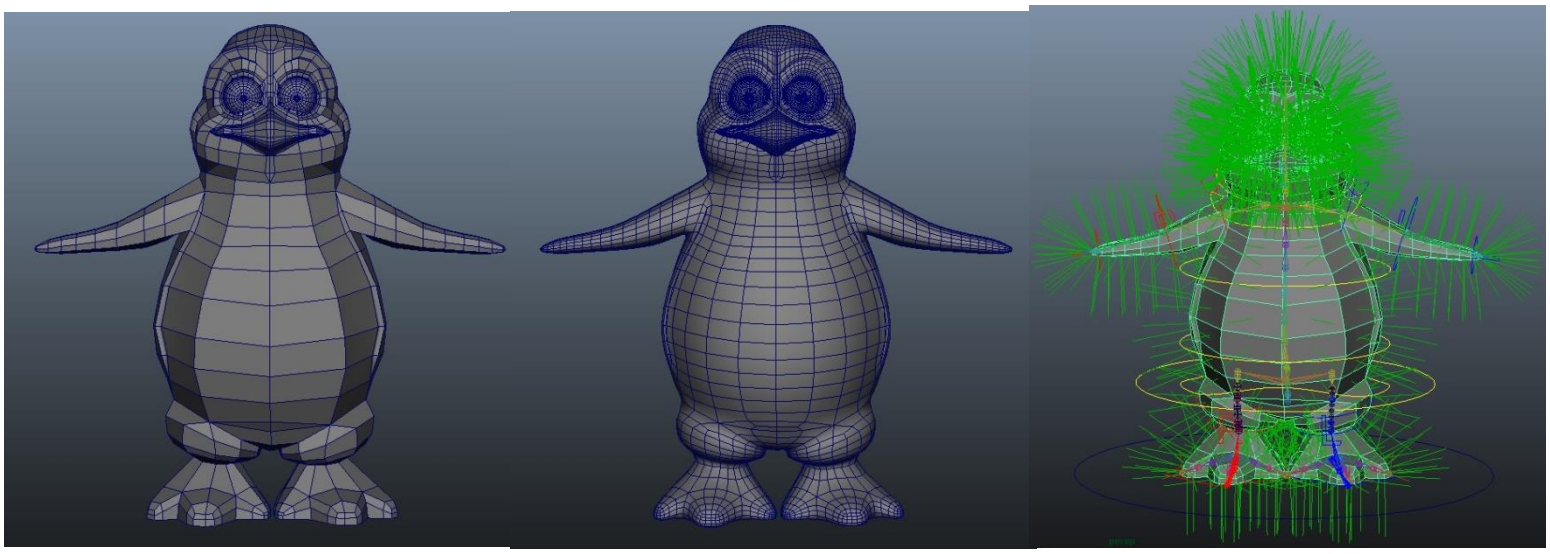


Το Maya κρατάει ιστορικό με όλες τις κινήσεις μας κατά τη διαδικασία δημιουργίας και επεξεργασίας του μοντέλου. Η διαγραφή του ιστορικού κατασκευής γίνεται συνήθως μόνο όταν ένα μοντέλο έχει ολοκληρωθεί και είναι έτοιμο να εισαχθεί στο παιχνίδι. Είναι σημαντικό να διαγράψουμε το ιστορικό του μοντέλου πριν την εισαγωγή του στη Unity για να καθαριστεί το αρχείο από άχρηστες πληροφορίες και να διαβαστεί σωστά από τη μηχανή παιχνιδιού.

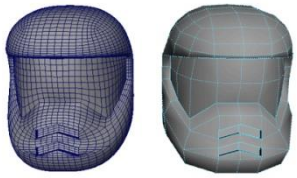
Μόλις οριστικοποιηθεί το ήμισυ του μοντέλου πολυγώνων, διαγράφουμε το ιστορικό κατασκευής του και δημιουργούμε το αντίθετο μισό αντιγράφοντας το πλάτος του άξονα συμμετρίας, έτσι ώστε να έχουμε το πλήρες μοντέλο, χρησιμοποιώντας τη Mirror Γεωμετρία. Πριν από την αντιγραφή του μοντέλου, ελέγχουμε ότι όλες οι συνοριακές ακμές βρίσκονται κατά μήκος του άξονα συμμετρίας.



Ανάλογα με την προβλεπόμενη χρήση του μοντέλου, μπορούμε να έχουμε μια έκδοση χαμηλής ανάλυσης, μια έκδοση υψηλής ανάλυσης, ή και τα δύο. Στο Maya, είναι εύκολο να αυξήσουμε την ανάλυση του μοντέλου χρησιμοποιώντας τη λειτουργία ομαλοποίησης Smooth.

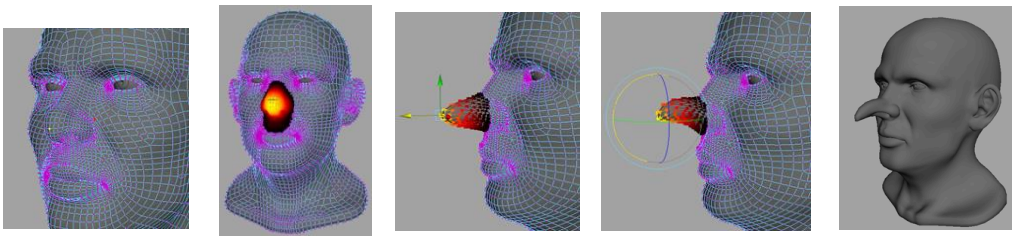


Μπορούμε να τσαλακώσουμε ή να σκληρύνουμε τις ακμές στα πολυγωνικά πλέγματα καθορίζοντας τον τρόπο που οι μεταβάσεις μεταξύ των επιφανειών ενισχύουν τον ρεαλισμό του μοντέλου. Όταν σκληρύνει μια ακμή σε ένα πλέγμα αλλάζει η κατεύθυνση των normals που συνδέονται με την κοινή ακμή, η οποία με τη σειρά της επηρεάζει τη σκίαση κατά μήκος αυτών των ακμών. Όταν τσακίσει μια ακμή ενός πλέγματος εξομαλύνεται έχοντας έτσι υψηλή ανάλυση.

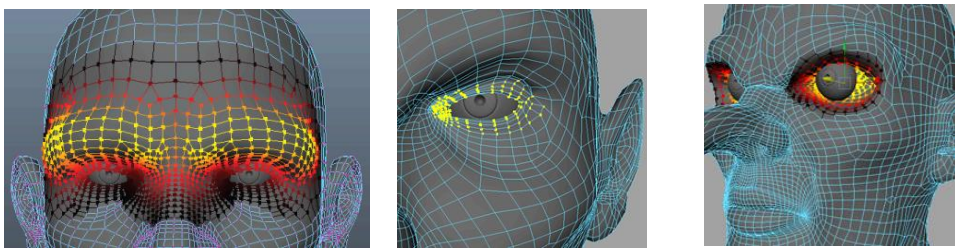


Turning on the Keep Hard Edge attribute displays the hardened edges on the smoothed version of the mesh.

Με το εργαλείο Soft και επιλέγοντας συστατικά-components με falloff σε ένα πολύγωνο ή μια επιφάνεια NURBS μπορούμε να παραμορφώσουμε μια περιοχή. Falloff είναι μια περιοχή γύρω από τα επιλεγμένα συστατικά που επηρεάζονται από μια μεταμόρφωση, σύμφωνα με ένα συντελεστή. Ο μετασχηματισμός συστατικών με falloff μας επιτρέπει να εκτελέσουμε μια ομαλή μετατροπή σε πολλαπλά συστατικά, χωρίς να χρειάζεται να ρυθμίσουμε κάθε συστατικό ξεχωριστά.



Με αυτή τη διαδικασία δημιουργήσαμε τα μάτια και φρύδια του πιγκουίνου μας.



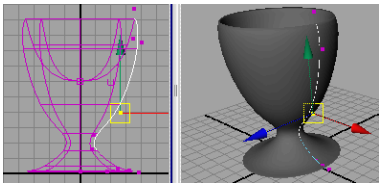
Στο Maya μπορούμε να δημιουργήσουμε πολύπλοκα πολυγωνικά μοντέλα με πολύ λίγες τεχνικές. Ξεκινώντας από μια primitive επιφάνεια, όπως έναν κύβο, μπορούμε να την επεξεργαστούμε με εξομαλύνση, κλιμάκωση, κίνηση, εξώθηση, διάσπαση, και περιστροφή των συστατικών της δημιουργώντας μια έκδοση χαμηλής ανάλυση του μοντέλου που θέλουμε να δημιουργήσουμε. Θα πρέπει να ρυθμίζουμε τις κορυφές για να τελειοποιήσουμε το σχήμα, και να εξομαλύνουμε τις άκρες μεταξύ των όψεων, όπου είναι επιθυμητό, ώστε να παραχθεί η τελική έκδοση του μοντέλου μας.

Σε διαδραστικά παιχνίδια οι πολυγωνικές επιφάνεις πρέπει να είναι μικρές με έναν μικρό αριθμό πολυγώνων. Το Μάγια έχει μια σειρά από εργαλεία για την ελαχιστοποίηση του αριθμού των πολυγωνικών έδρων ενός αντικειμένου, όπως το Reduce Tool. Λιγότερες όψεις (faces) σημαίνει απλούστερη γεωμετρία. Είναι σημαντικό για την επίτευξη αυξημένης διαδραστικής απόδοσης στις εφαρμογές παιχνιδιών να είναι λιγότερα τα πολύγωνα.

3.7.3 NURBS Modeling

Οι NURBS (Non-Uniform Rational B-splines) χρησιμοποιούν μια μέθοδο μαθηματικής περιγραφής των καμπυλών και επιφανειών που είναι κατάλληλες για εφαρμογές 3D. Οι NURBS χαρακτηρίζονται από τις ομαλές οργανικές μορφές που παράγουν.

Οι επιφάνειες NURBS μπορούν να μοντελοποιηθούν γρήγορα και η επεξεργασία γίνεται χρησιμοποιώντας μία ποικιλία τεχνικών. Δημιουργούνται με τη χρήση ενός ή περισσότερων NURBS καμπυλών που καθορίζουν το προφίλ του σχήματος που θέλουμε για μια επιφάνεια, και στη συνέχεια χρησιμοποιώντας μια ειδική μέθοδο κατασκευής δημιουργούμε την τελική επιφάνεια. Μία επιφάνεια μπορεί να μοντελοποιηθεί με NURBS και στη συνέχεια να μετατραπεί σε ένα πολυπλέγμα.



3.7.4 Animation

Το Maya μας επιτρέπει να εφαρμόσουμε δράση στα αντικείμενα της 3D σκηνής. Όταν ένα αντικείμενο ή κάποιο χαρακτηριστικό αλλάζει σε σχέση με το χρόνο αναφέρεται ως κινούμενο (animated). Το Maya προσφέρει μια μεγάλη ποικιλία από εργαλεία για να μας βοηθήσει να εμψυχώσουμε (animate) τα αντικείμενα στη σκηνή μας. Μπορούμε να χρησιμοποιήσουμε ένα συνδυασμό από διάφορες τεχνικές για να επιτύχουμε τα επιθυμητά αποτελέσματα.

Υπάρχουν κοινές τεχνικές και χαρακτηριστικά που αναδεικνύουν την τεχνολογία animation του Maya σε υπολογιστές όπως τα Keyframes και ο Graph Editor, τα Set Driven Key, το Path animation, και η Inverse kinematics.

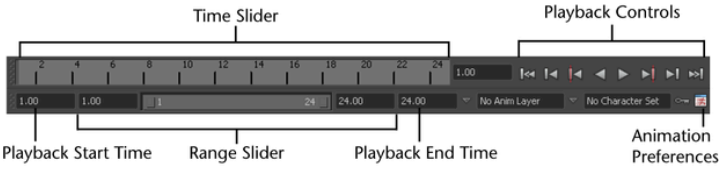
Keyframes και Graph Editor

Όταν ορίζουμε ένα βασικό καρέ ή κλειδί (keyframe ή key), μπορούμε να εκχωρήσουμε μια τιμή για την ιδιότητα ενός αντικειμένου (για παράδειγμα θέσης, περιστροφής, κλίμακας, χρώματος) σε μια συγκεκριμένη χρονική στιγμή.

Τα περισσότερα συστήματα κίνησης χρησιμοποιούν το πλαίσιο (frame) ως βασική μονάδα μέτρησης, καθώς κάθε πλαίσιο αναπαράγεται σε γρήγορη διαδοχή για να δώσει την ψευδαίσθηση της κίνησης.

Ο ρυθμός καρέ (frame rate), που είναι καρέ ανά δευτερόλεπτο, ο οποίος χρησιμοποιείται για την αναπαραγωγή ενός animation βασίζεται στο μέσο που θα αναπαραχθεί η κινούμενη εικόνα, για παράδειγμα τον κινηματογράφο, τη τηλεόραση, τα video game, και ούτω καθεξής.

Όταν ορίσουμε αρκετά κλειδιά σε διαφορετικές χρονικές στιγμές με διαφορετικές τιμές, το Maya δημιουργεί τις τιμές παραμέτρων μεταξύ εκείνων των χρόνων καθώς η σκηνή αναπαράγει κάθε καρέ. Το αποτέλεσμα είναι η κίνηση ή η αλλαγή αυτών των αντικειμένων ή των χαρακτηριστικών με την πάροδο του χρόνου.



Στο Time Slider μπορούμε να ορίσουμε πόσο χρόνο θα διαρκέσει το animation και εμφανίζει το εύρος αναπαραγωγής και τα κλειδιά που έχουμε ορίσει για ένα επιλεγμένο αντικείμενο. Τα κλειδιά εμφανίζονται ως κόκκινες γραμμές. Το κουτί στα δεξιά του Time Slider μας επιτρέπει να ορίσουμε το τρέχον πλαίσιο, δηλαδή το χρόνο, του animation .

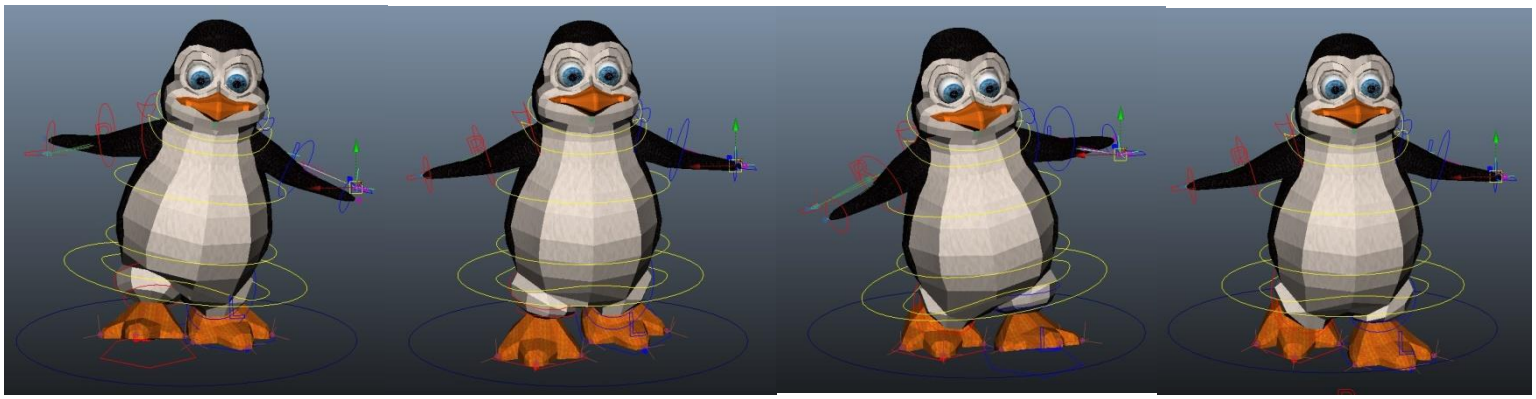
Οι Playback Controls ελέγχουν την αναπαραγωγή του animation, όπως τα πλήκτρα για να ξεκινήσει το παίζει (play) και να επιστρέψει στον χρόνο έναρξης (rewind). Το κουμπί παύσης (stop) εμφανίζεται μόνο όταν παίζει το animation. Το κουμπί Animation Preferences εμφανίζει ένα παράθυρο για τον καθορισμό των ρυθμίσεων προτίμησης του animation, όπως η ταχύτητα αναπαραγωγής. Το Range Slider ελέγχει το εύρος των πλαισίων που παίζουν όταν επιλέγουμε το κουμπί αναπαραγωγής .

Το εύρος αναπαραγωγής είναι ορισμένο σε μια σειρά πλαισίων (fames) από 1 έως 24. Με ταχύτητα αναπαραγωγής 24 καρέ ανά δευτερόλεπτο, η σκηνή μπορεί να παίξει για ένα δευτερόλεπτο. Στο Playback End Time box ορίζουμε το ρυθμό των πλαισίων. Ένα frame rate 24 καρέ ανά δευτερόλεπτο (fps) είναι η ταχύτητα καρέ που χρησιμοποιείται για κινηματογραφική ταινία. Για το βίντεο και τα παιχνίδια ο ρυθμός πλαισίων μπορεί να είναι 30 fps (NTSC) ή 25 fps (PAL) ανάλογα με τη μορφή που χρησιμοποιείται. Με μια σειρά αναπαραγωγής από 1 έως 72, είναι σε θέση να δημιουργήσει κινούμενα σχέδια τριών δευτερολέπτων (72 καρέ διαιρούμενο με 24 καρέ ανά δευτερόλεπτο = 3 δευτερόλεπτα).

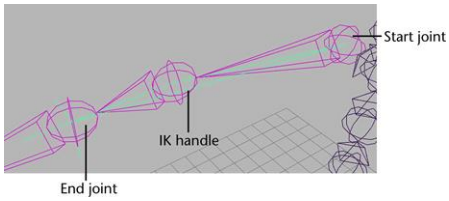
Μπορούμε να χρησιμοποιήσουμε τα βασικά καρέ για να ορίσουμε τις θέσεις έναρξης και λήξης της κίνησης. Ορίζουμε ένα κλειδί σε ένα καρέ για όλες τις παραμέτρους των χαρακτηριστικών X, Y, Z κίνησης του μοντέλου. Ανάμεσα σε δύο ορισμένα κλειδιά το Maya δημιουργεί κίνηση μεταξύ των θέσεων. Από προεπιλογή, η κινούμενη εικόνα παίζει σε ένα βρόχο από το πλαίσιο 1 έως το 72. Για μια ομαλή κίνηση πρέπει να ορίσουμε και τα ενδιάμεσα κλειδιά κίνησης μεταξύ της αρχής και του τέλους του animation.

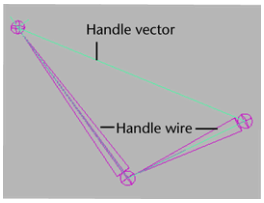
Χρειάζεται να ορίσουμε μόνο τα βασικά βήματα των κινήσεων που θέλουμε να αποδώσουμε και να έχουμε μεγάλες αποστάσεις μεταξύ των κλειδιών μας. Τα ενδιάμεσα βήματα μεταξύ των κλειδιών που ορίζουμε, αποδίδονται από το Maya αυτόματα, αναγνωρίζοντας τη θέση που βρίσκεται το μοντέλο και την θέση στην οποία κατευθύνεται, δημιουργώντας μια ομαλή και φυσική κίνηση.

Μπορούμε να φτιάξουμε ένα animation για οποιοδήποτε χαρακτηριστικό του μοντέλου, όχι μόνο για τη θέση, τη περιστροφή, και την κλίμακα, αλλά και για την ένταση του φωτός, τη διαφάνεια της επιφάνειας, τη περιστροφή της κάμερας, ή τη θέση των CVs. Μπορούμε επίσης να κάνουμε αποκοπή, αντιγραφή, επικόλληση και διαγραφή κλειδιών απευθείας στο Time Slider για να επεξεργαστούμε το animation.



Inverse Kinematics (IK)



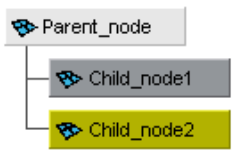


Η Αντίστροφη Κινηματική (IK) μας επιτρέπει να θέσουμε αποτελεσματικές πόζες στα μοντέλα και τους χαρακτήρες μας για τα κινούμενα σχέδια. Δίνουμε πόζα στο σκελετό με την τοποθέτηση IK λαβών, που συνήθως βρίσκονται στο τέλος μιας IK αλυσίδας αρθρώσεων (joint chain), για παράδειγμα στο χέρι. Οι περιστροφές για τις άλλες αρθρώσεις υπολογίζονται αυτόματα από έναν IK solver. Ένας IK solver υπολογίζει τις περιστροφές κάθε ένωσης πίσω στην ιεραρχία της αλυσίδας αρθρώσεων με βάση τη θέση της λαβής IK.

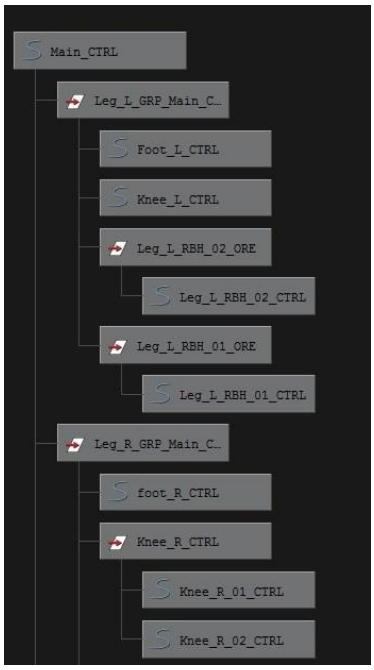
Αντίθετα, η Εμπρόσθια Κινηματική (Forward Kinematics-FK) απαιτεί να περιστρέψουμε κάθε άρθρωση με τη σειρά μέχρι να έχουμε την επιθυμητή πόζα για ένα βασικό καρέ. Όταν έχουμε ένα σύνθετο σκελετό, η διαδικασία αυτή είναι κουραστική. Είναι πιο γρήγορο να δημιουργήσουμε ένα χαρακτήρα χρησιμοποιώντας Αντίστροφη Κινηματική επειδή οι κοινές περιστροφές που βρίσκονται πιο ψηλά στην ιεραρχία υπολογίζονται αυτόματα με βάση την τοποθέτηση της IK λαβής. Η χρήση και των δύο FK και IK τεχνικών για τον ίδιο σκελετό ονομάζεται ανάμειξη FK / IK.

Η IK είναι χρήσιμη για τη δημιουρία πόζας σε animated χαρακτήρες που έχουν ένα σκελετό (δύο πόδια, τέσσερα πόδια και ούτω καθεξής) και σε άλλες εφαρμογές όπως σκελετούς μέσα σε μηχανικά μοντέλα. Για τη δημιουργία IK ακολουθούμε βασικές διαδικασίες οι οποίες είναι οι εξής:

- Κατασκευή ενός μοντέλου ιεραρχίας για να παρέχουμε τις σχέσεις μεταξύ των συστατικών για την κίνηση της πόζας και τα βασικά καρέ.
- Ο συνδυασμός των συστατικών ενός 3D μοντέλου σε μια ιεραρχική ομαδοποίηση.
- Την κατασκευή ενός ενιαίου σκελετού αλυσίδας για να κινήσουμε το μοντέλο χρησιμοποιώντας Αντίστροφη Κινηματική IK. Συχνά οι χαρακτήρες είναι με πολλαπλές αλυσίδες αρθρώσεων.
- Το συνδυασμό του σκελετού με το μοντέλο δημιουργώντας μια σχέση parenting μεταξύ των συστατικών του μοντέλου στην ιεραρχία σκελετού. Parenting είναι μια μέθοδος για τον συνδυασμό ενός μοντέλου σε μια ιεραρχία σκελετού. Όταν χρησιμοποιούμε χαρακτήρες που έχουν χαρακτηριστικά κάμψης ή παραμόρφωσης, μια πιο συνηθισμένη τεχνική είναι η χρήση του δέρματος.
- Τη δημιουργία ενός συστήματος IK που επιτρέπει την ιεραρχία του σκελετού να δημιουργεί μια πόζα. Συχνά οι χαρακτήρες έχουν πολλαπλές αλυσίδες αρθρώσεων και πολλαπλές IK λαβές για να τις ελέγχουν.
- Τη δημιουργία ενός αντικειμένου ελέγχου για το χειρισμό του IK συστήματος. Τα αντικείμενα ελέγχου απλοποιούν την επεξεργασία του χαρακτήρα.
- Τη χρήση τριών τύπων περιορισμών (Point, Orient, Parent) για τον έλεγχο της θέσης και του προσανατολισμού των συστατικών εντός του συστήματος IK.
- Τον περιορισμό του εύρους της κίνησης του συστήματος IK. Περιορίζεται ο μετασχηματισμός του αντικειμένου ελέγχου, προκειμένου να περιοριστεί η κίνηση του IK. Δεν θέτουμε όρια για την περιστροφή ενός συστήματος IK καθώς περιορίζει το πώς ο solver υπολογίζει τη θέση των αρθρώσεων.
- Τη δημιουργία της Pose και τον ορισμό των βασικών καρέ για το σύστημα IK.
- Τη δημιουργία του animation ενός αντικειμένου περιορίζοντας την κίνησή του σε δύο ή περισσότερα αντικείμενα.



Η Inverse Κινηματική στηρίζεται στις ιεραρχικές σχέσεις μεταξύ των συστατικών ενός μοντέλου και του συστήματος IK για να δημιουργήσει και να εμψυχώσει τα στοιχεία. Μια ιεραρχία αποτελείται από μια σειρά κόμβων που συνδυάζονται δημιουργώντας μια σχέση για κάποιο σκοπό. Οι μεμονωμένοι κόμβοι μπορούν να αντιπροσωπεύουν τις επιφάνειες ενός μοντέλου, ένα δίκτυο υφών που περιγράφουν ένα υλικό σκίασης ή μια σειρά αρθρώσεων σε ένα σκελετό. Οι ιεραρχίες είναι δομημένες με έναν top-down τρόπο, με έναν κόμβο στο πάνω μέρος (ο κόμβος γονέας ή κόμβος ρίζα) και άλλους κόμβους (κόμβους ή κόμβους φύλλα) που επισυνάπτονται και συνδέονται κάτω από τον ανώτερο κόμβο.



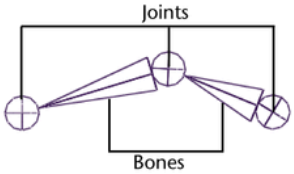
Οι ιεραρχίες μας επιτρέπουν να δημιουργήσουμε πολύπλοκες δομές με τις σχέσεις μεταξύ των components. Για παράδειγμα, για να φτιάξουμε το animation σε ένα ιεραρχικό μοντέλο, πρέπει απλά να επιλέξουμε και να μετακινήσουμε το γονικό κόμβο της ιεραρχίας και κινείται και το υπόλοιπο του μοντέλου (κόμβοι παιδί). Ανεξάρτητα από το αν κινήσουμε το μοντέλο με FK ή IK, ή ένα συνδυασμό των δύο μεθόδων, η καλύτερη προσέγγιση περιλαμβάνει τη δημιουργία μιας ιεραρχικής δομής.

Η δημιουργία μιας ιεραρχίας για το μοντέλο διασφαλίζει ότι τα συστατικά της ιεραρχίας μπορούν να περιστρέφονται σαν να ήταν μία μονάδα, με ένα κοινό σημείο περιστροφής που δημιουργείται στον κόμβο γονέα. Επίσης διαφαλίζεται ότι όταν ένα συγκεκριμένο τμήμα του μοντέλου είναι σε πόζα και είναι keyframed, τα στοιχεία που είναι χαμηλότερα στην ιεραρχία επίσης είναι στην ίδια πόζα και είναι και αυτά keyframed.

Με το εργαλείο Hypergraph μπορούμε να δούμε την ιεραρχία του μοντέλου, το οποίο είναι ένα πρόγραμμα επεξεργασίας διαχείρισης της σκηνή που παρουσιάζει μια γραφική άποψη της ιεραρχίας της σκηνής. Μας επιτρέπει να διαχειριστούμε πιο εύκολα την ιεραρχία των αντικειμένων και των κόμβων στη σκηνή μας. Τα κουτιά αντιπροσωπεύουν τους κόμβους για τα διάφορα αντικείμενα στη σκηνή. Ορισμένα στοιχεία στη σκηνή έχουν ήδη σχέσεις parenting σε μια ιεραρχία. Αυτό υποδεικνύεται από τις γραμμές που συνδέουν τους κόμβους.

Δημιουργία μιας ιεραρχίας σκελετού

Για να συνδέσουμε τα διάφορα συστατικά του μοντέλου σε μια ιεραρχία για να δημιουργήσουμε μια πόζα, θα πρέπει να δημιουργήσουμε έναν σκελετό. Ο σκελετός είναι μια ιεραρχία των αρθρώσεων που συνδέονται μεταξύ τους με τα οστά.



Στη σκηνή, μια άρθρωση-joint αντιπροσωπεύει έναν ειδικό τύπο κόμβου που δημιουργείται σε μια ιεραρχία σκελετού. Μια άρθρωση ενεργεί ως κόμβος γονέας για οποιεσδήποτε άλλες αρθρώσεις που είναι στην ιεραρχία κάτω από

αυτό. Κάθε άρθρωση έχει ένα pivot σημείο περιστροφής που συνδέεται με αυτό. Ένα κόκαλο είναι η οπτική αναπαράσταση που χρησιμοποιείται στην σκηνή για να συνδέσουμε τις αρθρώσεις και βοηθάει στην απεικόνιση της αλυσίδας των αρθρώσεων.

Ένας σκελετός είναι παρόμοιος με ένα σκελετό στον πραγματικό κόσμο καθώς ενεργεί ως υποκείμενη δομή για τις επιφάνειες που επισυνάπτονται. Ενώ μπορούμε να δούμε τον σκελετό με τα οστά και τις αρθρώσεις του στην σκηνή, δεν εμφανίζεται στις εικόνες απόδοσης. Σκοπός του είναι να μας βοηθήσει στη δημιουργία και το στήσιμο των μοντέλων και των χαρακτήρων μας και να απεικονίσει την κίνηση που θέλουμε να πετύχουμε. Οι σκελετοί αποτελούν αναπόσπαστο μέρος της κίνησης του κάθε χαρακτήρα ή του ιεραρχικού μοντέλου.

Parenting είναι ο όρος που χρησιμοποιείται κατά την τοποθέτηση ενός κόμβου κάτω από ένα άλλο σε μια ιεραρχία έτσι ώστε να γίνεται το παιδί του κόμβου πάνω από αυτό (του γονέα του) στην ιεραρχία.

Μετά τη δημιουργία του σκελετού πρέπει να συνδέσουμε τις επιφάνειες του μοντέλου στην ιεραρχία σκελετού ώστε να μπορούν να τοποθετηθούν σε πόζα και να φτιάξουμε το animation του μοντέλου. Η τεχνική για το συνδυασμό ενός μοντέλου σε μια ιεραρχία σκελετού αναφέρεται ως skinning ή binding. Το Bind Skin tool συνδυάζει τα επιφανειακά συστατικά και τις αρθρώσεις μαζί σε μια ιεραρχία και επίσης λαμβάνει υπόψη τυχόν παραμορφώσεις που μπορεί να συμβούν όταν ο χαρακτήρας λυγίζει ή κάμπτεται, όπως στα γόνατα, τη μέση, τον αυχένα, τους αγκώνες και ούτω καθεξής.

Αφού δημιουργήσουμε την ιεραρχία του σκελετού και τις συνδέσεις parenting, μπορούμε να επιλέξουμε και να περιστρέψουμε τις αρθρώσεις του σκελετού για να φτιάξουμε την πόζα, και στη συνέχεια να ορίσουμε βασικά καρέ για το animation. Το στήσιμο της πόζας ενός χαρακτήρα χρησιμοποιώντας τις περιστροφές των αρθρώσεων αναφέρεται ως Forward Κινηματική (FK). Η FK επιτρέπει τον ακριβή έλεγχο κατά τη δημιουργία μιας πόζας, αλλά μπορεί να είναι μια χρονοβόρα διαδικασία στην οποία επιλέγουμε και περιστρέφουμε κάθε άρθρωση στην επιθυμητή θέση.

Όταν ένα animation είναι περισσότερο επικεντρωμένο στην τοποθέτηση της τελικής άρθρωσης μιας ιεραρχίας σε μια συγκεκριμένη θέση, είναι πιο αποτελεσματικό να χρησιμοποιήσουμε την Αντίστροφη Κινηματική για να φτιάξουμε την πόζα του σκελετού. Πρέπει να διαλέξουμε την τεχνική που θα χρησιμοποιήσουμε αναλόγως το αποτέλεσμα που θέλουμε να επιτύχουμε και κατά πόσο μας εξυπηρετεί η κάθε τεχνική με βάση τις επιλογές που παρέχει. Συχνά καταλήγουμε στο συνδυασμό των τεχνικών animation.

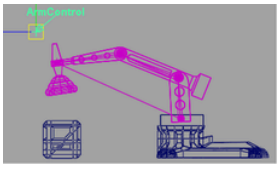
Η IK μας επιτρέπει να ελέγχουμε τις περιστροφές των αρθρώσεων σε ένα σκελετό χρησιμοποιώντας μια λαβή IK. Η πόζα στο σκελετό γίνεται τοποθετώντας τη λαβή IK, που

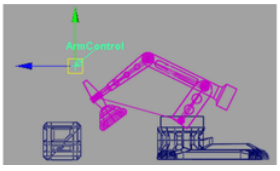
συνήθως βρίσκεται στο τελευταίο άκρο του σκελετού, και οι περιστροφές για τις άλλες αρθρώσεις υπολογίζονται αυτόματα από τον IK solver. Η χρήση της λαβής IK για τη δημιουργία μιας αλυσίδας IK δεν είναι καλή τακτική. Η λαβή IK μπορεί να είναι δύσκολο να επιλεχθεί στη σκηνή, ειδικά όταν είναι ομαδοποιημένη σε μια ιεραρχία που περιλαμβάνει ένα σκελετό και άλλα στοιχεία της επιφάνειας. Η καλύτερη τακτική είναι η δημιουργία ενός αντικειμένου ελέγχου. Μπορούμε να δημιουργήσουμε ένα αντικείμενο ελέγχου για να επιλέξουμε και να χειριστούμε ένα σύστημα IK αντί να χρησιμοποιούμε απευθείας τη λαβή IK. Μπορούμε να δημιουργήσουμε ένα αντικείμενο ελέγχου με μια καμπύλη ή έναν locator, τα οποία χρησιμοποιούνται συχνά επειδή δεν εμφανίζονται στην εικόνα που αποδίδεται. Το αντικείμενο ελέγχου ελέγχει την κίνηση της λαβής IK χρησιμοποιώντας έναν περιορισμό. Μπορούμε να περιορίσουμε τη θέση, τον προσανατολισμό, ή την κλίμακα ενός αντικειμένου σε άλλα αντικείμενα που χρησιμοποιούν περιορισμούς. Για να δημιουργήσουμε ένα αντικείμενο ελέγχου ακολουθούμε ορισμένες τυπικές διαδικασίες:

- Δημιουργία και τοποθέτηση του αντικειμένου ελέγχου στην σκηνή.
- Ονομασία του κόμβου για το αντικείμενο ελέγχου στο Hypergraph.
- Πάγωμα των μετασχηματισμών για το αντικείμενο του ελέγχου.
- Ονομασία του αντικειμένου ελέγχου στην σκηνή.
- Περιορισμός της IK λαβής με το αντικείμενο ελέγχου.

Πριν συνδέσουμε την IK λαβή με το αντικείμενο του ελέγχου, πρέπει να παγώσουμε τους μετασχηματισμούς του αντικειμένου ελέγχου. Το πάγωμα των μετασχηματισμών μηδενίζει τις τιμές των μετασχηματισμών για ένα αντικείμενο χωρίς να αλλάξει η θέση του αντικειμένου. Με αυτόν τον τρόπο μπορούμε να επανερχόμαστε στην αρχική θέση του αντικειμένου μηδενίζοντας τις τιμές των σχετικών παραμέτρων. Για να εντοπίζουμε εύκολα τα αντικείμενα ελέγχου τα επισημαίνουμε με μια ετικέτα σχολιασμού.

Μπορούμε να περιορίσουμε την IK λαβή με το αντικείμενο του ελέγχου (πχ ArmControl) χρησιμοποιώντας ένα σημείο περιορισμού. Ένα σημείο περιορισμού επιτρέπει τα χαρακτηριστικά του μετασχηματισμού ενός αντικειμένου να ελέγχονται από τους μετασχηματισμούς ενός άλλου αντικειμένου. Για παράδειγμα, μπορούμε να επιλέξουμε και να τοποθετήσουμε έναν ελεγκτή χεριού ArmControl χωρίς να χρειάζεται να αγγίξουμε τη λαβή IK. Μπορούμε επίσης να ρυθμίσουμε την απόσταση (offset) που θα έχει το αντικείμενο ελέγχου από την την λαβή IK, με την οποία εξασφαλίζουμε ότι η λαβή IK δεν κινείται στην ίδια θέση με το αντικείμενο του ελέγχου.





Όταν σύρουμε το αντικείμενο ελέγχου κινείται το IKHandle. Επειδή η χειρολαβή IK ελέγχει το σκελετό και το μοντέλο (τμήμα του μοντέλου), κινηνούνται επίσης και αυτά. Η απόσταση μεταξύ του αντικειμένου ελέγχου και της λαβής IK διατηρείται.

Ο προσανατολισμός του αντικειμένου βασίζεται στην περιστροφή που είχε στην αρχική του θέση, ανεξάρτητα από τον προσανατολισμό των άλλων συστατικών που περιστρέφονται από το αντικείμενο ελέγχου. Ένας περιορισμός orient περιορίζει τον προσανατολισμό (περιστροφή) των X, Y, και Z αξόνων ενός αντικειμένου ώστε να ταιριάζουν με εκείνα του αντικειμένου στόχου.

Είναι δυνατόν να κινήσουμε το αντικείμενο ελέγχου ώστε το τμήμα του μοντέλου που ελέγχει να εκτείνεται πλήρως σε ευθεία θέση ή να προσανατολίζεται σε άλλες θέσεις που δεν θέλουμε απαραίτητα. Μπορούμε να περιορίσουμε το εύρος της κίνησης του αντικειμένου ελέγχου για να διασφαλιστεί ότι το σύστημα IK θα δημιουργήσει μια πόζα με ένα προβλέψιμο τρόπο. Ο περιορισμός της κίνησης για το σύστημα IK περιλαμβάνει κάποια βασικά βήματα: Κλειδώνουμε τους μετασχηματισμούς του αντικειμένου ελέγχου, έτσι ώστε να μπορεί να κινηθεί μόνο σε Y και Z άξονες. Κλειδώνοντας τον X άξονα έτσι ώστε να μην μπορεί να επιλεγεί ή να τροποποιηθεί, το τμήμα που επηρεάζει το αντικείμενο ελέγχου δεν είναι σε θέση να μετακινηθεί από τη μία πλευρά στην άλλη. Κλειδώνουμε τις περιστροφές στη βάση περιστροφής του αντικειμένου ελέγχου έτσι ώστε να περιστρέφεται το μοντέλο μόνο γύρω από τον άξονά του Y, και όχι γύρω από τους X ή Z άξονες. Περιορίζουμε τους μετασχηματισμούς για το αντικείμενο ελέγχου, ώστε το μοντέλο να μην μπορεί να επεκταθεί πλήρως σε μια ευθυγραμμισμένη θέση. Ο αριθμός των κόμβων στην ιεραρχία του συστήματος IK μπορεί να είναι πολύ μεγάλος και η δημιουργία της πόζας αποδοτική και αποτελεσματική. Μπορούμε να απλοποιήσουμε την απεικόνιση των κόμβων με την κατάρρευση τμημάτων της ιεραρχίας. Έπειτα από όλες αυτές τις διαδικασίες το σύστημα IK είναι rigged και έτοιμο για τη δημιουρία κινουμένων σχεδίων.

Αν θέλουμε να περιορίσουμε κάποιες δυνατότητες στο μοντέλο του ελεγχτή του αντικειμένου, μπορούμε να χρησιμοποιήσουμε έναν γονέα περιορισμού. Όταν ένας γονέας περιορισμού εφαρμόζεται σε ένα αντικείμενο, το αντικείμενο περιορίζεται και συμπεριφέρεται σαν να ήταν ένας κόμβος παιδί του περιοριστικού αντικειμένου.

Ένα αντικείμενο μπορεί να έχει πολλούς περιορισμούς που εφαρμόζονται σε αυτό. Η επίδραση του κάθε περιορισμού στο αντικείμενο μπορεί να αλλάξει τροποποιώντας το χαρακτηριστικό του βάρους για τον γονέα περιορισμού. Το χαρακτηριστικό βάρους είναι μια animatable παράμετρος που μπορεί να ρυθμιστεί για να είναι ενεργοποιημένη ή απενεργοποιημένη σε συγκεκριμένα πλαίσια. Για να ρυθμίσουμε τους γονείς περιορισμών τοποθετούμε το αντικείμενο στις θέσεις όπου η επιρροή αρχίζει ή τελειώνει, και στη συνέχεια ορίζουμε του περιορισμούς του γονέα.

Ολοκληρώνοντας τις παραπάνω ρυθμίσεις μπορούμε να φτιάξουμε την πόζα που θέλουμε και να ορίσουμε τα keyframes. Η σειρά κινουμένων σχεδίων συμβαίνει κατά τη διάρκεια μιας περιόδου 180 πλαισίων.

3.7.5 Character Setup

Ένα τυπικός 3D χαρακτήρας μπορεί να αποτελείται από πολλές επιφάνειες και συστατικά. Για να εξασφαλιστεί ότι ο χαρακτήρας θα κινηθεί με τον τρόπο που θέλουμε σχεδιάζουμε προσεκτικά τη διαδικασία της εγκατάστασης του χαρακτήρα. Character setup ή rigging είναι ο γενικός όρος που χρησιμοποιείται για την παρασκευή των 3D μοντέλων με τις αρθρώσεις και τους σκελετούς για τα κινούμενα σχέδια.

Ανάλογα με το μοντέλο που κινείται, η εγκατάσταση χαρακτήρων μπορεί να περιλαμβάνει τις ακόλουθες τεχνικές :

- Δημιουργία ενός σκελετού με κόκαλα και αρθρώσεις που λειτουργεί ως πλαίσιο (framework) για το 3D μοντέλο του χαρακτήρα. Ορίζουμε τα όρια στις αρθρώσεις, ώστε να περιστρέφεται με πειστικό τρόπο. Όταν φτιάχνουμε το animation του χαρακτήρα, τοποθετούμε τις αρθρώσεις του χρησιμοποιώντας είτε προς την εμπρόσθια ή την αντίστροφη κινηματική τεχνική (FK ή IK) .
- Δέσμευση (Binding) των 3D επιφανειών στο σκελετό, έτσι ώστε να κινούνται μαζί. Η διαδικασία της δέσμευσης μπορεί επίσης να περιλαμβάνει τον καθορισμό του τρόπου με τον οποίο οι αρθρώσεις του χαρακτήρα λυγίζουν ή πώς οι επιφάνειες του δέρματος διογκώνονται για την προσομοίωση των μυών.
- Ορισμός και περιορισμός συγκεκριμένων κινούμενων χαρακτηριστικών, προκειμένου να περιοριστεί το εύρος της κίνησης ή να ελεγχθεί ένα χαρακτηριστικό με βάση την κίνηση του άλλου .
- Ομαδοποίηση των συστατικών της επιφάνειας, όπως σε CVs σύνολα που ονομάζονται clusters έτσι ώστε τα μέρη του χαρακτήρα να είναι κινούμενα σε ένα πιο λεπτομερές επίπεδο .

Τα πιο κοινά χαρακτηριστικά εγκατάστασης χαρακτήρα είναι η δημιουργία σκελετού και χρήση της κινηματικής, ομαλό skinning, Cluster και η μίξη σχημάτων deformers.

Skeletons and kinematics

Μετά την μοντελοποίηση των επιφανειών που συνθέτουν έναν άνθρωπο, ζώο ή άλλο χαρακτήρα, δημιουργούμε ένα σκελετό και τον συνδέουμε με την επιφάνεια του χαρακτήρα, που ονομάζεται δέρμα.

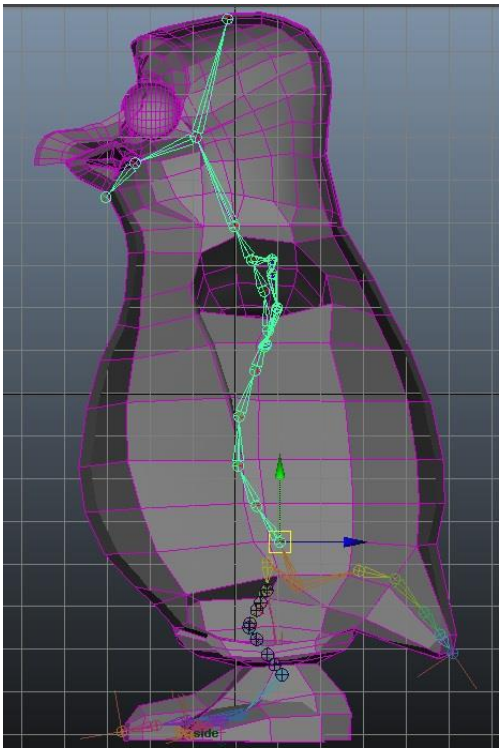
Ένας σκελετός παρέχει μια δομή για την εμψύχωση του χαρακτήρα. Όταν δημιουργούμε ένα σκελετό στο Maya, μπορούμε να δημιουργήσουμε μια σειρά από οστά με τις αρθρώσεις στις θέσεις του σκελετού που θέλουμε τον χαρακτήρα να λυγίζει ή να στρίβει. Μπορούμε να φτιάξουμε το animation ενός σκελετού χωρίς να έχει δέρμα. Οι βασικές τεχνικές για τη δημιουργία ενός απλού σκελετό για ένα ανθρώπινο χαρακτήρα περιλαμβάνουν :

- Δημιουργία συνδέσμων για ένα χαρακτήρα.
- Ονομασία των αρθρώσεων στο Hypergraph.
- Χρήση της συμμετρίας κατά τη δημιουργία του χαρακτήρα.
- Αρθρώσεις γονέα στο σκελετό.
- Χρήση τεχνικών Αντίστροφης Κινηματικής (IK) για τη δημιουργία του σκελετού.
- Δέσμευση ενός χαρακτήρα στο σκελετό.

Δημιουργία Joints

Ένας σκελετός αποτελείται από οστά και αρθρώσεις, με τους συνδέσμους να βρίσκονται στις θέσεις του σκελετού που θέλουμε ο χαρακτήρας να λυγίζει ή να στρίβει. Μια κοινή τεχνική για τη δημιουργία ενός σκελετού είναι να δημιουργήσουμε διάφορες ανεξάρτητες αλυσίδες αρθρώσεων, μία για κάθε χέρι, μια για κάθε πόδι, μια για την σπονδυλική στήλη/κεφάλι, και στη συνέχεια ομαδοποιούμε τις αλυσίδες μαζί για να δημιουργήσουν μια ενιαία ιεραρχία σκελετού. Επίσης πρέπει να προσέχουμε ώστε οι αρθρώσεις να έχουν κλίση προς την κατεύθυνση που θέλουμε να κινούνται τα οστά.

Στο Hypergraph, που δείχνει τα ονόματα των αρθρώσεων, μπορούμε να επιλέξουμε, να μετονομάσουμε, και να ορίσουμε γονείς-αντικείμενα. Είναι παρόμοιο με το Outliner, αλλά έχει τα χαρακτηριστικά προσαρμοσμένα για την εγκατάσταση του χαρακτήρα και απεικονίζει όλες τις σχέσεις γονέα-παιδιού σε μια εύκολη μορφή. Οι αρθρώσεις έχουν ιεραρχική σχέση, με το πρώτο joint στην ιεραρχία είναι η ρίζα και ο γονέας όλων των παιδιών, και το κάθε παιδί είναι ο γονέας των επόμενων παιδιών στην αλυσίδα. Εάν αλλάξουμε τη θέση της ρίζας, θα αλλάξει η θέση ολόκληρης της αλυσίδα άρθρωσης.



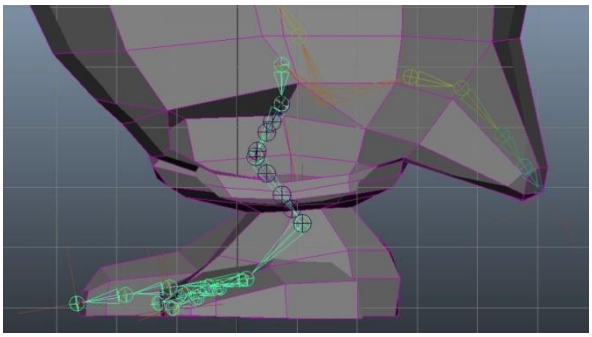
Οι αρθρώσεις εκπροσωπούνται από σφαιρικά εικονίδια. Τα οστά διαχωρίζουν τις αρθρώσεις και αντιπροσωπεύονται από επιμηκυμένα εικονίδια πυραμίδας. Το στενό μέρος ενός οστού δείχνει προς την κάτω κατεύθυνση της ιεραρχίας.

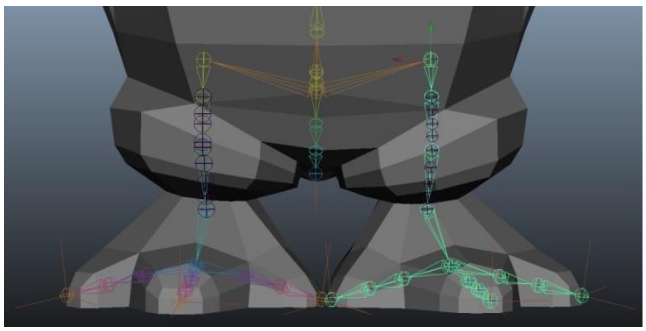
Η τοποθέτηση του σκελετού γίνεται αναλόγως το σχήμα του μοντέλου και τα μέρη του σώματος που μας ενδιαφέρουν να κινούνται. Φτιάχνουμε πρώτα το σκελετό από την πλαινή όψη ακολουθώντας το σχήμα του χαρακτήρα, και έπειτα ελέγχουμε τη θέση του από την μπροστινή όψη καθώς και την όψη από πάνω, και τον μετακινούμε ώστε να εφαρμόζει στο σχήμα.

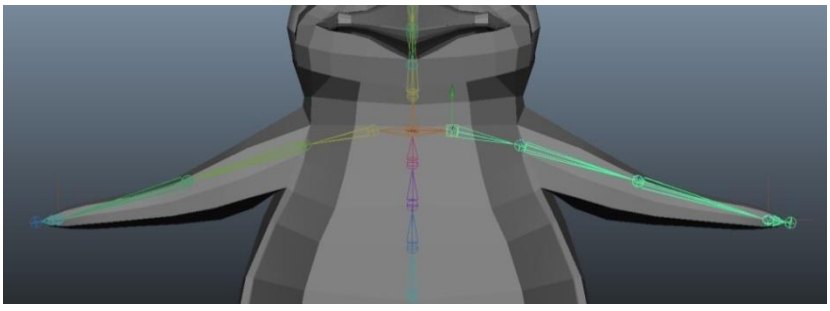
Το κεντρικό τμήμα του σκελετού είναι η σπονδυλική στύλη, η οποία ξεκινάει από τους γοφούς και επεκτείνεται μέχρι το λαιμό, η οποία συνήθως συνεχίζεται μέχρι το κεφάλι. Ο κατάλληλος αριθμός των αρθρώσεων σε ένα σκελετό εξαρτάται

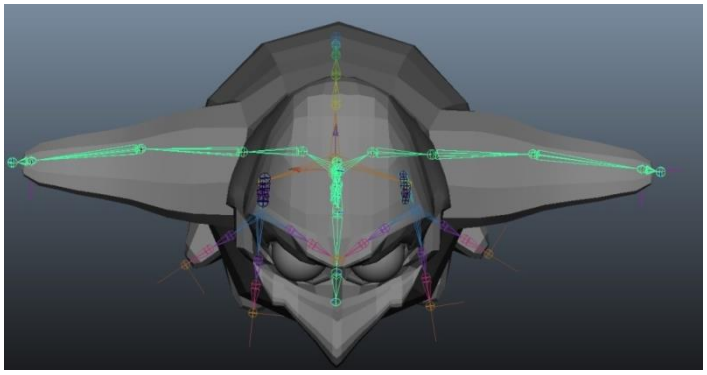
από τα ανατομικά μέρη του χαρακτήρα που θέλουμε να χειριστούμε. Περισσότερες αρθρώσεις σημαίνει καλύτερος έλεγχος σε βάρος της μεγαλύτερης πολυπλοκότητας. Αναλόγως την λεπτομέρεια που θέλουμε να δώσουμε στην κίνηση και το πόσο έλεγχο θέλουμε να έχουμε στο μοντέλο, προσαρμόζουμε τον αριθμό των αρθρώσεων. Η κλίση που δίνουμε στο σκελετό είναι ανάλογη της κίνησης που θέλουμε να προσφέρει, και μοιάζει αρκετά με την κλίση ενός πραγματικού σκελετού αντίστοιχου χαρακτήρα (μοιάζει με το γράμμα S). Αν δεν θέλουμε να ελέγξουμε τις εκφράσεις του προσώπου και απλά θέλουμε να ελέγξουμε την κίνηση του κεφαλιού, συνήθως οι αρθρώσεις σε ένα κεφάλι είναι τέσσερις, μία στο λαιμό που ελέγχει την κλίση του κεφαλιού, μία στο κέντρο του με την οποία ενώνονται η άρθρωση που βρίσκεται στο σαγόνι και η άρθρωση που βρίσκεται στην κορυφή του κεφαλιού.

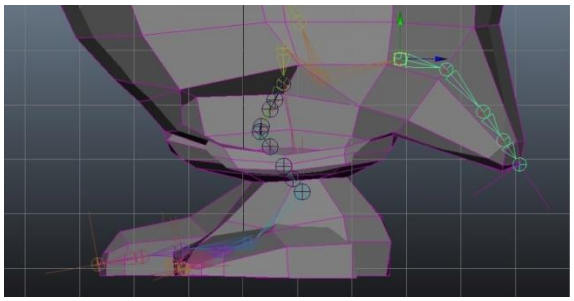
Εκτός από το κεντρικό κομμάτι του σκελετού, πρέπει να διαμορφώσουμε τα άκρα του χαρακτήρα, χέρια, πόδια, ή και ουρά. Η ουρά, αν έχουμε, ανήκει στο κεντρικό κομμάτι του σκελετού και είναι μια προέκταση της σπονδυλικής στύλης στην ίδια ευθεία με άλλη κλίση. Όμως τα χέρια και τα πόδια έρχονται σε όμοια ζευγάρια στο πλάι της σπονδυλικής στύλης. Μπορούμε να φτιάξουμε τα κόκαλα και τις αρθρώσεις στο μισό μέρος του σώματος του χαρακτήρα, πχ στην δεξιά μεριά, και έπειτα χρησιμοποιώντας τη λειτουργία Mirror δημιουργούνται αυτόματα οι αρθρώσεις και τα κόκαλα από την αριστερή μεριά με απόλυτη συμμετρία.

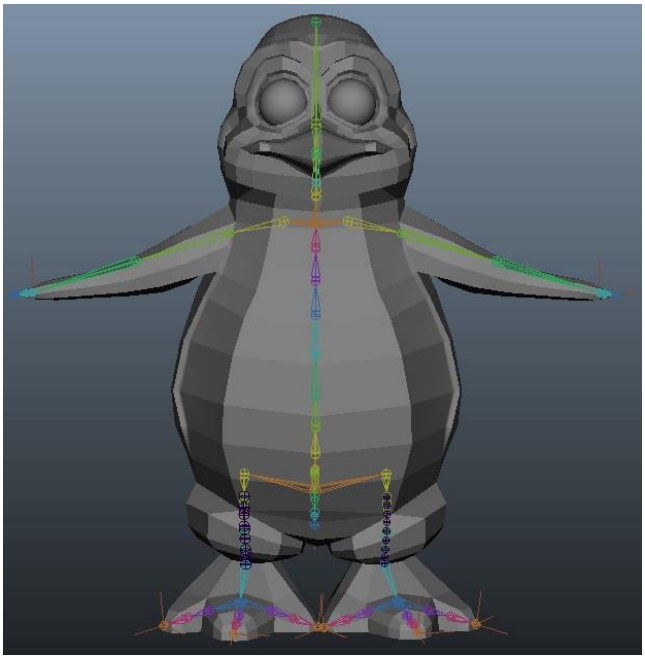












Όταν δημιουργήσουμε το σκελετό, δεσμεύουμε το χαρακτήρα σε αυτόν για να παράγει φυσικές παραμορφώσεις του δέρματος κατά τη διαδικασία του animation. Είναι συνήθως καλύτερο και πιο εύκολο να φτιάχνουμε το animation με ολόκληρο τον σκελετό από πόζα σε πόζα σε επιθυμητά πλαίσια, παρά να χρησιμοποιούμε ξεχωριστά το κάθε διαφορετικό τμήμα του σκελετού. Για το λόγο αυτό, ολοκληρώνοντας τα διάφορα τμήματα του σκελετού στα διάφορα μέρη του σώματος του ήρωα, πρέπει να τα ενώσουμε ώστε να έχουμε έναν ενιαίο σκελετό. Για να δημιουργήσουμε μια ενιαία ιεραρχία, κάνουμε parenting τους σκελετούς αυτούς με την κοντινότερη άρθρωση της σπονδυλικής στήλης.

Όταν συνδέσουμε ένα χαρακτήρα σε ένα σκελετό, ο σκελετός παρέχει μια δομή για το δέρμα του χαρακτήρα και αποτρέπει το δέρμα από την κατάρρευση. Μπορούμε επίσης να προσθέσουμε επιπλέον αρθρώσεις σε περιοχές του χαρακτήρα που θέλουμε να διατηρηθεί ο όγκος της επιφάνειας κατά την παραμόρφωση.

Όπως περιγράψαμε και παραπάνω, υπάρχουν δύο τεχνικές για να φτιάξουμε μια πόζα ενός σκελετού, η προς τα εμπρός κινηματική και η αντίστροφη κινηματική. Μπορούμε να συνδυάσουμε τις δύο τεχνικές κινηματικής IK και FK ή να κάνουμε εναλλαγή μεταξύ τους στις αρθρώσεις που ελέγχονται από μια λαβή IK. Υπάρχουν πολλά είδη των IK λαβών που παρέχουν διαφορετικά χειριστήρια για το χειρισμό μέρη του σκελετού. Ιδιαίτερα αξιοσημείωτη είναι η λαβή IK spline, η οποία το καθιστά εύκολο το στρίψιμο και την κυματιστή κίνηση σε ουρές, λαιμούς, σπονδυλική στύλη, και ούτω καθεξής.

Για να δημιουργήσουμε ένα χαρακτήρα με την εμπρόσθια κινηματική, μπορούμε να περιστρέψουμε κάθε σύνδεσμο ξεχωριστά, μέχρι να πάρει την επιθυμητή θέση. Για παράδειγμα, για να μετακινήσουμε το χέρι σε κάποια θέση, θα πρέπει να περιστραφούν πολλές αρθρώσεις του βραχίονα για να φτάσει στο σημείο που θέλουμε. Φτιάχνοντας animation με την εμπρόσθια κινηματική, το Maya παρεμβάλλει τις περιστροφές των αρθρώσεων ξεκινώντας με την ρίζα-άρθρωση, στη συνέχεια τις αρθρώσεις-παιδιά της ρίζας,

και ούτω καθεξής, μέσω της δράσης της ιεραρχίας σκελετού. Το Maya προχωρεί προς τα εμπρός μέσα από την δράση της ιεραρχίας, ξεκινώντας από την ρίζα.

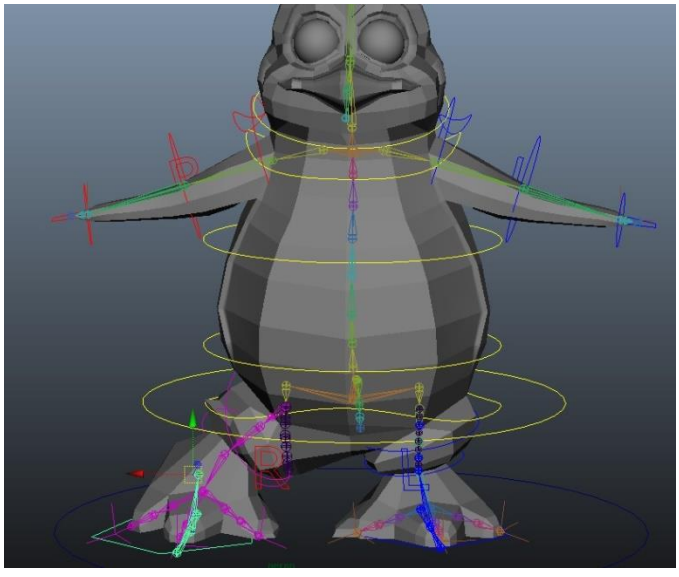
Η εμπρόσθια κινηματική είναι κατάλληλη για τη δημιουργία απλών κινήσεων τόξου, αλλά είναι κουραστική για έναν σύνθετο σκελετό. Επίσης δεν είναι κατάλληλη για την κατευθυνόμενη κίνηση σε έναν συγκεκριμένο στόχο. Για παράδειγμα, για να μετακινήσουμε το χέρι σε κάποια θέση, δεν είναι προφανές το πώς να περιστρέψουμε τις αρθρώσεις σε ένα χέρι.

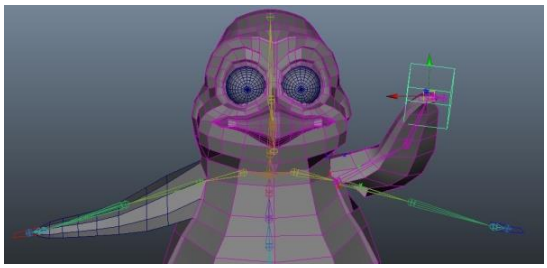
Με την ΙΚ, μπορούμε να δημιουργήσουμε μια επιπλέον δομή ελέγχου, μία λαβή ΙΚ, για συγκεκριμένες αλυσίδες αρθρώσεων, όπως τα χέρια και τα πόδια. Μια λαβή ΙΚ επιτρέπει να δημιουργούμε και να εμψυχώσουμε μια ολόκληρη αλυσίδα από αρθρώσεις με τη μετακίνηση ενός ενιαίου χειρισμού. Καθώς κινούμε τη ΙΚ λαβή, περιστρέφει αυτόματα όλες τις αρθρώσεις στην αλυσίδα αρθρώσεων.

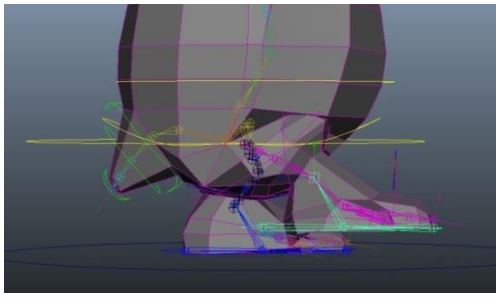
Η ΙΚ είναι καλύτερη από ό, τι η εμπρόσθια κινηματική για ένα στόχο κατευθυνόμενης κίνησης, επειδή μπορούμε να επικεντρωθούμε στο στόχο και όχι στο πώς θα πρέπει να περιστρέψουμε κάθε άρθρωση για την επίτευξη αυτού του στόχου.

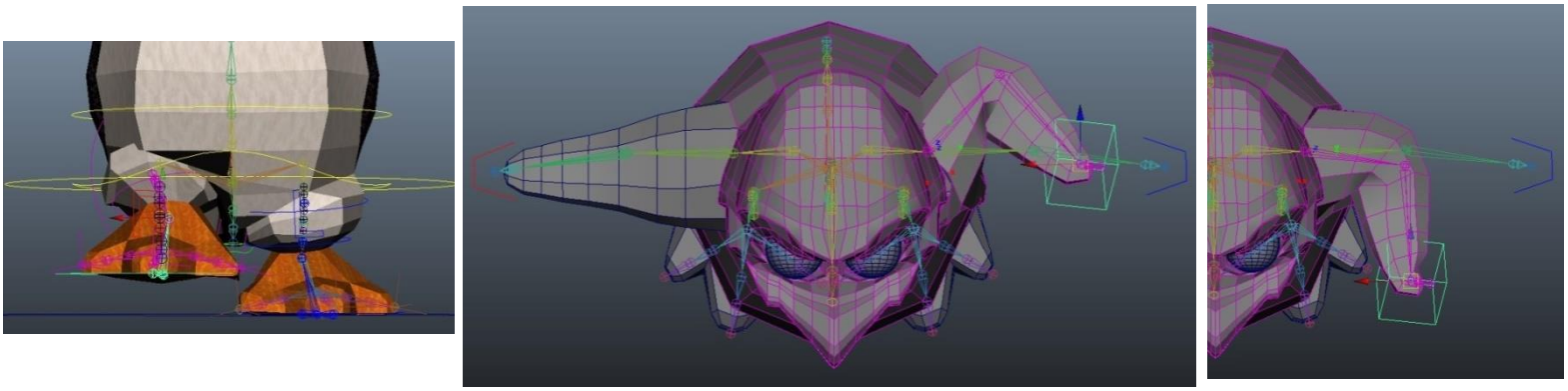
Κατά τη δημιουργία πόζας και animations με την ΙΚ, θέτουμε τις τρέχουσες γωνίες αρθρώσεων σε όλο το σκελετό ως τις προτιμώμενες γωνίες, μετά την ολοκλήρωση του σκελετού. Το Maya στη συνέχεια χρησιμοποιεί την τρέχουσα κάμψη στα γόνατα και τους αγκώνες ως προτιμώμενη αρχική κατεύθυνση περιστροφής των αρθρώσεων αυτών κατά τη διάρκεια της αντίστροφης κινηματικής (ΙΚ). Αυτό καθιστά ευκολότερο να θέσουμε το χαρακτήρα σε κίνηση με τρόπο φυσικό για έναν ανθρώπινο χαρακτήρα.

Στη συνέχεια δημιουργούμε μια λαβή ΙΚ που μας επιτρέπει να ελέγχουμε όλες τις αρθρώσεις, η οποία συνδέεται συνήθως στην τελευταία άρθρωση της αλυσίδας που ελέγχουμε. Ορίζουμε άλλο ένα κλειδί για τη λαβή αυτή και φτιάχνουμε το animation.

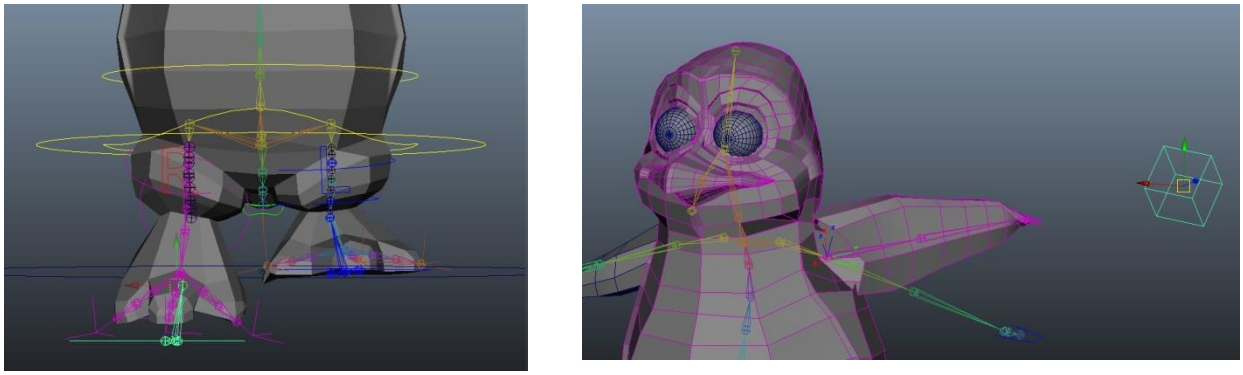








Δεν έχει σημασία πόσο μακριά τραβάμε τη λαβή χειρισμού, οι αρθρώσεις του τμήματος του σώματος που ελέγχουμε (πχ του ποδιού) δεν θα επεκταθούν πέρα από την ευθεία θέση του. Αυτό είναι σημαντικό καθώς δεν θέλουμε το μέγεθος του σκελετού να αλλάζει όταν του αλλάζουμε τη θέση. Ωστόσο, αν σύρουμε μια άρθρωση, το οστό ιεραρχικά πάνω από αυτή την άρθρωση θα επιμηκυνθεί, για αυτό πρέπει να επιλέγουμε την άρθρωση με τη IK λαβή πριν τη μετακινήσουμε.



Καθώς τραβάμε τη λαβή σε κάποιες θέσεις, οι αρθρώσεις γυρίζουν απότομα για αυτό είναι δύσκολο να ελέγξουμε την τοποθέτηση σε μια περιοχή. Υπάρχουν όμως αρκετά εργαλεία για την επίλυση αυτού του προβλήματος, όπως το IK Rotate Plane handle, το Twist manipulator, το Pole Vector XYZ και το Pole Vector Constraint. Εάν μετακινήσουμε τις αρθρώσεις, τα οστά μακραίνουν και το δεσμευμένο δέρμα παραμορφώνεται.

Smooth Skinning

Αφού δημιουργήσουμε το σκελετό, τον δεσμεύουμε με την επιφάνεια του χαρακτήρα, έτσι ώστε οι επιφάνειες να κινούνται με το σκελετό κατά τη διάρκεια της κίνησης. Η δέσμευση ονομάζεται skinning, και η επιφάνεια ενός χαρακτήρα μετά τη δέσμευση ονομάζεται δέρμα. Μπορούμε να δεσμεύσουμε έναν σκελετό σε πολλαπλές επιφάνειες ή ακόμη και σε μια επιλογή από κορυφές του πολυγώνου ή NURBS CVs ή επιφάνειες υποδιαίρεσης.

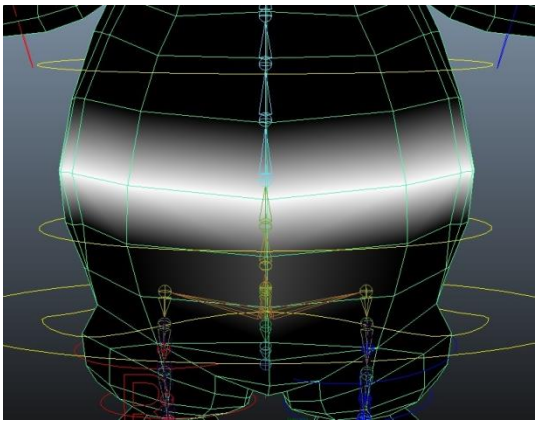
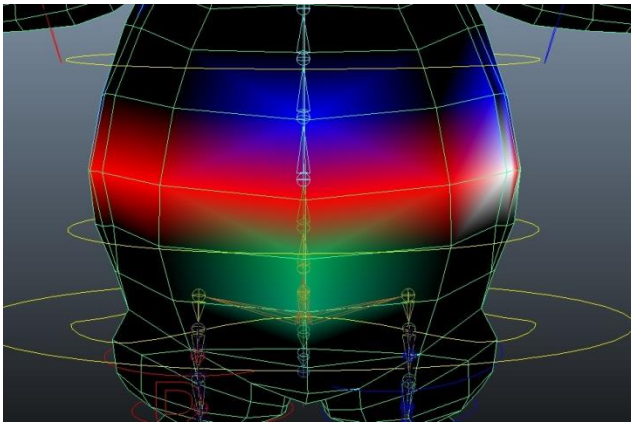
Το δέρμα του χαρακτήρα πρέπει να παραμορφώνεται φυσικά καθώς κινείται ο σκελετός. Κοντά σε αρθρώσεις υπάρχουν εξογκώματα του δέρματος ή ζάρες όταν

περιστρέφουμε τις αρθρώσεις. Στο Maya, το δέρμα παραμορφώνεται επειδή οι κορυφές της επιφάνειας (ή CV) κινούνται σε απόκριση της περιστροφής των γειτονικών αρθρώσεων. Οι κορυφές είναι γνωστές ως σημεία του δέρματος, τα οποία είναι χρήσιμα για την κίνηση στους αγκώνες, τους ώμους, τους λαιμούς, και ούτω καθεξής.

Από προεπιλογή, η επιρροή μιας άρθρωσης στην κίνηση ενός σημείου του δέρματος εξαρτάται από το πόσο κοντά είναι στην άρθρωση αυτή. Μπορούμε να επεξεργαστούμε το σημείο του δέρματος στάθμισης και να αλλάξουμε την προεπιλεγμένη κίνηση ακολουθώντας συγκεκριμένες διαδικασίες δένοντας ομαλά το σκελετό και τροποποιώντας τα βάρη του δέρματος χρησιμοποιώντας το εργαλείο Paint Skin Weights Tool αλλάζοντας έτσι την παραμόρφωση του δέρματος που επηρεάζεται από τα αντικείμενα.

Το πολύ τρεις αρθρώσεις μπορούν να επηρεάζουν ένα σημείο του δέρματος. Από προεπιλογή, η πλησιέστερη άρθρωση προς το σημείο έχει τη μεγαλύτερη επιρροή. Η δεύτερη πιο κοινή επιρροή είναι η άρθρωση γονέας της εν λόγω άρθρωσης ή το παιδί της, ανάλογα με το ποια είναι πιο κοντά στο σημείο. Η τρίτη μεγαλύτερη επιρροή είναι η πλησιέστερη άρθρωση γονέας ή παιδί της δεύτερης άρθρωσης. Η επιρροή μειώνεται με την απόσταση από τις αρθρώσεις. Το ποσό της επιρροής που έχει η κάθε άρθρωση σε οποιοδήποτε σημείο του δέρματος είναι το βάρος του δέρματος. Τα προεπιλεγμένα βάρη του δέρματος δημιουργούν ομαλές παραμορφώσεις του δέρματος σε αγκώνες, γόνατα, και σε άλλα μέρη που οι κοντινές αρθρώσεις περιστρέφονται. Προσαρμόζουμε τα βάρη του δέρματος για να αποτρέψουμε την κατάρρευση μιας περιοχής κατά την παραμόρφωση

Για να θέσουμε σωστά τα βάρη πρέπει να δοκιμάσουμε διαφορετικές πόζες στο σκελετό και βλέποντας την παραμόρφωση που σχηματίζεται, τα προσαρμόσουμε αναλόγως. Το Maya δείχνει τα βάρη είτε σε μια κλίμακα του γκρι, είτε σε μια ποικιλία χρωμάτων η οποία παρέχει περισσότερες λεπτομέρειες για τα μικρά βάρη. Το κάθε χρώμα δείχνει την επιρροή που έχει η επιλεγμένη άρθρωση στην παραμόρφωση του δέρματος. Σε γενικές γραμμές, μια λευκή περιοχή του δέρματος επηρεάζεται σχεδόν εξ ολοκλήρου από την επιλεγμένη ένωση. Μια γκρίζα περιοχή επηρεάζεται σημαντικά από μία ή δύο επιπλέον αρθρώσεις, και μια μαύρη περιοχή δέρματος δεν επηρεάζεται από την επιλεγμένη άρθρωση. Αντίστοιχα με τα χρώματα, η κόκκινη επιφάνεια επηρεάζεται εξ ολοκλήρου από την επιλεγμένη άρθρωση, οι κιτρινες, πορτοκαλί, πράσινες και μπλε επηρεάζονται από πολλαπλές αρθρώσεις, ενώ οι μαύρες καθόλου.

Μπορούμε να τροποποιήσουμε τα βάρη του δέρματος ώστε να επηρεάζονται από οποιαδήποτε άρθρωση αλλάζοντας την περιοχή των χρωμάτων. Έμφαση πρέπει να δίνεται στην κύρια άρθρωση που έχει την μεγαλύτερη επιρροή. Επιθυμητά αποτελέσματα έχουμε μετά από πολλούς πειραματισμούς και δοκιμές με διαφορετικές πόζες καλύβοντας αρκετές περιπτώσεις κίνησης, ώστε να έχουμε μια φυσική κίνηση.

Τα βάρη του δέρματος και τα αντικειμένα επιρροής δεν μπορούν να ξεπεράσουν όλα τα προβλήματα μοντελοποίησης, όπως ανεπιθύμητες τσαλακώσεις δέρματος ή βαθουλώματα. Αυτό συμβαίνει λόγω της ασύμμετρης διάταξης των πολυγωνικών ακμών στην περιοχή του ισχίου του αρχικού μοντέλου. Για να διορθώσουμε το πρόβλημα, θα πρέπει να αποσυνδέσουμε το δέρμα, να αλλάξουμε το αρχικό μοντέλο, και να δεσμεύσουμε πάλι ομαλά το μοντέλο .

Cluster and blend shape deformers

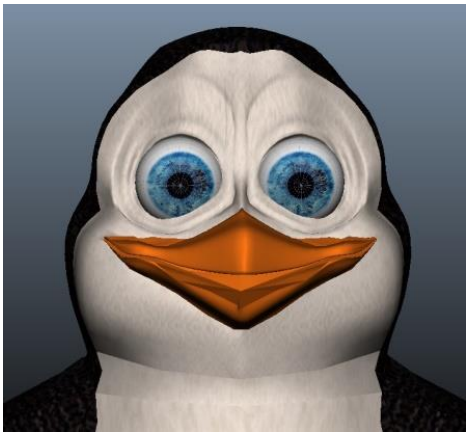
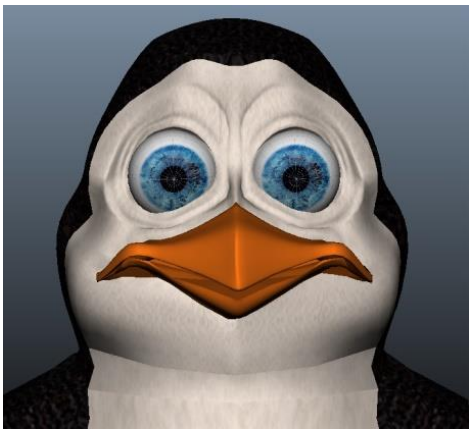
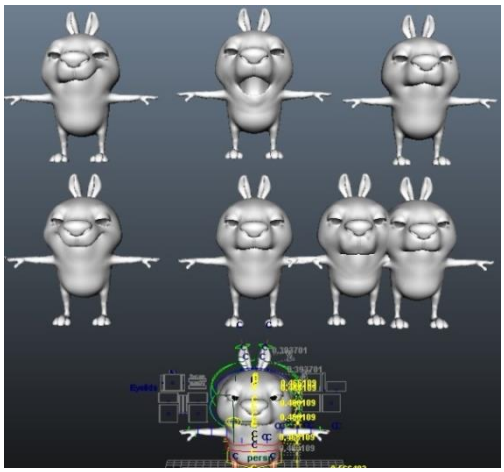
Το Facial animation (προσώπου) είναι αναπόσπαστο στοιχείο του χαρακτήρα κινουμένων σχεδίων. Το πρόσωπο ενός χαρακτήρα μπορεί να μεταδώσει μια σειρά από συναισθήματα και εκφράσεις. Το Maya έχει δύο εργαλεία παραμόρφωσης που διευκολύνουν την εγκατάσταση του χαρακτήρα για τα κινούμενα σχέδια του προσώπου, τους deformers διασποράς (cluster) και μίξης σχήματος (blend) .

Οι Cluster Deformers μας επιτρέπουν να ελέγχουμε ένα σύνολο σημείων, όπως CVs, κορυφές, ή σημεία πλέγματος, που ανήκουν σε ένα αντικείμενο με ποικίλες ποσότητες επιρροών για να δημιουργήσουν ένα σχήμα στόχου για μια κινούμενη εικόνα. Με αυτούς μπορούμε να διαμορφώσουμε το στόμα ενός χαρακτήρα και να θέσουμε ένα χαμόγελο.

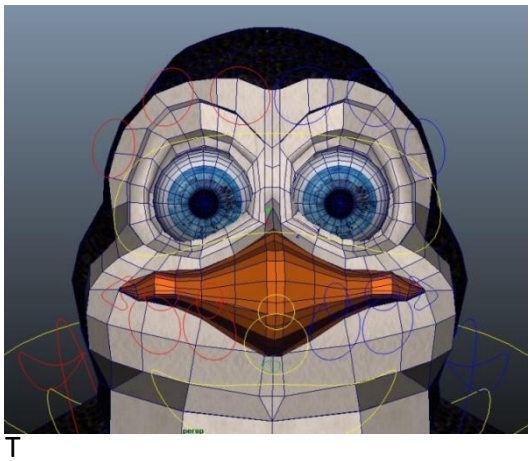
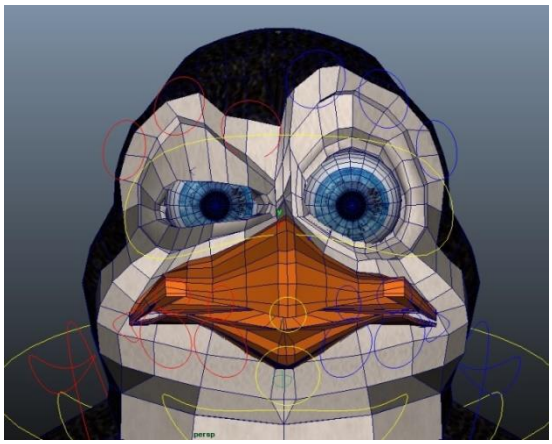
Ένας Blend shape deformer παρέχει μια διεπαφή για την ανάμειξη διαφόρων σχημάτων στόχων, και ελέγχει το εύρος της κίνησης για το σύμπλεγμα ενός αντικειμένου ή προσώπου. Με αυτόν μπορούμε να φτιάξουμε το animation ενός προσώπου από μια ουδέτερη στάση σε ένα χαμόγελο. Για αυτό να το καταφέρουμε αυτό, επιλέγουμε τα CVs σε μια περιοχή και δημιουργούμε έναν deformer συμπλέγματος. Χρησιμοποιούμε τη λαβή του συμπλέγματος deformer για να ρυθμίσουμε τη θέση του deformer συμπλέγματος, και το εργαλείο Cluster

Weights Tool για να βελτιώσουμε τις τιμές βάρους του συμπλέγματος. Τέλος, δημιουργούμε ένα deformer Blend σχήμα για να ελέγχει την ανάμιξη μεταξύ των σχημάτων στόχου.

Για να επεξεργαστούμε το πρόσωπο με το μείγμα σχήματος διπλασιάζουμε αρχικά το κεφάλι αν ο χαρακτήρας αποτελείται μόνο από το κεφάλι, διαφορετικά διπλασιάζουμε ολόκληρο το σώμα. Δημιουργούμε έναν deformer συμπλέγματος στο δεύτερο πρόσωπο (αντικείμενο-στόχο) για να διαμορφώσουμε την έκφραση σε ένα χαμόγελο. Ο deformer συμπλέγματος θα χρησιμοποιηθεί για να αλλάξουμε το αντικείμενο βάσης σε ένα χαμογελαστό πρόσωπο.

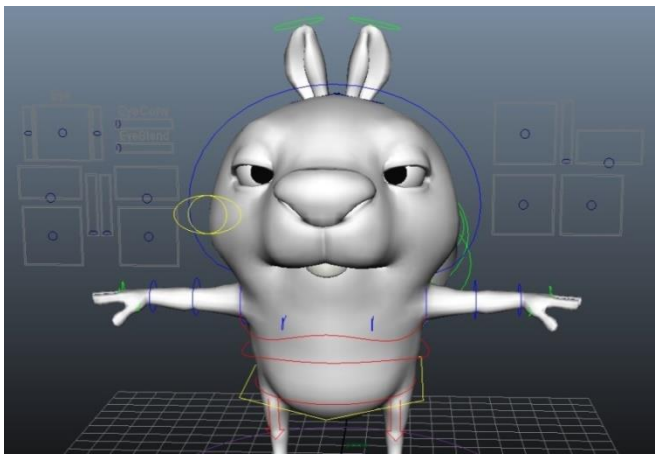
Ο deformer συμπλέγματος δημιουργεί ένα σύνολο του οποίου τα μέλη αποτελούνται από επιλεγμένα σημεία (CVs, κορυφές, ή σημεία πλέγματος). Μπορούμε να εκχωρήσουμε ένα ποσοστό του βάρους του κάθε σημείου, που δείχνει πόσο πολύ θέλουμε κάθε σημείο να επηρεάζεται από οποιαδήποτε μετάφραση, περιστροφή, ή κλίμακα του συνόλου του συμπλέγματος. Όταν μετατρέψουμε το σύμπλεγμα, τα σημεία μετασχηματίζονται σύμφωνα με τα ποσοστά που έχουμε ορίσει. Ένα σύμπλεγμα είναι χρήσιμο για το τέντωμα, τη μετακίνηση ή τη συμπίεση του συνόλου ή μέρους της επιφάνειας.

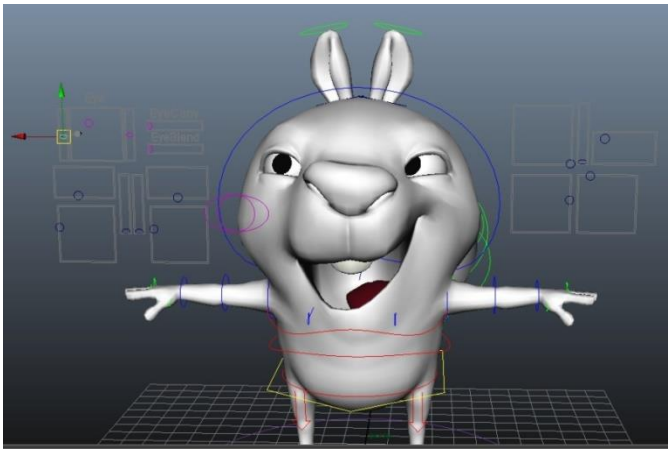



T

Τα σημεία σε ένα σύμπλεγμα κινούνται όταν μετακινούμε τη λαβή του συμπλέγματος. Η απόσταση που κινείται ένα συγκεκριμένο σημείο εξαρτάται από το βάρος του. Τα χαμηλότερα βάρη προκαλούν λιγότερη κίνηση, ενώ τα υψηλότερα βάρη προκαλούν υπερβολική κίνηση.

Μπορούμε να δημιουργήσουμε ένα μείγμα σχήματος deformer για να ελέγχει το χαμόγελο στο αντικείμενο βάσης χρησιμοποιώντας το αντικείμενο στόχου. Ένα μείγμα σχήματος deformer είναι ιδανικό για τα κινούμενα σχέδια του προσώπου, όπου θα πρέπει να έχουμε μια σειρά από θέσεις του προσώπου άμεσα διαθέσιμες για χρήση σε σειρά κινουμένων σχεδίων. Με ένα μείγμα σχήματος deformer, μπορούμε να ρυθμίσουμε το πρόσωπο ενός χαρακτήρα να συνδυάζει χαμόγελο, συνοφρύωμα, προσποιητό χαμόγελο, και ούτω καθεξής. Για κάθε διαφορετικό μορφασμό που θέλουμε να αποδώσουμε στο χαρακτήρα, δημιουργούμε ένα νέο πρόσωπο αντιγράφοντας το αρχικό πρόσωπο, και εφαρμόζουμε τις αλλαγές στο νέο. Όταν ολοκληρώσουμε όλες τις εκφράσεις που θέλουμε στα νεά πρόσωπα, τα ενώνουμε όλα στο αρχικό, έτσι ώστε να υπάρχει μόνο ένα πρόσωπο με διαφορετικές εκφράσεις.





Για να διαχειριστούμε τις εκφράσεις του προσώπου χρειάζονται και οι δύο deformers, cluster και blend shape. Για καλύτερο αποτέλεσμα και καλύτερο έλεγχο των εκφράσεων του προσώπου, δημιουργούμε πολλά clusters με deformer μείγμα σχήματος, πχ για να προσομοιώσουμε τα φωνήματα του λόγου. Το σχήμα στόχου δεν πρέπει να είναι πολύ διαφορετικό από το σχήμα βάσης.

3.7.6 Polygon Texturing

Οι χάρτες υφής μας επιτρέπουν να τροποποιήσουμε την εμφάνιση των 3D μοντέλων και των σκηνών μας στο Maya. Οι χάρτες υφής είναι εικόνες που εφαρμόζουμε και τοποθετούμε με ακρίβεια πάνω σε επιφάνειες χρησιμοποιώντας μια διαδικασία που ονομάζεται χαρτογράφηση υφής (texture mapping). Όταν μια εικόνα χαρτογραφηθεί ως υφή

πάνω σε μία επιφάνεια, μεταβάλλει την εμφάνιση της επιφάνειας με έναν μοναδικό τρόπο. Οι χάρτες υφής μας επιτρέπουν να δημιουργήσουμε πολλά ενδιαφέροντα οπτικά εφέ, να εφαρμόσουμε ετικέτες και λογότυπα σε επιφάνειες, να εφαρμόσουμε ανάγλυφες λεπτομέρειες και τα χαρακτηριστικά σε μια επιφάνεια, αντί να χρειάζεται να διαμορφώσουμε τις λεπτομέρειες στην επιφάνεια άμεσα, και ακόμη να δημιουργήσουμε σκηνικά. Τα περισσότερα χαρακτηριστικά σκίασης σε μία επιφάνεια υλικού μπορούν να μεταβάλλονται από ένα χάρτη υφής όπως το χρώμα, τα κάτοπτρα, η διαφάνεια, και η ανακλαστικότητα. Η χαρτογράφησης υφής, είναι ένα βασικό συστατικό στην 3D παραγωγή.

UV texture mapping

Υπάρχουν διάφορες τεχνικές για τη χαρτογράφηση υφής σε 3D επιφάνειες ανάλογα με τον τύπο της επιφάνειας (NURBS, πολύγωνα, επιφάνειες υποδιαίρεσης). Ορισμένες τεχνικές περιλαμβάνουν την προετοιμασία των επιφανειών για τη χαρτογράφηση υφής, όπως μια χαρτογράφηση υφής σε μια πολυγωνική και σε μια υποδιαίρεσης επιφάνεια οι υφές εφαρμόζονται χρησιμοποιώντας UV συντεταγμένες υφής .

Οι UV συντεταγμένες υφής, ή UVs, είναι δύο διαστάσεων συντεταγμένες που βρίσκονται οι πληροφορίες των συστατικών κορυφής για μια 3D επιφάνεια. Τα UVs ελέγχουν την τοποθέτηση ενός χάρτη υφής σε ένα 3D μοντέλο συσχετίζοντας τη θέση pixel του 2D χάρτη υφής στις θέσεις κορυφών του μοντέλου, έτσι ώστε η υφή να τοποθετηθεί (χαρτογραφηθεί) σωστά. Οι συντεταγμένες UV είναι απαραίτητες για την εφαρμογή και την ακριβή τοποθέτηση χαρτών υφής για πολυγωνικές επιφάνειες. Ένα UV μπορεί να δημιουργηθεί για ένα πλέγμα μιας επιφάνειας χρησιμοποιώντας μια ποικιλία από τεχνικές χαρτογράφησης UV (Planar χαρτογράφηση, σφαιρική χαρτογράφηση, κυλινδρική χαρτογράφηση, και αυτόματη χαρτογράφηση).

Τα UVs μπορούν να δημιουργηθούν και να τροποποιηθούν ώστε να ανταποκρίνονται στις απαιτήσεις του χάρτη υφής ακολουθώντας συγκεκριμένες διαδικασίες:

- Αντιστοιχίζουμε το 2D χάρτη υφής στο πολυγωνικό μοντέλο.
- Χαρτογραφούμε έναν χάρτη υφής συντεταγμένων (UVs) στην πολυγωνική επιφάνεια.
- Συσχετίζουμε τις UVs μεταξύ της σκηνής και του συντάκτη UV υφής (UV Texture Editor).
- Χρησιμοποιούμε τον UV Texture Editor για να απεικονίσουμε το πώς συνδέεται η υφή συντεταγμένων από ένα τρισδιάστατο μοντέλο με ένα ειδικό δισδιάστατο χάρτη υφής.
- Προσδιορίζουμε τις βασικές απαιτήσεις διάταξης UV.
- Ορίζουμε τις ρυθμίσεις ώστε να απεικονίσουμε τα σύνορα υφής στον UV Texture Editor και στην σκηνή για να κατανοήσουμε καλύτερα τον τρόπο που ο χάρτης υφής τοποθετείται στο πολυγωνικό μοντέλο.

- Επιλέγουμε και τοποθετούμε τις UV συντεταγμένες υφής εντός του UV Επεξεργαστή Υφής χρησιμοποιώντας τα εργαλεία μετατροπής για να κάνουμε τα UVs να ταιριάζουν με ένα προκαθορισμένο χάρτη υφής.
- Ράβουμε τις UV συντεταγμένες υφής στα υπάρχοντα κελιά UV .

Οι επιφάνειες εμφανίζονται σε μια εικόνα με βάση τη σκίαση των υλικών που εφαρμόζουμε σε αυτά. Το προεπιλεγμένο υλικό σκίασης που ανατίθεται σε επιφάνειες παρέχει βασικά χαρακτηριστικά, όπως το χρώμα και τη διαφάνεια. Ανάλογα με τις τελικές απαιτήσεις της εικόνας, μπορούμε να βελτιώσουμε την οπτική εμφάνιση ενός αντικειμένου με την προσθήκη ενός ή περισσοτέρων χαρτών υφής στο υλικό σκίασης. Ένας βασικός χάρτης υφής που μπορούμε να εφαρμόσουμε είναι μια εικόνα bitmap που αναφέρεται επίσης ως αρχείο υφής.

Κατά τη δημιουργία ενός μοντέλου το υλικό που χρησιμοποιείται από προεπιλογή είναι το Lambert, το οποίο δίνει ένα έντονο γκρι χρώμα στο μοντέλο. Οι τροποποιησείς δεν γίνονται στο αρχικό υλικό σκίασης, αλλά δημιουργούμε ένα νέο υλικό σκίασης για το οποίο επεξεργαζόμαστε. Ο Επεξεργαστής Χαρακτηριστικών (Attribute Editor) εμφανίζει τα διαθέσιμα υλικά και τα διάφορα χαρακτηριστικά τους, στον οποίο μπορούμε να τα τροποποιήσουμε και να προσθέσουμε τις εικόνες υφής.

Μπορούμε να εξάγουμε μια εικόνα bitmap 2D του UV Texture Editor για την να χρησιμοποιήσουμε ως οδηγό για το σχεδιασμό της υφής στο λογισμικό δημιουργίας της εικόνας μας. Εάν χρησιμοποιούμε το Adobe ® Photoshop ® για τη δημιουργία της εικόνας υφής και την επεξεργασία , μπορούμε να δημιουργήσουμε ένα στιγμιότυπο UV σε μορφή αρχείου .PSD, που μας επιτρέπει να κρατήσουμε τις διάφορες συνιστώσες του χάρτη υφής σε πολλαπλά στρώματα. Αυτό είναι χρήσιμο καθώς μπορούμε να επεξεργαστούμε ένα στοιχείο της υφής κατά τη διάρκεια της παραγωγής. Ορίζουμε την πηγή του αρχείου του χάρτη υφής στο υλικό που θέλουμε για να αποδωθεί η εικόνα.

Όταν μια υφή εφαρμόζεται σε ένα υλικό σκίασης, τα χαρακτηριστικά της προστίθονται στις υπάρχουσες ιδιότητές του, δηλαδή όλα τα υπάρχοντα χαρακτηριστικά σκίασης παραμένουν αμετάβλητα, εκτός από το χαρακτηριστικό που τροποποιείται με την υφή. Έτσι το γκρι χρώμα που αρχικά έχει το μοντέλο αντικαταστάται από τα χρώματα και σχέδια της υφής.

Ο χάρτης υφής δεν εμφανίζεται σωστά στο μοντέλο, επειδή οι προεπιλεγμένες UVs για το primitive αντικείμενο δεν συσχετίζονται με την υφή του χάρτη σωστά. Ο χάρτης υφής σχεδιάστηκε μόνο για συγκεκριμένες πλευρές του αντικειμένου, όμως εμείς έχουμε τροποποιήσει το αρχικό σχήμα του αντικειμένου για να φτιάξουμε το χαρακτήρα και επίσης θέλουμε να προσαρμοστεί η υφή σε όλες τις πλευρές του. Για να διορθώσουμε αυτό το πρόβλημα, πρέπει να τροποποιήσουμε τις προεπιλεγμένες UV συντεταγμένες υφής του

μοντέλου έτσι ώστε να ταιριάζει με τη διάταξη της εικόνας υφής, το οποίο επιτυχάνουμε με τη χρήση του UV Texture Editor.

Ο UV Texture Editor μας επιτρέπει να δούμε και να επεξεργαστούμε αλληλεπιδραστικά τις UV συντεταγμένες υφής για τα πολύγωνα της επιφανειας του μοντέλου. Οι UV συντεταγμένες υφής εμφανίζονται σε μορφή προβολής 2D στον UV συντάκτη υφής, όπως μια επίπεδη δισδιάστατη απεικόνιση. Τα UVs επισημαίνονται με τις γραμμές που συνδέουν τα UVs υποδείκνύοντας την περιοχή της υφής των Uvs που εκπροσωπούν. Πρέπει να εξασφαλίσουμε ότι το σχήμα και η τοποθέτηση των UVs ταιριάζει με την εικόνα, όπως εμφανίζεται στην προβολή 2D του UV Editor Υφής, ώστε να εμφανίζεται ο σωστά ο χάρτης στο μοντέλο στην προβολή 3D.

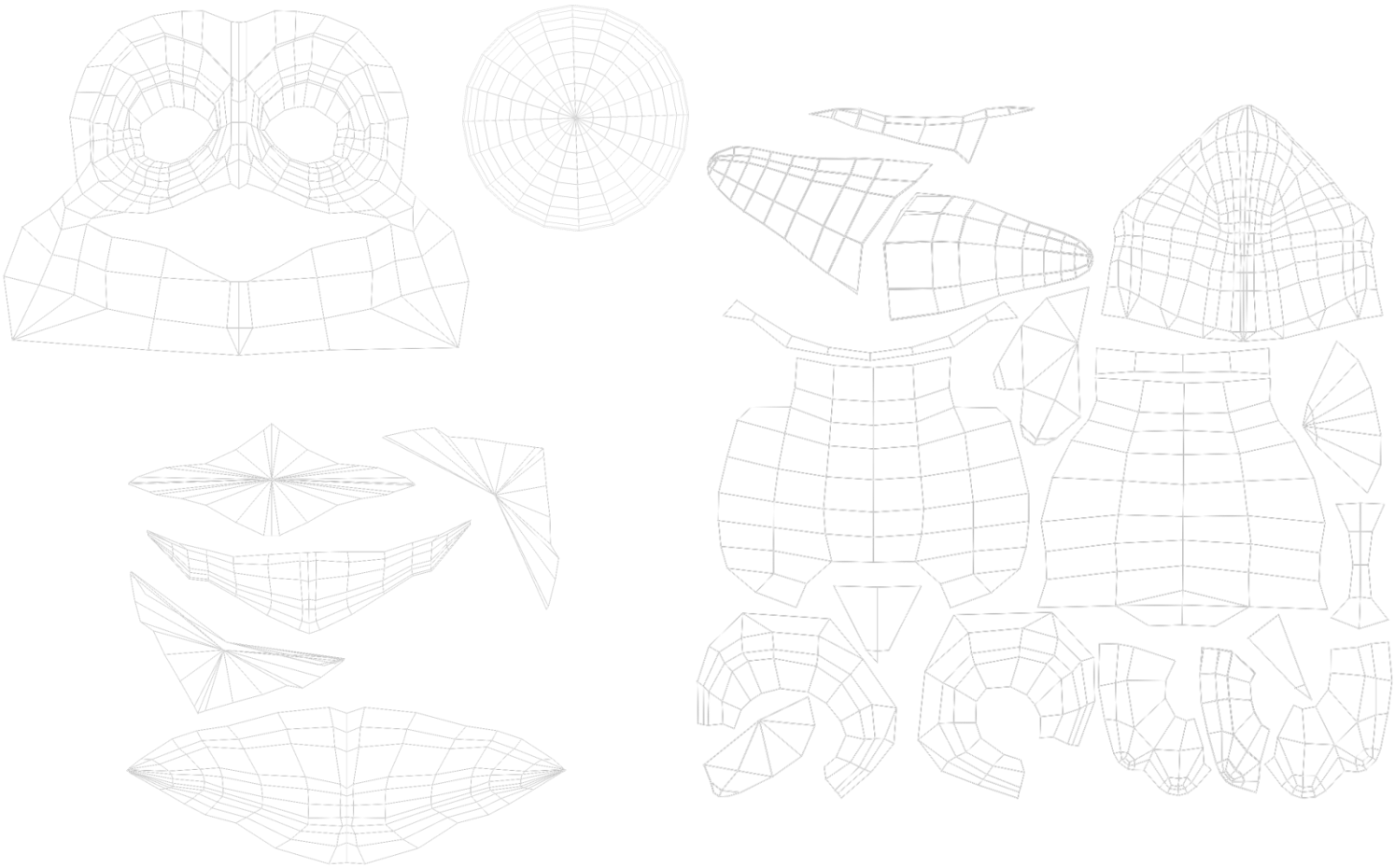
Η εικόνα που έχουμε ορίσει ως το χάρτη υφής για το μοντέλο επίσης εμφανίζεται σε 2D μορφή στον UV Texture editor, στο πάνω δεξιά τεταρτημόριο της 2D γραφικής παράστασης που ονομάζεται το εύρος της εικόνας UV ή UV Texture Space. Οι συντεταγμένες για αυτό το εύρος τεταρτημόριων είναι από 0,0 έως 1,1 και αντιπροσωπεύουν το χώρο υφής για την επιφάνεια. Ο τρόπος που τα UVs εμφανίζονται σε αυτό το τεταρτημόριο σε σχέση με την εικόνα που εμφανίζεται, έχει άμεση σχέση με το πώς η υφή χαρτογραφείται στην επιφάνεια.

Για τη σωστή χαρτοργάφηση υφής, κόβουμε τις πλευρές των επιφανειών του μοντέλου, με τρόπο που να μας βολεύει, καλύπτοντας όσο το δυνατόν μεγαλύτερες και όμοιες επιφάνειες, και στη συνέχεια τις ξεδιπλώνουμε, τις απλώνουμε, τις κλιμακώνουμε και τις τοποθετούμε κατάλληλα στον UV Texture editor ώστε να χωρέσουν στο πρώτο τεταρτημόριο εντός της περιοχής 0 έως 1.

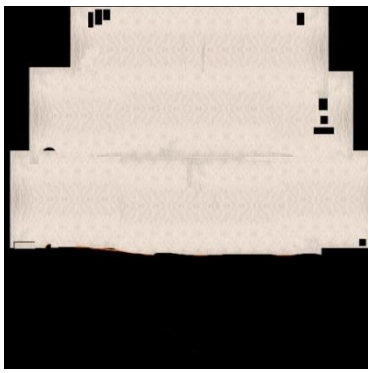
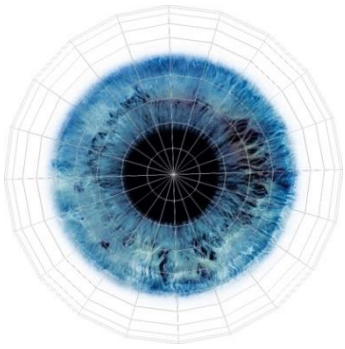
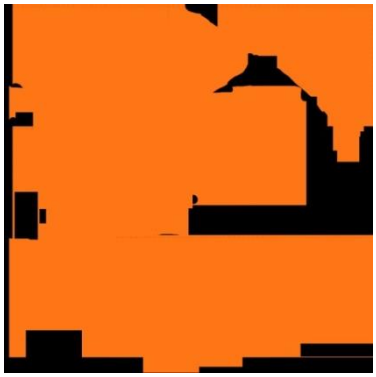
Όταν τοποθετήσουμε τα UVs του επιλεγμένου υλικού του μοντέλου στο πρώτο τεταρτημόριο, χωρίς να επικαλύπτει το ένα το άλλο, φτιάχνουμε την υφή στο Photoshop αναλόγως τη θέση τους. Ακριβώς πάνω από τη θέση του κάθε UV βάζουμε την εικόνα και το χρώμα που θέλουμε να έχει η επιφάνεια που αντιπροσωπεύει, και έπειτα αφαιρούμε τα UV κρατώνας μόνο την εικόνα που δημιουργήσαμε. Στη συνέχεια προσαρτούμε το αρχείο του Photoshop στο υλικό που θέλουμε, και εφαρμόζεται πάνω σε αυτό η εικόνα που κατασκευάσαμε, η οποία εμφανίζεται επίσης στον UV Texture editor μαζί με τα UVs. Πρέπει όμως να προσέχουμε να εφαρμόζεται τέλεια η εικόνα πάνω στα UVs για να έχουμε καλά αποτελέσματα.

Για περισσότερες λεπτομέρειες, μπορούμε να χωρίσουμε το μοντέλο σε πολλά διαφορετικά υλικά αναλόγως τα μέρη του σώματος που θέλουμε να αποδώσουμε την εικόνα, ανεξάρτητα αν αποτελούνται από το ίδιο υλικό. Ο λόγος που το κάνουμε αυτό είναι ότι πρέπει όλα τα UVs να χωρέσουν στο πρώτο τεταρτημότιο του επεξεργαστή των υφών. Αν το μοντέλο είναι πολύπλοκο δεν μπορούμε να τα χωρέσουμε όλα και να δώσουμε λεπτομέρειες. Έτσι το

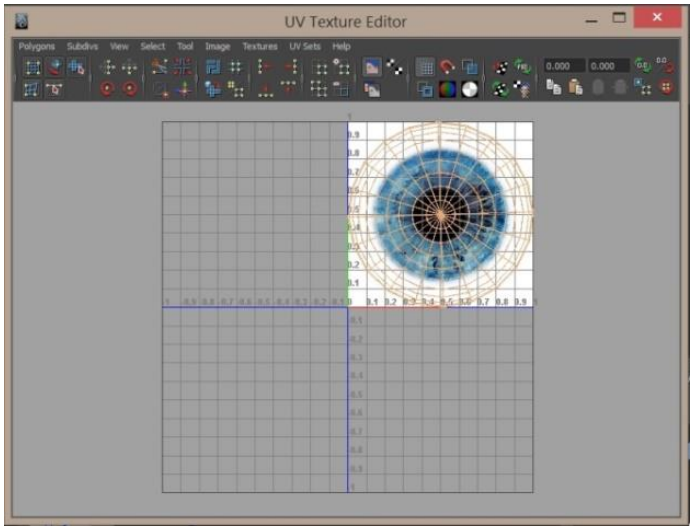
χωρίζουμε σε τμήματα υλικών, και έχουμε διαφορετικά UVs και αρχεία υφών για το καθένα από αυτά. Στις παρακάτω εικόνες φαίνονται τα UVs του κεφαλιού του πιγκουίνου, των ματιών, του ράμφους, και του σώματός του.

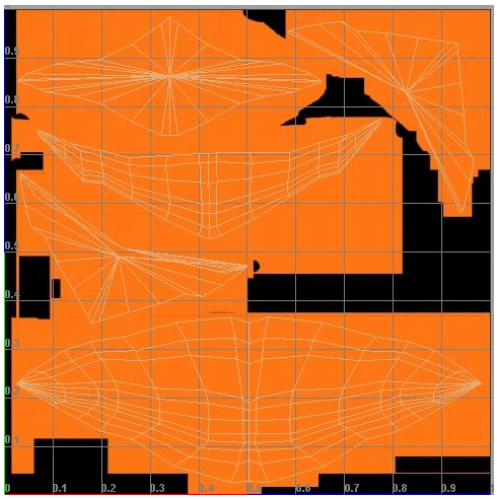


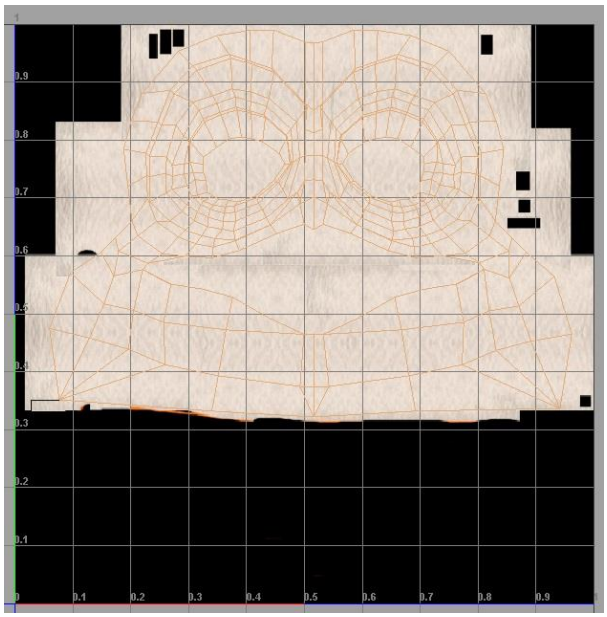
Στις επόμενες εικόνες φαίνονται τα αρχεία υφής που δημιουργήσαμε για τα αντίστοιχα UVs στο Photoshop:

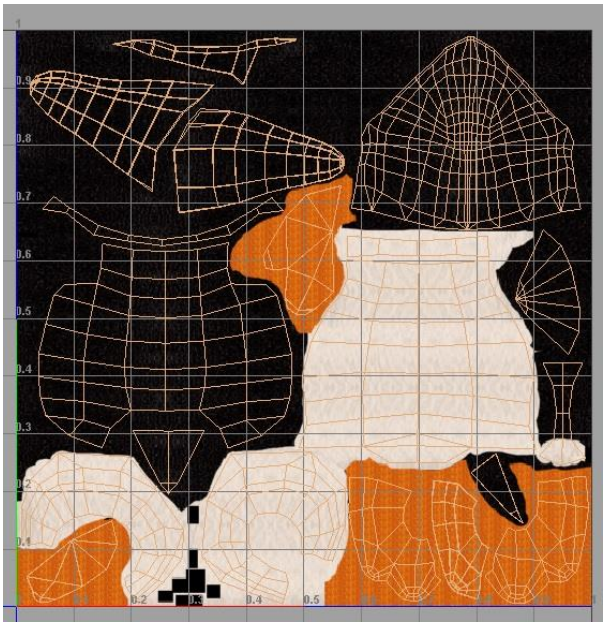





Και στις επόμενες δείχνουμε πως φαίνονται στον UV Texture Editor όταν προσαρμόσουμε αυτές τις υφές στα αντίστοιχα υλικά:









Συνήθως τα UVs μιας επιφάνειας είναι άσχημα μπερδεμένα ή λείπουν κάποια UVs. Αυτό μπορεί να συμβεί όταν μια επιφάνεια έχει υποστεί επεξεργασία ή τροποποιηθεί κατά κάποιο τρόπο, και γίνεται δύσκολο να ελέγξουμε ποια UVs μπορεί να λείπουν. Χρειαζόμαστε ένα μοναδικό σύνολο UVs για ένα συγκεκριμένο σκοπό. Το Maya μας επιτρέπει να δημιουργήσουμε UVs για πολυγωνικές επιφάνειες χρησιμοποιώντας μια διαδικασία χαρτογράφησης προβολής, που αναφέρεται επίσης ως UVs χαρτογράφηση. Παρέχονται διάφοροι τύποι χαρτογράφησης προβολής που χαρτογραφούν ό,τι φαίνεται από μια συγκεκριμένη προβολή σε μια επίπεδη 2D προβολή, που μπορεί στη συνέχεια να συσχετιστεί

με την υφή του χάρτη μας, χρησιμοποιώντας τον επεξεργαστή UV υφής. Τα UVs ονομάζονται UV κελιά (shells), και πρέπει να χωρέσουν εντός της περιοχής υφής UV 0 έως 1. Για να εμφανιστεί σωστά η υφή του χάρτη πρέπει τα κελιά UV να μετακινηθούν, ώστε να ευθυγραμμιστούν με τις αντίστοιχες συνιστώσες του χάρτη εικόνας.

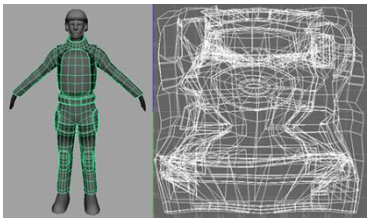
Σε μια χαρτογράφηση υφής μιας επιφάνειας πολυγώνων του μοντέλου είναι συχνά αναγκαίο να τροποποιηθούν τα στοιχεία UV έτσι ώστε να ταιριάζουν με το χάρτη υφής. Τα σχετίζουμε με το μοντέλο και τα τροποποιούμε με ακρίβεια, ενώνουμε μαζί τα όμοια κελιά (Sewing UV shells) για απλοποίηση της διαδικασίας, μετακινούμε και περιστρέφουμε τα UVs για να τα ευθυγραμμίσουμε με το πλαίσιο του χάρτη υφής. Τα UV συνδέονται με τα components του μοντέλου των κορυφών, των ακμών, αλλά και των επιφανειών.

Τα Sewing UV shells συγχωνεύουν μαζί τα κελιά UV μιας κοινής ακμής που έχουμε ορίσει. Μπορούμε να τα μετακινήσουμε και να τροποποιήσουμε ως μεγαλύτερες συνεχόμενες μονάδες υφής, έτσι ώστε να ταιριάζουν τα UVs με την υφή σε πολλές καταστάσεις πιο αποτελεσματικά. Οι χάρτες υφής μπορούν να εμφανίζονται πιο ομοιόμορφοι στα σύνορα της υφής, όταν η υφή εφαρμόζεται σε ένα ή περισσότερα ενωμένα κελιά UVs σε σύγκριση με πολλά ξεχωριστά κελιά UV. Τα ενωμένα κελιά μειώνουν την πιθανότητα μιας ανεπιθύμητης αντιστοιχίας υφής κατά μήκος των ακμών της υφής.

Η υφή ενός μοντέλου δημιουργείται μόνο αφού το μοντέλο έχε ολοκληρωθεί, διαφορετικά οι αλλαγές στο μοντέλο μπορεί να επηρεάσουν τις σχετικές UV συντεταγμένες υφής, που με τη σειρά τους θα επηρεάσουν το πώς εμφανίζεται η υφή με το μοντέλο.

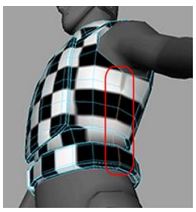
UV unfolding

Ενώ η υφή ενός επίπεδου αντικειμένου είναι αρκετά απλή, η υφής ενός αντικειμένου με καμπύλες μπορεί να δημιουργήσει μερικά προβλήματα. Πρέπει να σχεδιάσουμε μια 2D υφή σε μια κυρτή επιφάνεια, σε βαθουλώματα και εξογκώματα επιφάνειας. Χωρίζοντας το πλέγμα UV σε τμήματα, μπορεί να καταλήξουμε με εκατοντάδες μικρά κομμάτια. Για την αντιμετώπιση αυτών των προβλημάτων χρησιμοποιούμε μια τεχνική που ονομάζεται ξεδίπλωμα (unfolding). Το ξεδίπλωμα ενός πλέγματος UV αναφέρεται στη διαδικασία της κοπής μιας ραφής στο πλέγμα UV (ένα πλέγμα αποτελείται από UVs παρόμοια με το πώς ένα πλέγμα πολυγώνου αποτελείται από κορυφές) και στη συνέχεια εκτυλίσσεται κατά μήκος αυτής της ραφής. Θέτοντας τα UVs με αυτόν τον τρόπο, μπορουμε εύκολα να ζωγραφίσουμε μια υφή στην επιφάνεια 2D, την οποία στη συνέχεια μπορούμε να τυλίξουμε γύρω από το μοντέλο. Πρέπει να χωρίσουμε το πλέμα σε ξεχωριστά τμήματα για την ύφανση, να αντιστοιχίσουμε ένα βασικό μοτίβο για να κρίνουμε την ποιότητα της χαρτογράφησης να δημιουργήσουμε μια επίπεδη χαρτογράφηση, να κόψουμε τα άκρα UV, να ξεδιπλώσουμε το πλέγμα UV και να ξεδιπλώσουμε το πλέγμα χρησιμοποιώντας πάντα τα κατάλληλα εργαλεία του Maya.



Τα UVs για το σώμα, τα χέρια και τα πόδια είναι διασκορπισμένα σε ένα σχέδιο που δεν μοιάζει με το πραγματικό μοντέλο, έτσι είναι δύσκολος ο σχεδιασμός του χάρτη της υφής. Για την απλοποίηση της εργασίας, διαιρούμε το πλέγμα σε πολλά κομμάτια που φτιάχνουμε την υφή στο καθένα ξεχωριστά. Γενικά διαιρούμε το πλέγμα σε κομμάτια που μπορούν να ξεδιπλωθούν περίπου σε τετράγωνο ή ορθογώνιο σχήμα.

Ο χωρισμός του πλέγματος γίνεται μέσω των ακμών ή των όψεων της επιφάνειας (edges ή faces) του μοντέλου, τις οποίες κόβουμε και ράβουμε ανάλογα την επιφάνεια στην οποία θέλουμε να εφαρμόσουμε την υφή, και έπειτα την ξεδιπλώνουμε και την απλώνουμε στον επεξεργαστή ώστε να καλύψει μια σχετικά μεγάλη επιφάνεια για τον ομαλό σχεδιασμό της υφής. Συνήθως χωρίζουμε από το σώμα τα χέρια, τα πόδια, το κεφάλι, την ουρά και το κεντρικό σώμα. Αν έχουν πολύπλοκο σχέδιο τα χωρίζουμε και αυτά σε πιο μικρά τμήματα για την μπροστινή και την πίσω όψη, ώστε να μην επικαλύπτονται τα UVs. Για περισσότερες λεπτομέρειες μπορούμε να τα χωρίσουμε και αυτά σε ακόμα μικρότερα τμήματα, όπως χωρίσαμε το κεφάλι στο ράμφος τα μάτια και το πρόσωπο. Από τα UVs που παίρνουμε, ενώνουμε αυτά των χεριών, και των ποδιών καθώς είναι ακριβώς ίδια λόγω συμμετρίας, ώστε να εξοικονομίσουμε χώρο στο πρώτο τεταρτημόριο και να απλοποιήσουμε την διαδικασία. Για την υφή ενός οργανικού αντικειμένου, ξεκινάμε τη δημιουργία μιας επίπεδης χαρτογράφησης ως βάση, την οποία στη συνέχεια ξεδιπλώνουμε ώστε να μην επικαλύπτονται τα UVs.



Η ξεδίπλωση ενός πλέγματος UV είναι η διαδικασία της κοπής ενός πλέγματος έτσι ώστε να μπορούμε να απλώσουμε ολόκληρη την επίπεδη επιφάνειά του, και πρέπει τα UVs να απλώνονται ομοιόμορφα κατά μήκος της επιφάνειας. Μπορούμε να σταθεροποιήσουμε ένα UV και να απλώσουμε το πλέγμα γύρω από αυτό προς όλες τις κατευθύνσεις, ή προς μία κατεύθυνση. Για να ελέγξουμε αν είναι ομοιόμορφα τοποθετημένα τα UVs, θέτουμε στην επιφάνεια του μοντέλου μια εικόνα με ομοιόμορφα μικρά τετράγωνα άσπρα και μαύρα, ώστε να δούμε σε ποια

σημεία υπάρχει παραμόρφωση. Ενώ ελαχιστοποιείται η στρέβλωση, μπορεί να δημιουργηθεί μια ορατή ραφή στο σημείο που κόβουμε τις άκρες UV. Δυστυχώς, αυτό είναι αναπόφευκτο. Σε γενικές γραμμές είναι καλύτερα να βάλουμε ραφές σε λιγότερο εμφανή σημεία. Μπορούμε επίσης να αυξήσουμε τον αριθμό των τετραγώνων στο σχέδιο τετραγώνων σκακιέρας ώστε να εντοπίσουμε πιο εύκολα τις παραμορφώσεις στο σχήμα. Μετά την ξεδίπλωση των UVs, τα εξάγουμε με τη λήψη ενός στιγμιότυπου της διάταξης των UVs, ώστε να μπορούμε να σχεδιάσουμε την υφή τους.

Δεν έχει σημασία το μέγεθος, η θέση και η κατεύθυνση των UVs στον επεξεργαστή υφής, αρκεί να εφαρμόζουν σωστά με την υφή που σχεδιάσαμε και να μην πανωγράφονται διαφορετικά UVs. Μερικές φορές μπορεί τα UVs ενός πλέγματος να βρίσκονται πολύ κοντά μεταξύ τους δημιουργώντας παραμόρφωση και προβλήματα στην απόδοση της υφής, και έτσι να χρειάζεται να τα απλώσουμε ή μπορεί να θέλουμε να ξεδιπλωθούν εν μέρει. Για να ξεδιπλώσουμε ή να χαλαρώσουμε το πλέγμα και να διορθώσουμε ατέλειες σε περιοχές κελιών χρησιμοποιούμε το UV Smooth tool. Μετά από όλα αυτά δημιουργούμε την υφή με ένα πρόγραμμα σχεδιασμό και το εισάγουμε στο μοντέλο.

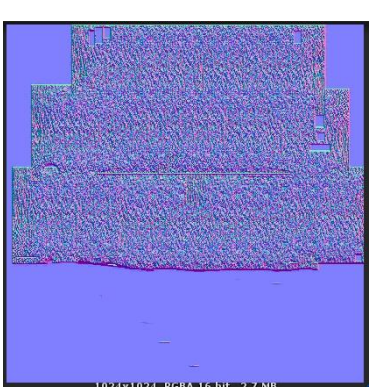
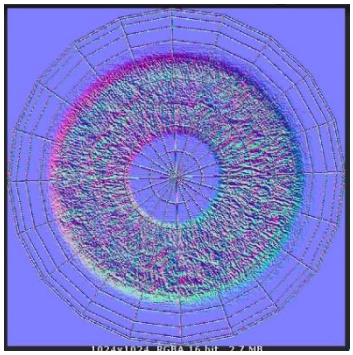
Normal mapping

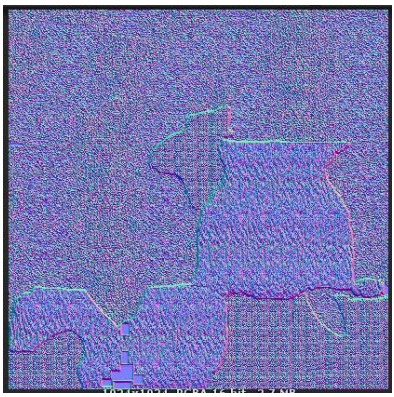
Η κανονική χαρτογράφηση (Normal mapping) είναι μια τεχνική στην οποία μπορούμε να χρησιμοποιήσουμε ένα πλέγμα υψηλής ανάλυσης για να δημιουργήσουμε ένα χάρτη για ένα πλέγμα χαμηλής ανάλυσης. Τα πλέγματα υψηλής ανάλυσης έχουν πιο περίπλοκη γεωμετρία και να επιβαρύνουν το πρόγραμμα και μειώνεται πολύ η απόδοση. Η κανονική χαρτογράφηση χρησιμοποιείται συνήθως σε αυτές τις καταστάσεις για να συλλάβει την λεπτομέρεια ενός πλέγματος υψηλής ανάλυσης με τη γεωμετρία ενός μοντέλου χαμηλής ανάλυσης.

Ένας κανονικός χάρτης διαφέρει από ένα χάρτη υφής από το ότι παράγει μια εικόνα πολλαπλών καναλιών με βάση τα normals του πλέγματος υψηλής ανάλυσης (πηγή). Οι πληροφορίες αυτές χρησιμοποιούνται για να χαρτογραφηθούν λεπτομέρειες του πλέγματος χαμηλή ανάλυση (στόχος) πειστικά, ακόμα κι αν η ίδια η επιφάνεια είναι σχετικά επίπεδη.

Η λεπτομέρεια του αρχικού μοντέλου εξαρτάται από τον αριθμό των πολυγώνων και των συστατικών που αποτελείται. Όσο μικρότερος είναι ο αριθμός αυτός, τόσο πιο εύχρηστο είναι το μοντέλο και με καλύτερη απόδοση ειδικά σε μηχανές παιχνιδιού. Όμως υστερεί σε λεπτομέρειες εικόνας και σχεδιασμού. Αυτό καλύπτεται από το normal mapping που αποδίδει λεπτομέρειες χωρίς να επιβαρύνεται και να μειώνεται η απόδοση του μοντέλου. Για να δημιουργήσουμε ένα κανονικό χάρτη χρειάζονται δύο αρχεία, την πηγή και τον στόχο. Το μοντέλο πηγής είναι γενικά υψηλότερης ανάλυσης και πιο λεπτομερή έκδοση του μοντέλου στόχου. Ο στόχος έχει τη γεωμετρία που θέλουμε για τη σκηνή μας.

Δημιουργήσαμε normal map υφές του πιγκουίνου του κεφαλιού, των ματιών, του ράμφους και του σώματος μέσω της Unity, η οποία τις δημιούργησε αυτόματα πάνω στις αρχικές υφές:



Εφαρμόζοντας αυτές τις υφές normal map, σε συνδυασμό με τις αρχικές απλές υφές πάνω στον πιγκουίνο είναι αισθητή η διαφορά της ανάλυσης καθώς παρατηρούμε ότι το πλέγμα έγινε πιο ομαλό, πιο ρεαλιστικό και με περισσότερες λεπτομέρειες.


ΚΕΦΑΛΑΙΟ 4

ΣΧΕΔΙΑΣΜΟΣ ΠΑΙΧΝΙΔΙΟΥ & ΓΡΑΦΙΚΗ ΔΙΕΠΑΦΗ ΧΡΗΣΤΗ

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο περιγράφουμε τη σχεδίαση και τη γραφική διεπαφή του παιχνιδιού, αναλύοντας το χειρισμό και τη λειτουργία του σε κάθε βήμα.

4.2 Σενάριο

Ο ήρωας του παιχνιδιού είναι ένας πιγκουίνος. Σκοπός του είναι να βρει το θησαυρό της κάθε πίστας, δηλαδή το σημείο τερματισμού, εξουδετερώνοντας τους βίκινγκς που τον κυνηγάνε και τα κανόνια που τον σημαδεύουν. Δευτερεύον στόχος του είναι η βαθμολογία της κάθε πίστας να είναι όσον το δυνατόν υψηλότερη, συλλέγοντας νομίσματα και μπόνους αντικείμενα (ψαράκια).

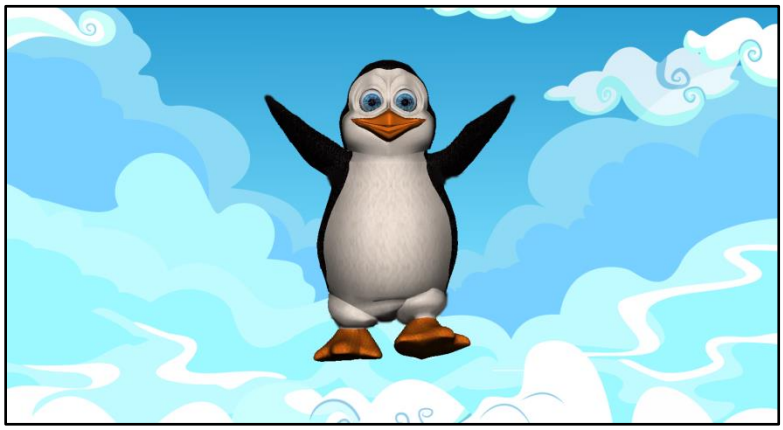
Στο παιχνίδι προσφέρονται τέσσερις διαφορετικοί κόσμοι, που χωρίζονται αναλόγως το θέμα τους και τον τρόπο χειρισμού τους. Δεν μπορούμε να μεταβούμε από τον έναν κόσμο στον άλλον, μόνο από την μία πίστα στην επόμενή της του ίδιου κόσμου. Η εισαγωγή μας στον κάθε κόσμο, και σε κάποια πίστα γίνεται από το μενού.

Οι δύο πρώτοι κόσμοι είναι σχεδιασμένοι για τη πλάγια όρθια θέση της συσκευής, και ο χειρισμός τους γίνεται με κουμπιά αφής. Ο ήρωας μπορεί να περιηγηθεί μέσα στον κόσμο του παιχνιδιού ελεύθερα.

Οι άλλοι δύο κόσμοι είναι σχεδιασμένοι για την όρθια θέση της συσκευής. Στον τρίτο κόσμο ο ήρωας έχει προκαθορισμένη κατεύθυνση πετώντας στον αέρα και ο χειρισμός γίνεται με την κλίση της συσκευής, και με κάποια κουμπιά αφής. Στον τέταρτο κόσμο ο ήρωας μπορεί να πάει όπου θέλει μέσα σε κάποια όρια της πίστας, έχοντας μια προκαθορισμένη ταχύτητα κίνησης καθόλη τη διάρκεια του παιχνιδιού, και έτσι δεν μπορεί να μείνει στάσιμος. Ο χειρισμός του τέταρτου κόσμου γίνεται από την κίνηση και κατεύθυνση των δαχτύλων του χρήστη πάνω στην οθόνη, με την κλίση της συσκευής και με κουμπιά αφής.

4.3 Αναλυτική Περιγραφή Παιχνιδιού

Η εφαρμογή όταν εγκατασταθεί στη συσκευή έχει ένα εικονίδιο με το πρόσωπο του πιγκουίνου. Όταν επιλεχθεί η εφαρμογή, μέχρι να φορτώσει το παιχνίδι στην συσκευή εμφανίζεται μια εικόνα με τον ήρωά μας σε πλάγια θέση της συσκευής πάνω στην οποία εμφανίζεται ο χρόνος φόρτωσης της εφαρμογής.



4.3.1 Μενού

Το παιχνίδι ξεκινάει με το κυρίως μενού, το οποίο περιλαμβάνει ένα κεντρικό κουμπί έναρξης (Play) το οποίο μας εισάγει στο παιχνίδι, και τρία μικρότερα κουμπιά με τα οποία έχουμε πρόσβαση στις οδηγίες του παιχνιδιού, ελέγχεται η ενεργοποίηση και η απενεργοποίηση του ήχου, και ο τερματισμός του παιχνιδιού.

Στην συνέχεια οδηγούμαστε σε ένα άλλο μενού με τέσσερις κόσμους στους οποίους μπορούμε να περιηγηθούμε, τη SnowLand, τη WaterLand, την JumpLand και τη RunLand. Κάθε φορά στην οθόνη φαίνεται μόνο ο ένας κόσμος. Δεξιά και αριστερά από αυτόν βρίσκονται οι υπόλοιποι, στους οποίους μπορούμε να πάμε είτε πατώντας τα βέλη που βρίσκονται δεξιά και αριστερά από αυτόν, είτε σύροντας το δάχτυλό μας πάνω στην οθόνη αντίστοιχα.

Μόνο από το μενού αυτού μπορούμε να αλλάξουμε κόσμο, και όχι μέσα από τον έναν κόσμο στον άλλον. Η επιστροφή μας στο μενού μέσα από έναν κόσμο γίνεται με το κουμπί της Παύσης, που αναλύεται παρακάτω.

Με την επιλογή του κάθε κόσμου εμφανίζονται οχτώ πίστες αριθμημένες σε οχτώ μικρά μπλε κουτάκια, οι οποίες είναι αρχικά κλειδωμένες εκτός από την πρώτη. Για να ξεκλειδώσει η κάθε πίστα θα πρέπει να τερματιστεί επιτυχώς η προηγούμενή της. Στην κάθε πίστα εμφανίζεται το μεγαλύτερο σκορ του χρήστη όσον αφορά τα νομίσματα και τα ψαράκια μπόνους που συνέλεξε. Αν ο χρήστης συλλέξει και τα τρία ψαράκια, η συγκεκριμένη πίστα στο μενού από μπλε γίνεται χρυσή.

Μπορούμε να έχουμε πρόσβαση σε οποιαδήποτε ξεκλειδωμένη πίστα και να εισαχθούμε σε αυτή ανεξαρτήτως σειράς μόνο μέσα από το μενού αυτό. Μέσα από την κάθε πίστα του κόσμου, μπορούμε να μεταβούμε μόνο στην επόμενή της εφόσον είναι ξεκλείδωτη.

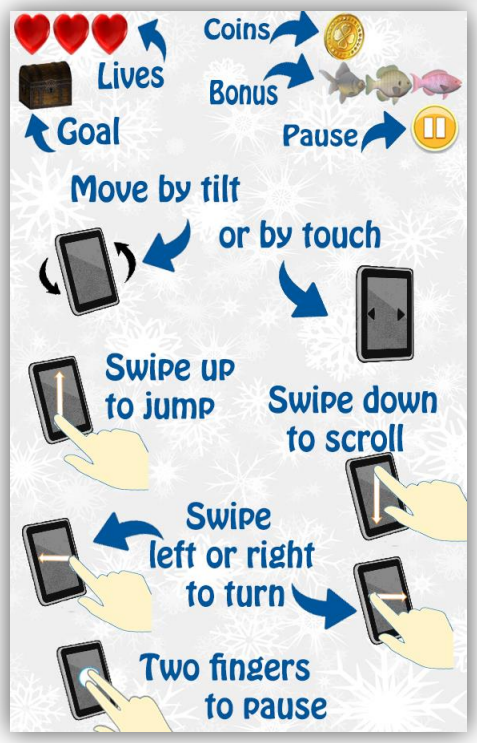
Στο πάνω μέρος της οθόνης υπάρχουν τέσσερα επιπλέον κουμπιά. Τα δύο εμφανίζονται στο μενού των κόσμων και στο μενού με τις πίστες, το ένα για τον έλεγχο του ήχου, και το άλλο για την επιστροφή στο αρχικό μενού. Τα άλλα δύο κουμπιά εμφανίζονται μόνο στο μενού των πιστών, το ένα για την επιστροφή στο μενού των κόσμων, και το άλλο είναι για τη διαγραφή όλων των αποθηκευμένων δεδομένων αυτού του κόσμου, κλειδώνοντας πάλι όλες τις πίστες εκτός από την πρώτη και διαγράφοντας όλα τα σκορ ώστε να ξεκινήσουμε από την αρχή.

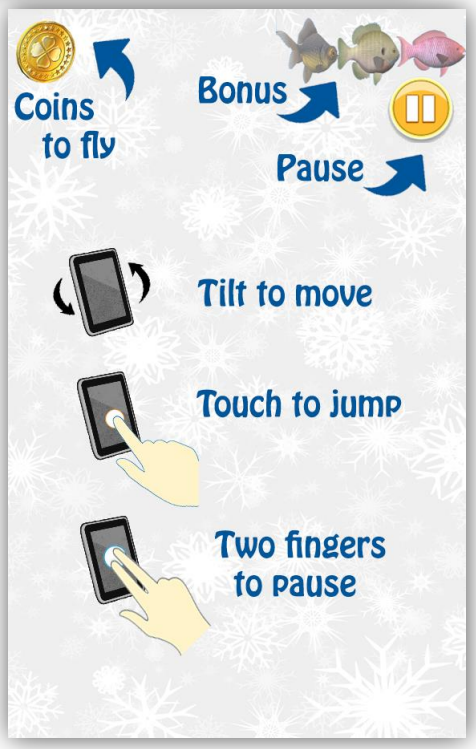
Το μενού είναι σχεδιασμένο ώστε να κρατάμε τη συσκευή παράλληλα ως προς το έδαφος με το κεντρικό κουμπί στο πλάι, είτε στο δεξί είτε στο αριστερό χέρι. Αν περιστρέψουμε τη συσκευή από τη μία πλάγια θέση στην άλλη τότε θα περιστραφεί και το μενού αντίστοιχα. Αν περιστρέψουμε τη συσκευή σε όρθια θέση με το κεντρικό κουμπί προς τα κάτω, το μενού θα είναι σαν να βρισκόταν σε πλάγια θέση με το κουμπί στο δεξί χέρι. Περιστρέφοντας τη συσκευή σε όρθια θέση ανάποδα με το κεντρικό κουμπί προς τα πάνω, το μενού θα περιστραφεί σαν να βρισκόταν σε πλάγια θέση με το κουμπί στο αριστερό χέρι. Αν αφήσουμε το κινητό κάτω δεν θα περιστραφεί. Σε όλες τις περιπτώσεις εξακολουθεί να είναι ενεργό και λειτουργικό.

Όταν ο χρήστης επιλέξει μια πίστα, μέχρι να φορτώσει και να ξεκινήσει να παίζει, εμφανίζονται στην οθόνη οι οδηγίες της συγκεκριμένης πίστας, οι οποίες συμπεριλαμβάνουν πληροφορίες όσον αφορα το στόχο του ήρωα, τους εχθρούς, τον τρόπο με τον οποίο μπορεί να κινηθεί, καθώς και τη λειτουργία και το σκοπό του κάθε κουμπιού. Στο κάτω μέρος της εικόνας αυτής υπάρχει μια μπάρα (Loading Bar) η οποία αυξάνεται αναλογικά με το χρόνο που

απομένει μέχρι να ξεκινήσει το παιχνίδι, ώστε να ειδοποιεί για τον χρόνο αναμονής και εκκίνησης της πίστας.

Οι οδηγίες για τις πίστες SnowLand και WareLand είναι ίδιες, καθώς από το παιχνίδι αλλάζει μόνο το περιβάλλον, και εμφανίζονται σε πλάγια θέση της συσκευής, όπως είναι οριοθετημένοι αυτοί οι κόσμοι. Οι οδηγίες για τη JumpLand και τη RunLand είναι διαφορετικές και εμφανίζονται σε όρθια θέση της συσκευής για να προηδεάσουν τον χρήστη να την περιστρέψει πριν ξεκινήσει το παιχνίδι, καθώς οι κόσμοι αυτοί είναι σχεδιασμένοι να λειτουργούν μόνο σε όρθια θέση της συσκευής.

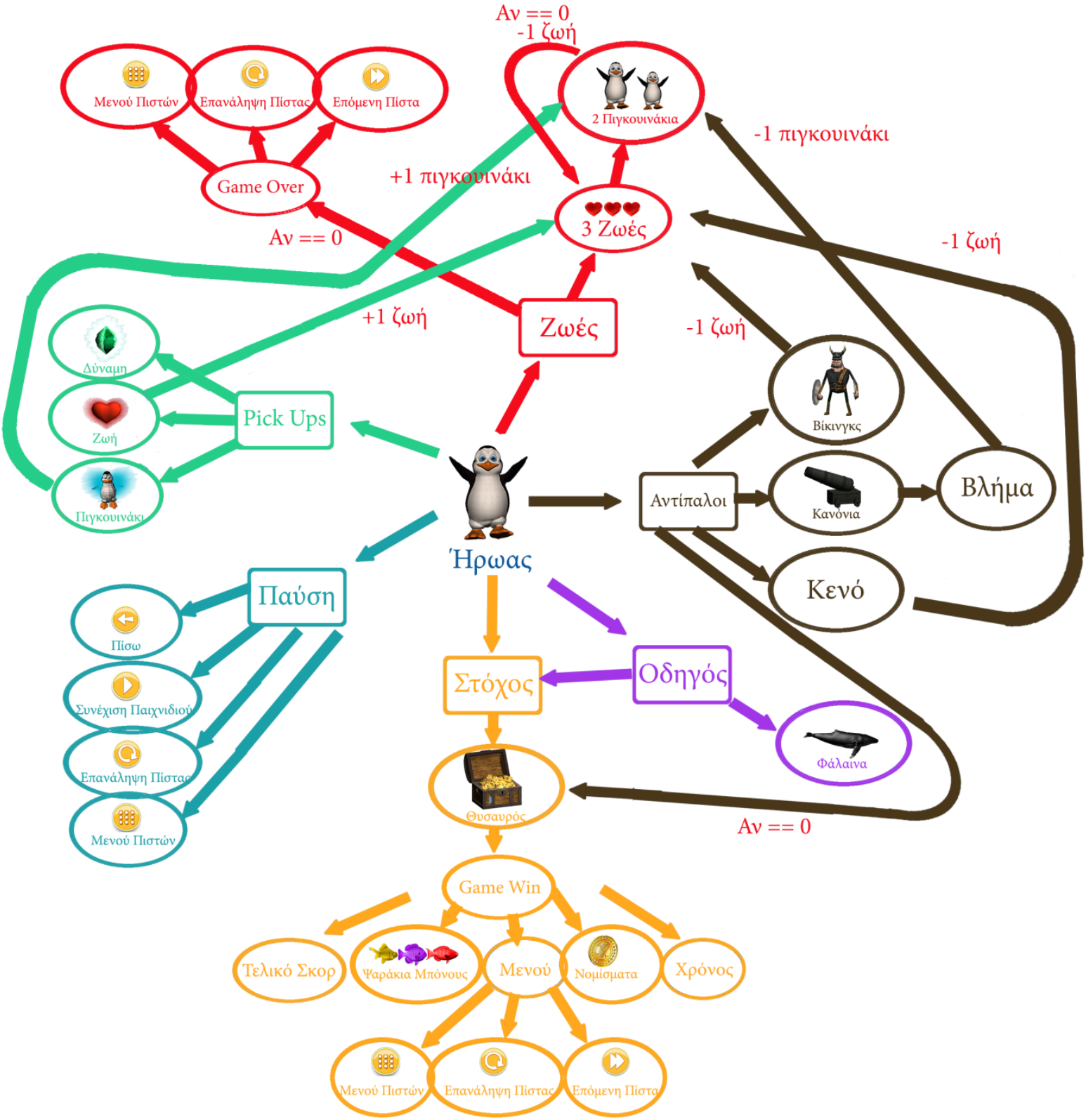




4.3.2 SnowLand

Στον κόσμο αυτό έχουμε διαμορφώσει το παιχνίδι έτσι ώστε να είναι ένας συνδυασμός παιχνιδιού υπολογιστών με παιχνίδι για κινητά. Παραπέμπει περισσότερο στα παιχνίδια που είναι γνωστά για υπολογιστές αλλά είναι σχεδιασμένο έτσι ώστε να είναι κατάλληλο και εύχρηστο σε κινητή συσκευή.

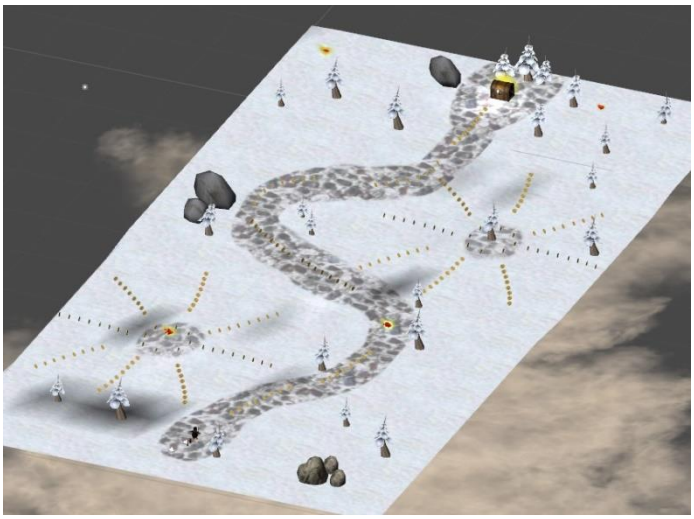
Για να παίξουμε στον κόσμο αυτό μπορούμε να κρατάμε τη συσκευή όπως θέλουμε. Λόγω όμως πολυπλοκότητας του χειρισμού αλλά και της ορατότητας της πίστας, είναι καλύτερο να κρατάμε τη συσκευή παράλληλα προς το έδαφος με το κεντρικό κουμπί στο δεξί ή το αριστερό χέρι και όχι σε όρθια θέση

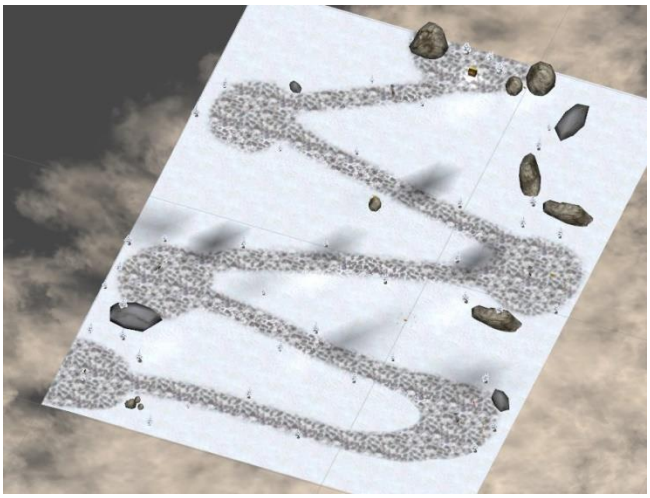

Αν == 0
-1 ζωή
Μενού Πιστών
Επανάληψη Πίστας
Επόμενη Πίστα
2 Πιγκουινάκια
-1 πιγκουινάκι
Game Over
+1 πιγκουινάκι
3 Ζωές
Αν == 0
-1 ζωή
+1 ζωή
Ζωές
-1 ζωή
Δύναμη
Ζωή
Pick Ups
Πιγκουινάκι
Βίκινγκς
Αντίπαλοι
Κανόνια
Βλήμα
Ήρωας
Κενό
Παύση
Πίσω
Στόχος
Οδηγός
Συνέχιση Παιχνιδιού
Επανάληψη Πίστας
Φάλαινα
Μενού Πιστών
Θυσαυρός
Αν == 0
Game Win
Τελικό Σκορ
Ψαράκια Μπόνους
Μενού
Νομίσματα
Χρόνος
Μενού Πιστών
Επανάληψη Πίστας
Επόμενη Πίστα

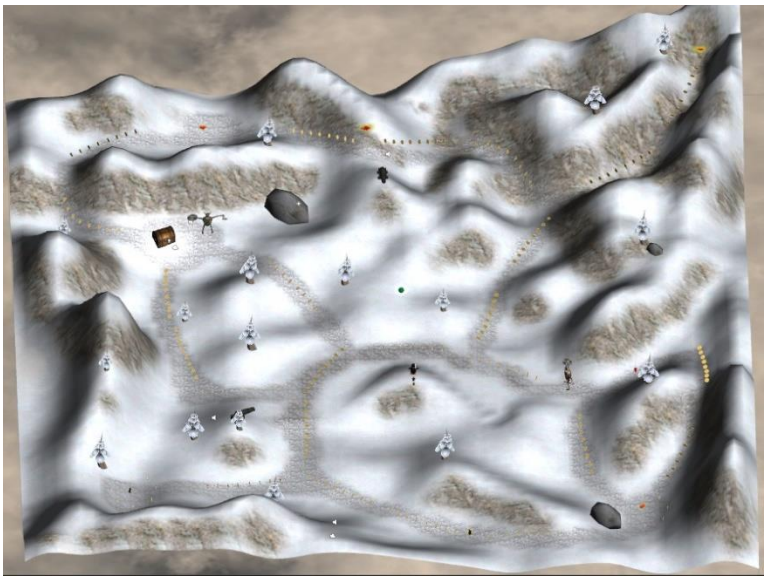
Πίστες

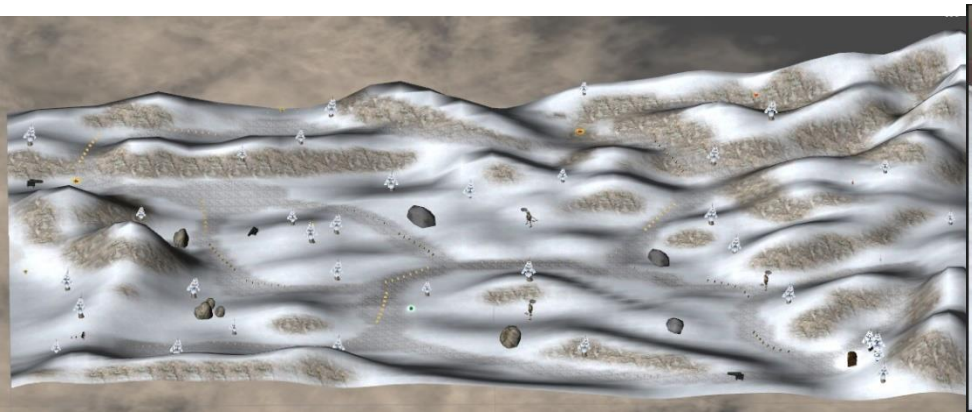
Οι πίστες περιβάλλονται κυρίως από χιόνια, αλλά και από βουνά, σύννεφα, θάλλασα, καταράκτες, βράχους, δέντρα και μονοπάτια. Ο πιγκουίνος μπορεί να εξερευνήσει την κάθε πίστα και να κινηθεί όπου θέλει, ακόμα και να κολυμπήσει, προσέχοντας να μην βρεθεί έξω από τα όρια του εδάφους γιατί θα πέσει στο κενό (σύννεφα) και θα σκοτωθεί. Σε σημεία όμως, που υπάρχει νερό στα όρια του εδάφους της πίστας, δεν του επιτρέπεται να προχωρήσει για να μην πέσει κάτω, ως βοήθεια στο χρήστη λόγω περιορισμένης ορατότητας.

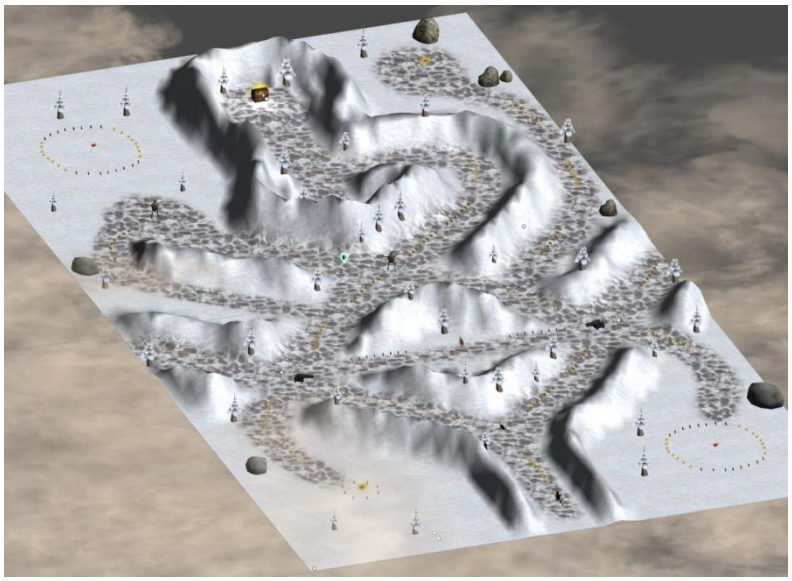
Ο χρήστης πάντα βλέπει το πίσω μέρος του ήρωα και ότι βρίσκεται μπροστά από αυτόν. Η κάμερα βρίσκεται πάνω από το κεφάλι του πιγκουίνου στο πίσω μέρος και κοιτάει προς τα μπροστά. Όσο προχωράει ο ήρωας προχωράει και η κάμερα με μια ομαλή κίνηση μέσω φίλτρων. Έτσι ο χρήστης πάντα βλέπει που βρίσκεται ο ήρωάς του αλλά και τι αντικρίζει κάθε στιγμή. Παρακάτω παρουσιάζουμε την οργάνωση της κάθε πίστας και δίπλα την οπτική γωνία του χρήστη.

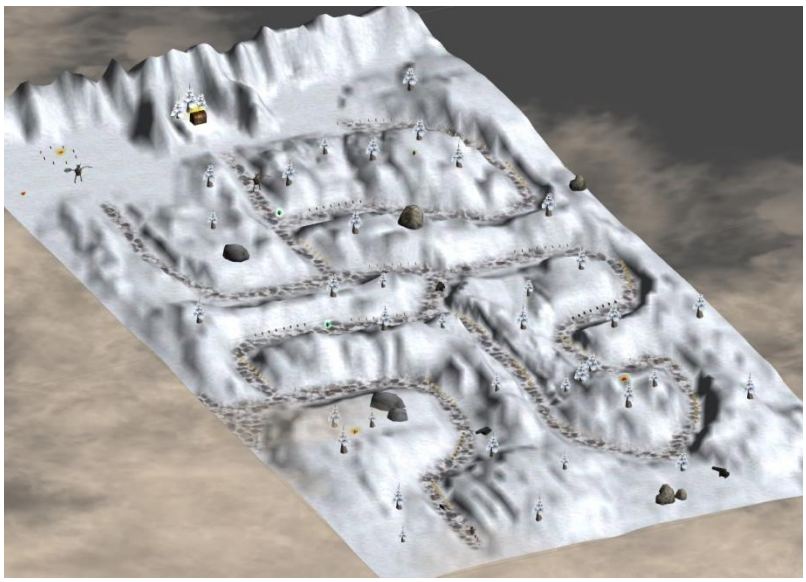


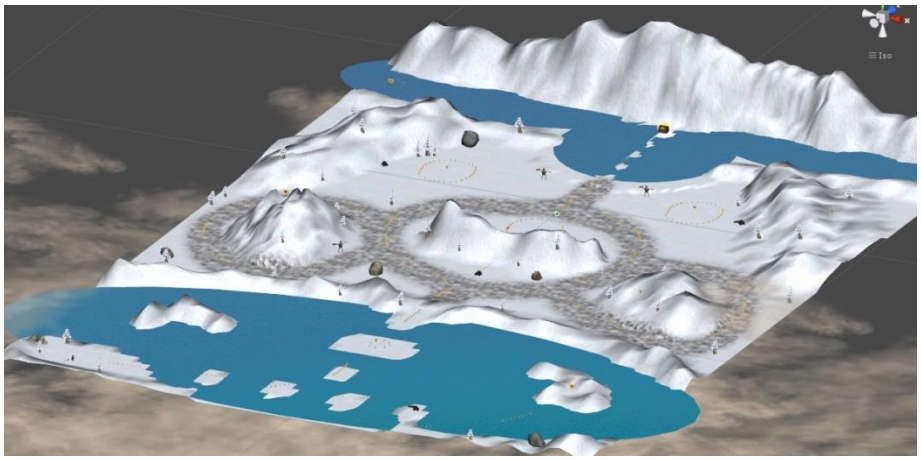






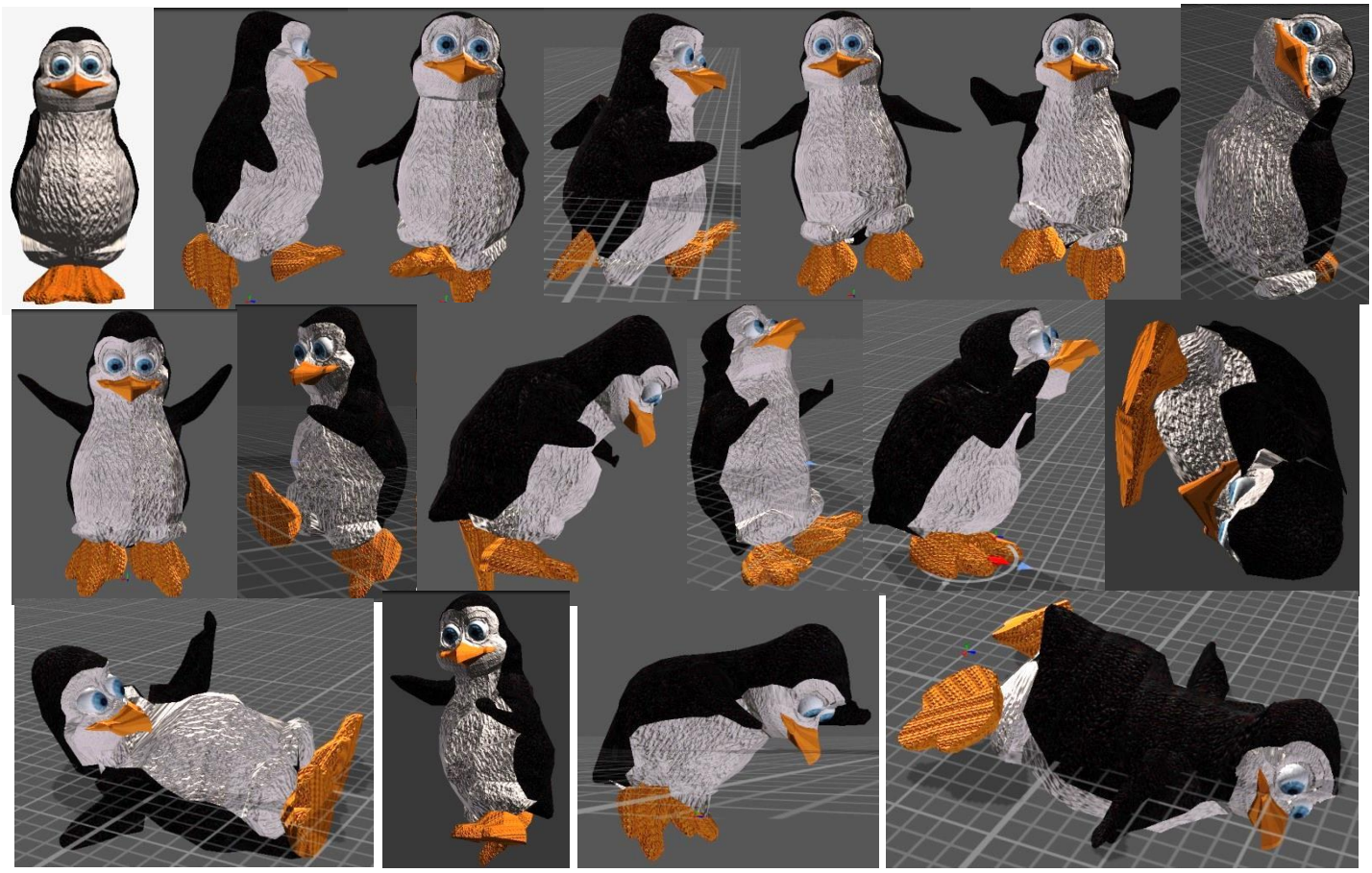






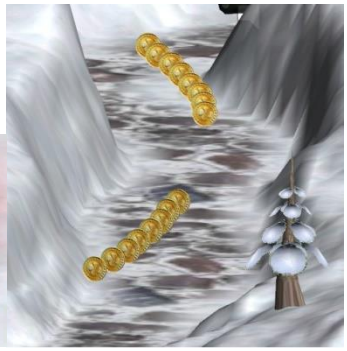
Κινήσεις

Για κάθε του κίνηση υπάρχει ένα διαφορετικό κινούμενο σχέδιο (animation) που του προσφέρει ζωή και αληθοφάνεια στην κίνησή του. Περιμένει να ξεκινήσουμε, περπατάει, στρίβει, τρέχει, πηδάει, πηδάει διπλά, περπατάει και τρέχει προς τα πίσω, κολυμπάει, πετάει χιονόμπαλες στους κακούς, κοιτάει να βρει το θησαυρό, πετάει, προσγειώνεται, πέφτει, χτυπάει, πεθαίνει και πανηγυρίζει όταν κερδίζει.



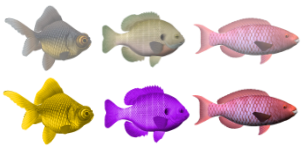
Πόντοι

Στο δρόμο του υπάρχουν χρυσά νομίσματα που περιστρέφονται στον αέρα. Όσα περισσότερα μαζέψει τόσο μεγαλύτερο σκορ θα έχει στο τέλος της πίστας. Κάθε νόμισμα δίνει από έναν βαθμό στην τελική βαθμολογία. Επιπλέον, σε κάθε πίστα υπάρχουν τρία φωτεινά ψαράκια μπόνους διαφορετικού χρώματος, ένα κίτρινο, ένα μωβ και ένα κόκκινο, τα οποία περιστρέφονται και αυτά στον αέρα. Το κίτρινο δίνει εκατό



πόντους στην τελική βαθμολογία, το μωβ δίνει 200, και το κόκκινο δίνει 300.

Ετικέτες



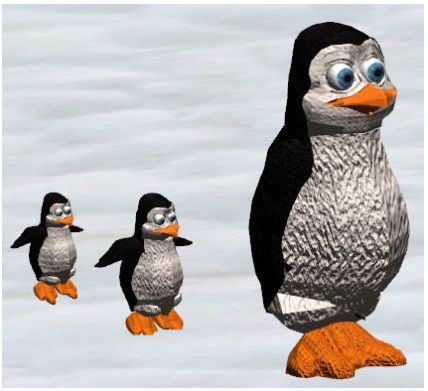
Στο πάνω μέρος της οθόνης αριστερά φαίνονται τα τρία ψαράκια με αχνό χρώμα, και όταν συλλέξει κάποιο από αυτά παίρνει έντονο χρώμα. Από κάτω αναγράφεται ο αριθμός των νομισμάτων που έχει συλλέξει.

Στο αριστερό μέρος υπάρχουν τρεις κόκκινες καρδούλες οι οποίες φανερώνουν τον αριθμό των ζωών του ήρωα. Με κάθε ζωή που χάνει αφαιρείται και μία καρδούλα. Όταν χάσει όλες τις ζωές παγώνουν όλες οι καρδιές και γίνονται γαλάζιες.

Κάτω από τις καρδούλες υπάρχουν δύο εικονίδια, ένα κανόνι και ένας βίκινγκ, που ενημερώνουν τον αριθμό των αντιπάλων που απομένουν σε κάθε πίστα, τα οποία εμφανίζονται μόνο εφόσον υπάρχουν στην συγκεκριμένη πίστα. Δίπλα στο κάθε εικονίδιο αναγράφεται ο αριθμός των αντίστοιχων αντιπάλων που υπάρχουν αρχικά στην πίστα. Κάθε φορά που εξουδετερώνουμε ένα αντίπαλο, ο αριθμός αυτός μειώνεται κατά ένα. Έτσι ο χρήστης μπορεί να ξέρει κάθε φορά πόσοι αντίπαλοι απομένουν ακόμα να σκοτώσει, ώστε να μπορέσει να τερματίσει.

Ζωές

Ο πιγκουίνος σε κάθε πίστα έχει τρεις ζωές και τον ακολουθούν δύο μικρά πιγκουινάκια τα οποία τον βοηθάνε να μην σκοτωθεί και του δίνουν παραπάνω ζωή. Χάνει ένα πιγκουινάκι με κάθε χτύπημα που δέχεται από τα κανόνια. Όταν χάσει και τα δύο, χάνει μια ζωή. Κάθε φορά που χάνει μία ζωή, ξεκινάει από την αρχική του θέση. Μαζί με αυτόν επανέρχονται στην αρχική τους θέση και επανενεργοποιούνται οι δύο μικροί πιγκουίνοι καθώς και οι αντίπαλοί του ακόμα και αν τους είχε εξουδετερώσει πριν χάσει. Επίσης στην αρχική τους θέση επανέρχονται και τα αντικείμενα που του δίνουν δυνάμεις, εν αντιθέση με τα νομίσματα και τα ψαράκια που έχει ήδη συλλέξει τα οποία παραμένουν προσθεσμένα στο σκορ.



Αντίπαλοι

Υπάρχουν δύο διαφορετικοί αντίπαλοι, τα κανόνια και οι βίκινγκς, οι οποίοι ποικίλουν στον αριθμό αναλόγως την κάθε πίστα. Μόλις ο ήρωας πλησιάσει αρκετά τα κανόνια, τον σημαδεύουν και εκτοξεύουν βλήματα κατά πάνω του. Η απόσταση με την οποία εντοπίζουν τα κανόνια τον ήρωα αλλάζει ανάλογα με την δυσκολία της πίστας, όσο πιο δύσκολη είναι, τόσο μεγαλύτερη είναι και η απόσταση από την οποία εντοπίζεται ο πιγκουίνος. Τα βλήματα εκτοξεύονται κάθε δύο δευτερόλεπτα και τα κανόνια σημαδεύουν τον ήρωα με μια μικρή καθυστέρηση ως προς την κίνησή του, ώστε να έχει την ευκαιρία να τους ξεφύγει και να μπορέσει να τα καταστρέψει. Αν το βλήμα πετύχει τον ήρωα, τότε αυτός χάνει ένα πιγκουινάκι

και περιστρέφεται γύρω από τον εαυτό του ώστε να χάσει ο χρήστης τον προσανατολισμό του και να δυσκολευτεί να εντοπίσει πάλι το κανόνι. Όταν χτυπηθεί δύο φορές χάνει και τα δύο πιγκουινάκια και περπατάει πληγωμένος καθώς δεν έχει άλλη ευκαιρία να ζήσει αν ξαναχτυπηθεί. Με το τρίτο χτύπημα χάνει μία ζωή και επανέρχεται στην αρχική του θέση και έχει πάλι και τα δύο πιγκουινάκια.

Οι βίκινγκς αρχικά κάθονται ακίνητοι. Όταν τους πλησιάσει ο πιγκουίνος σε κάποια απόσταση, γυρίζουν προς το μέρος του και τον κοιτάνε περιμένοντας να επιτεθούν. Αν τους πλησιάσει περισσότερο τότε τρέχουν κατά πάνω του να τον πιάσουν. Αν κινηθεί ο πιγκουίνος θα τον κυνηγήσουν και θα τον ακολουθήσουν μέχρι να τους σκοτώσει ή να απομακρυνθεί αρκετά από αυτούς και να μην τον βλέπουν. Η απόσταση στην οποία τον εντοπίζουν και τον κυνηγάνε διαφέρει ανάλογα με τη δυσκολία της κάθε πίστας. Αν ο βίκινγκς πιάσει τον ήρωα πεθαίνει αμέσως και χάνει μια ζωή, και έπειτα και οι δύο επιστρέφουν στην αρχική τους θέση.

Ο ήρωας πρέπει επίσης να προσέχει να πατάει μόνο πάνω στη γη και να μην πέσει στο κενό γιατί θα πεθάνει και θα χάσει μια ζωή.

Επίθεση

Ο πιγκουίνος μπορεί να εξουδετερώσει τους αντιπάλους του πετώντας τους χιονόμπαλες. Μπορεί να πετάξει όσες θέλει κάθε φορά χωρίς περιορισμό αφού γύρω του υπάρχουν παντού χιόνια. Όταν χτυπήσει ένα κανόνι με μια χιονόμπαλα, τότε το δημιουργείται μια έκρηξη που καταστρέφει το πάνω μέρος του κανονιού (αυτό που πυροβολάει). Όταν χτυπήσει έναν βίκινγκ, τότε

εκείνος κάνει μια περιστροφή γύρω από τον εαυτό του φωνάζοντας, και μετά εξαφανίζεται.

Τέλος παιχνιδιού

Αν χάσει και τις τρεις ζωές, το παιχνίδι τελείωνει και εμφανίζεται ένα μήνυμα τέλους παιχνιδιού. Έπειτα εμφανίζεται μια καρτέλα με τρία κουμπια, ένα για την επιστροφή στο μενού με τις πίστες αυτού του κόσμου, ένα κουμπί για την επανάληψη της ίδιας πίστας, και ένα κουμπί για την έναρξη της επόμενης πίστας αν και μόνο αν έχει τερματιστεί επιτυχώς η τωρινή πίστα έστω μία φορά, διαφορετικά δεν μπορεί να προχωρήσει στην επόμενη.

Pick Ups

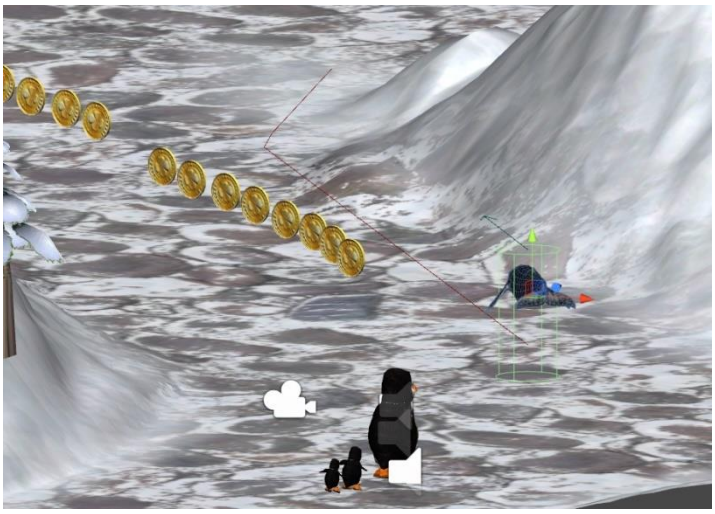
Κρυμμένα στην πίστα υπάρχουν κάποια αντικείμενα που βοηθάνε τον ήρωά μας. Αν έχει χάσει κάποια από τις τρεις ζωές του, οι καρδούλες που αιωρούνται του προσθέτουν μία ζωή ακόμα. Αν έχει ήδη τρεις ζωές και πάρει ακόμα μία δεν έχει παραπάνω ζωές, αλλά εξακολουθεί να έχει τρεις.

Αν έχει χτυπηθεί από κάποιο κανόνι και έχει χάσει κάποιο πιγκουινάκι, τα ιπτάμενα πιγκουινάκια το αναπληρώνουν. Δεν μπορεί όμως να έχει περισσότερα από δύο πιγκουινάκια, οπότε αν δεν έχει χάσει κάποιο από αυτά, και πάρει άλλο ένα δεν θα προστεθεί σε αυτά που τον ακολουθούν.

Υπάρχει ένα μαγικό σμαράγδι το οποίο δίνει δύναμη στον ήρωά μας. Μόλις το πάρει γίνεται διπλάσιος στο μέγεθος και αθάνατος για οχτώ δευτερόλεπτα. Όσο διαρκεί η επίδραση του μαγικού σμαραγδιού δεν μπορεί να πεθάνει ούτε από τους βίκινγκς ούτε από τα κανόνια, παρά μόνο αν πέσει στο κενό. Παράλληλα μπορεί να εξουδετερώσει τους αντιπάλους του.

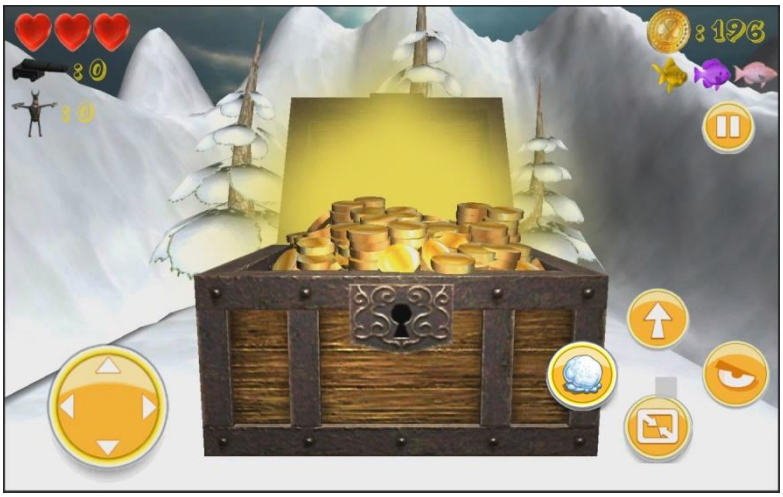
Οδηγός

Σε δυσκολότερες πίστες υπάρχουν λαβύρινθοι και ο πιγκουίνος δεν μπορεί εύκολα να βρει το δρόμο του. Για τον λόγο αυτό έχουμε τοποθετήσει μία φάλαινα για να οδηγεί τον ήρωα προς το θησαυρό. Της προσθέσαμε τεχνητή νοημοσύνη για να κατευθύνεται στο στόχο από την πιο σύντομη διαδρομή αποφεύγοντας όλα τα εμπόδια στο δρόμο της. Για να ξεκινήσει να οδεύει προς το θησαυρό πρέπει να την πλησιάσει ο πιγκουίνος. Αν απομακρυνθεί πολύ από αυτήν, τότε εκείνη τον περιμένει να γυρίσει για να συνεχίσει να προχωράει.



Στόχος

Σκοπός του πιγκουίνου για να τερματίσει την πίστα είναι να βρει το σεντούκι με το θησαυρό, ώστε να μπορέσει να προχωρήσει στην επόμενη. Όταν φτάσει στο σεντούκι ο πιγκουίνος πανηγυρίζει, αλλάζει η μουσική, εμφανίζεται ένα μήνυμα νίκης και στη συνέχεια εμφανίζεται η τελική βαθμολογία. Σε δυσκολότερες πίστες, το σεντούκι του θυσαυρού δεν ανοίγει αν ο ήρωας δεν έχει σκοτώσει όλους τους αντιπάλους του, οπότε και δεν μπορεί να τερματίσει. Αν πάει στο σεντούκι θα του εμφανίσει ένα μήνυμα που θα τον ενημερώνει πως πρέπει πρώτα να σκοτώσει τους εχθρούς του. Μόλις σκοτώσει όλους τους εχθρούς, η κάμερα δείχνει να ανοίγει το σεντούκι του θυσαυρού, ώστε να ειδοποιηθεί ο παίκτης ότι τώρα μπορεί να πάει στο σημείο τερματισμού.



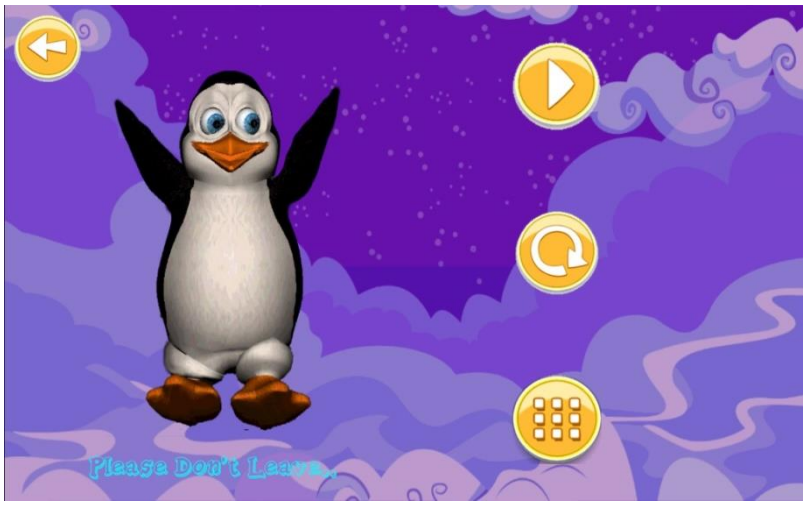
Νίκη

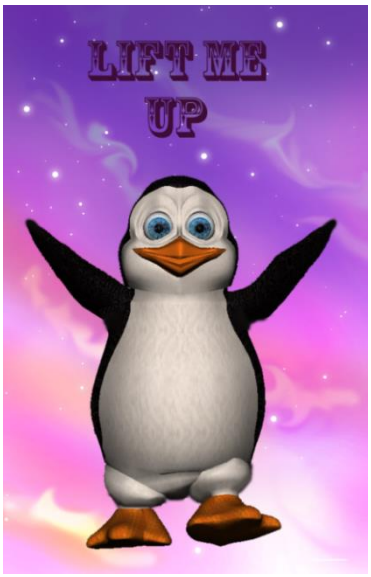
Όταν νικήσει αναγράφεται η τελική βαθμολογία, η οποία διαμορφώνεται από τα νομίσματα και τα ψαράκια που συνέλεξε καθώς και από το χρόνο που χρειάστηκε για να τερματίσει. Παίρνει ένα βαθμό από κάθε νόμισμα, εκατό βαθμούς για κάθε ψαράκι και μείον δέκα βαθμούς για κάθε λεπτό που έκανε μέχρι να νικήσει. Επίσης φαίνεται αναλυτικά πως προέκυψε η βαθμολογία καθώς αναγράφονται ο αριθμος των νομισμάτων, ο χρόνος του παιχνιδιού και εμφανίζονται με έντονο χρώμα τα ψαράκια που μάζεψε ενώ εκείνα τα οποία δεν κατάφερε να μαζέψει είναι με αχνό χρώμα. Στο κάτω μέρος υπάρχουν τρία κουμπιά, ένα για την επιστροφή στο μενού των πιστών αυτού του κόσμου, ένα την επανάληψη της πίστας και ένα για την έναρξη της επόμενης.

Παύση

Ο χρήστης μπορεί να διακόψει το παιχνίδι οποιαδήποτε στιγμή πατώντας το κουμπί παύσης που βρίσκεται πάνω δεξιά, ή πιέζοντας τρία δάχτυλα ταυτόχρονα στην οθόνη. Το παιχνίδι θα σταματήσει στο σημείο που ζητήθηκε η παύση και θα εμφανιστεί ένα μενού με τέσσερα κουμπιά. Ένα για την επιστροφή πίσω στο παιχνίδι, ένα για την συνέχιση του παιχνιδιού από το σημείο που σταμάτησε, ένα για την επανάληψη από την αρχή και ένα για την είσοδο στο μενού με τις πίστες αυτού του κόσμου και τον τερματισμό της πίστας. Ουσιαστικά τα δύο πρώτα κουμπιά κάνουν ακριβώς το ίδιο πράγμα. Παρόλα αυτά όμως τοποθετήθηκαν και τα δύο γιατί η επιλογή “Πίσω” είναι πιο φιλική προς τον χρήστη καθώς νιώθει πιο σίγουρος ότι θα επιστρέψει από εκεί που σταμάτησε.

Άλλος ένας τρόπος παύσης του παιχνιδιού είναι αφήνοντας τη συσκευή κάτω ή να δίνοντάς της τέτοια κλίση σαν να την είχαμε αφήσει σε μια επιφάνεια. Όσο δεν κρατάμε τη συσκευή το παιχνίδι σταματάει να παίζει. Μόλις την σηκώσουμε πάλι στα χέρια μας το παιχνίδι ξεκινάει από το σημείο που το αφήσαμε.





Χειρισμός

Η κίνηση γίνεται με ένα χειριστήριο αφής (joystick) στην αριστερή μεριά της οθόνης. Ανάλογα την κατεύθυνση του δαχτύλου του χρήστη πάνω στο χειριστήριο κινείται και περιστρέφεται ο πιγκουίνος καθώς επίσης και το χειριστήριο ώστε να δίνει την ψευδαίσθηση ενός πραγματικού χειριστηρίου. Η κίνηση μπορεί να είναι συνεχόμενη χωρίς να χρειάζεται να βγάζει ο χρήστης το δάχτυλο πάνω από το χειριστήριο.

Στην δεξιά μεριά, στο ίδιο ύψος με το χειριστήριο, υπάρχουν τέσσερα κουμπιά αφής. Το κουμπί με το βέλος προς τα πάνω είναι για να πηδάει ο ήρωας. Αν το πατήσουμε μία φορά

πηδάει μία φορά. Αν το πατήσουμε δύο φορές, σχετικά κοντά η μία φορά από την άλλη, τότε θα πηδήξει δύο φορές ώστε να πάει πιο ψηλά και να βρίσκεται στον αέρα με μεγαλύτερη διάρκεια.

Το κουμπί με την χιονόμπαλα είναι για να πετάει χιονόμπαλες για να αποκρούει τους αντιπάλους του. Όσες φορές το πατάμε, τόσες χιονόμπαλλες πετάει.

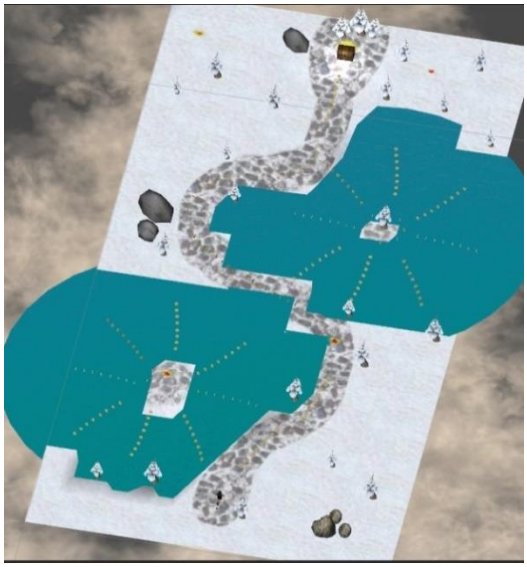
Το κουμπί με το μάτι πρέπει να πατηθεί συνεχόμενα για να λειτουργήσει. Όσο το πιέζουμε η κάμερα αλλάζει θέση και πλησιάζει το κεφάλι του πιγκουίνου, ο οποίος βλέπει προς το θησαυρό και τον στοχεύει με ακτίνες λέιζερ που βγαίνουν από τα μάτια του, ώστε να καταλάβει ο χρήστης προς τα που πρέπει να κατευθυνθεί. Αν βρισκόμαστε σε δυσκολότερη πίστα στην οποία υπάρχει η φάλαινα που μας δείχνει το δρόμο για το θησαυρό, το κουμπί αυτό μας δείχνει τη φάλαινα και όχι το σεντούκι καθώς δεν μπορεί να φανεί λόγω του λαβυρίνθου. Όταν αφήσουμε το κουμπί η κάμερα επανέρχεται στην θέση της.

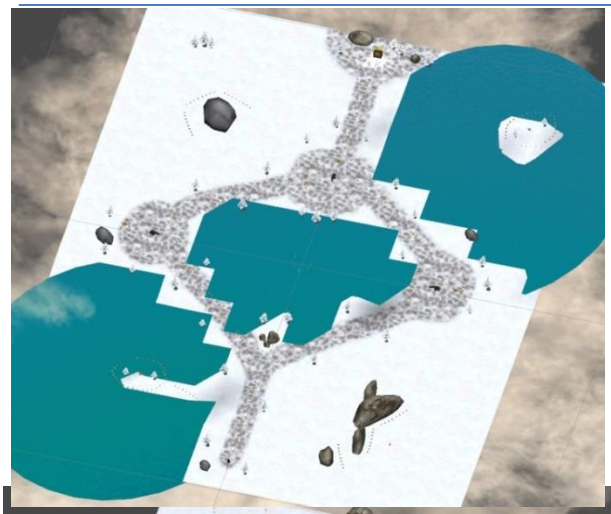
Το κουμπί με το τετράγωνο και το βέλος συμβολίζει την σμίκρυνση της εικόνας. Αυτό σημαίνει ότι όσο πατάμε το κουμπί (συνεχόμενα), η κάμερα αλλάζει θέση και σηκώνεται ψηλά ώστε να μπορούμε να δούμε τι γίνεται γύρω μας και προς τα που να κατευθυνθούμε αναλόγως με αυτό που ψάχνουμε. Μόλις αφήσουμε το κουμπί η κάμερα επιστρέφει στην θέση της.

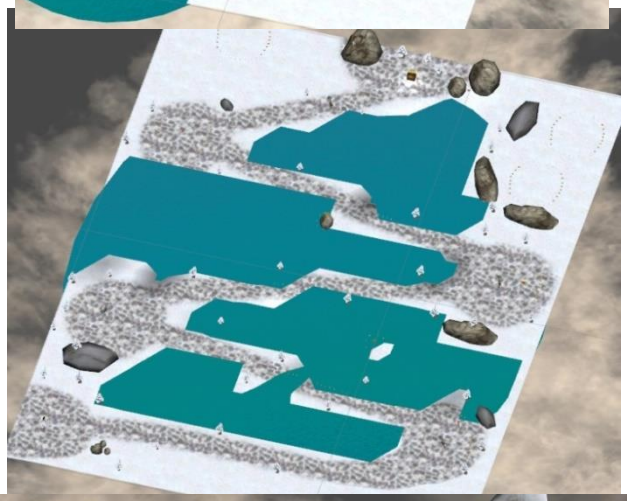
4.3.3 WaterLand

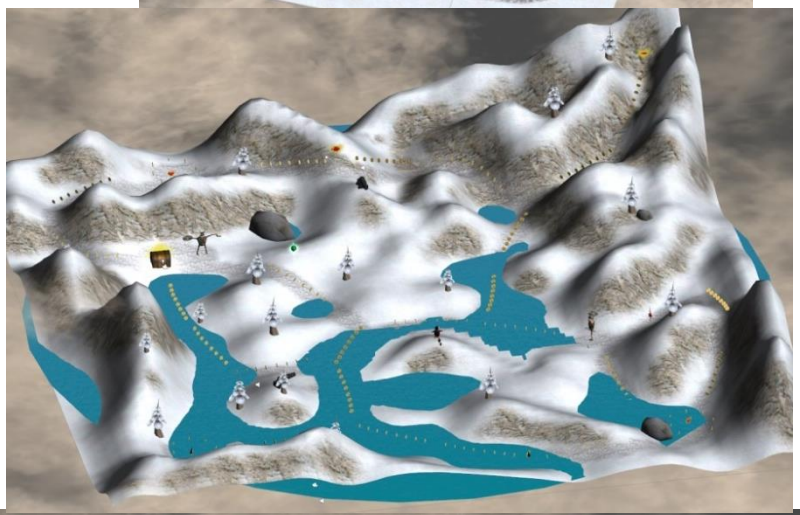
Η Waterland και η Snowland είναι παρόμοιες πίστες με μικρές διαφορές. Η ουσιώδης διαφορά ανάμεσα σε αυτούς τους δύο κόσμους είναι ότι η SnowLand έχει ως κύριο στοιχείο στις πίστες της τα χιόνια, ενώ η WaterLand έχει ως βασικό στοιχείο της τη θάλλασα η οποία έχει εισέλθει πάνω από τα χιόνια και δυσκολεύει την ορατότητα και τον προσανατολισμό του ήρωα.

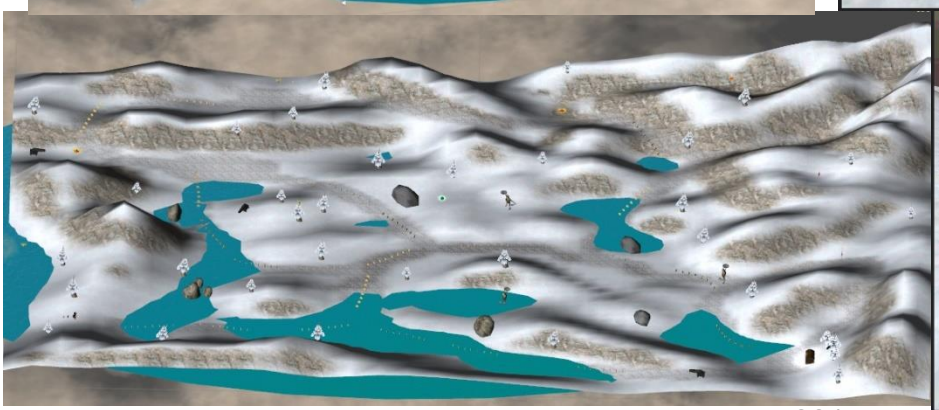
Πέραν αυτού η WaterLand έχει ένα μεγαλύτερο βαθμό δυσκολίας στην κάθε πίστα όσον αφορά τον τερματισμό και τους αντιπάλους του ήρωα. Οι αντίπαλοι εντοπίζουν τον ήρωα από μεγαλύτερη απόσταση, υπάρχουν λιγότερα αντικείμενα βοήθειας που μπορεί να συλλέξει ο ήρωας, και για να τερματίσει οποιαδήποτε πίστα πρέπει να έχει σκοτώσει πρώτα όλους του τους αντιπάλους.

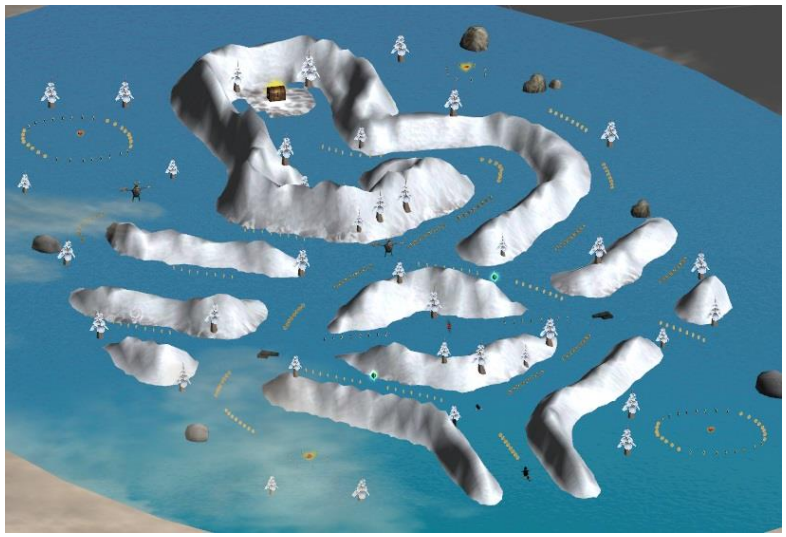


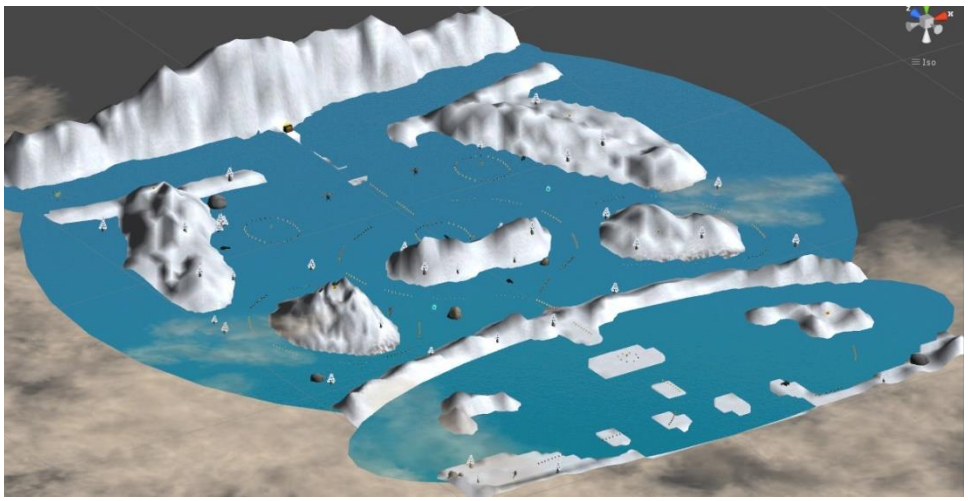








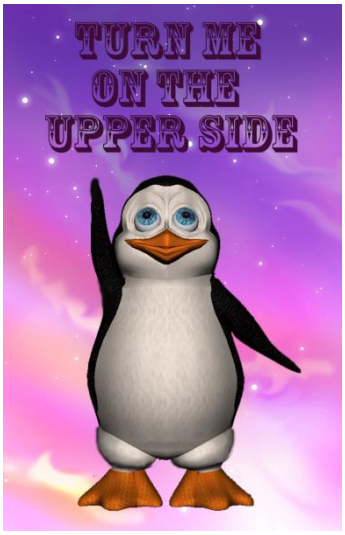




4.3.4 JumpLand

Σε αυτόν τον κόσμο ο πιγκουίνος μαθαίνει να πετάει για να μπορέσει να το σκάσει από τους βίκινκς που τον κυνηγάνε παντού. Το παιχνίδι εδώ είναι ειδικά σχεδιασμένο για κινητές συσκευές. Με τη βοήθεια του γυροσκοπίου ελέγχουμε την κλίση και τη θέση της συσκευής. Μπορούμε να παίξουμε μόνο κρατώντας τη συσκευή σε όρθια θέση με το κεντρικό κουμπί προς τα κάτω. Κρατώντας τη συσκευή σε οποιαδήποτε άλλη θέση (πχ σε πλάγια ή σε ανάποδη όρθια θέση) το παιχνίδι μπαίνει σε αναμονή μέχρι να περιστρέψουμε τη συσκευή στη σωστή θέση. Ανεξάρτητα με την περιστροφή της οθόνης, το παιχνίδι δεν περιστρέφεται και παραμένει σε όρθια θέση, ώστε να ξέρει ο χρήστης προς ποια κατεύθυνση να το γυρίσει.

Σε πρώτο στάδιο αντί για αναμονή του παιχνιδιού όταν ο χρήστης κρατούσε σε λάθος θέση τη συσκευή, είχαμε ένα μήνυμα που εμφανιζόταν σε όλη την έκταση της οθόνης, και ενημέρωνε τον χρήστη προς τα που έπρεπε να την περιστρέψει, αναλόγως τη θέση που την κρατούσε. Μόλις την τοποθετούσε στην σωστή θέση, το μήνυμα εξαφανιζόταν και ξεκινούσε το παιχνίδι. Οι χρήστες που δοκίμασαν το παιχνίδι, το θεώρησαν περιττό και προτιμούσαν να βλέπουν απευθείας την πίστα του παιχνδιού και όχι το μήνυμα περιστροφής, οπότε το αφαιρέσαμε.



Η κάμερα βρίσκεται μπροστά από τον ήρωα και αρκετά μακριά από

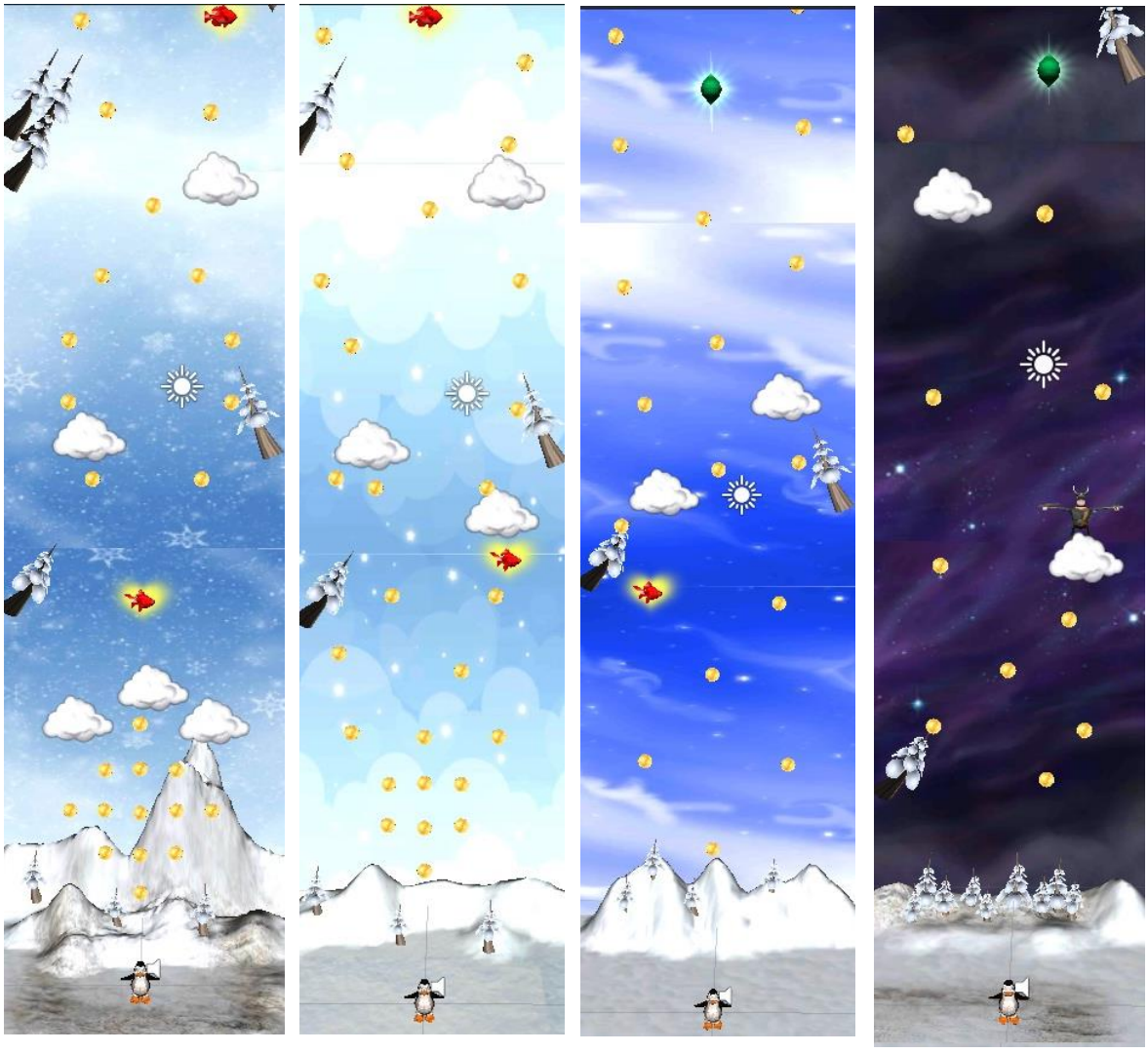
αυτόν και τον ακολουθεί με ομαλή κίνηση, μέσω φίλτρων, μόνο στον κάθετο άξονα. Ο χρήστης βλέπει τον ήρωα από το μπροστινό του προφίλ και ό,τι τον περιβάλλει.

Ο πιγκουίνος σε αυτό τον κόσμο περπατάει πλάγια, πηδάει, πετάει, πέφτει κάνοντας τούμπες στον αέρα και σκοτώνεται.

Πίστες

Οι πίστες εδώ είναι πολύ πιο απλές, καθώς ο πιγκουίνος κατά κύριο λόγο βρίσκεται στον αέρα. Σε κάθε πίστα υπάρχει ένα πολύ απλό και μικρό έδαφος, καθώς ο ήρωας δεν βρίσκεται πάνω σε αυτό μόνο μέχρι να ξεκινήσει το παιχνίδι, και από τον ουρανό, στον οποίο βρίσκονται τα αντικείμενα συλλογής (νομίσματα, ψαράκια, δυνάμεις), σύννεφα, δέντρα, και οι βίκινγκς που κρύβονται πάνω στα σύννεφα και στα δέντρα.

Ο ουρανός επεκτείνεται πάνω από το έδαφος όσο ο πιγκουίνος πετάει προς τα πάνω, μέχρι να φτάσει στο σημείο τερματισμού. Στις παρακάτω εικόνες φαίνεται μόνο το αρχικό κομμάτι της κάθε πίστας, καθώς δεν χωράει ολόκληρη.



Χειρισμός

Ο πιγκουίνος στέκεται ακίνητος στο κέντρο της οθόνης και κοιτάει τον χρήστη. Μπορεί να κινηθεί μόνο λίγα μέτρα δεξιά και αριστερά ώστε να μην βγαίνει έξω από τα πλαίσια της οθόνης, καθώς η κάμερα δεν τον ακολουθεί στην οριζόντια κίνηση. Η κίνησή του στον οριζόντιο άξονα ελέγχεται από το επιταχυνσιόμετρο και από ένα φίλτρο που τοποθετήσαμε ώστε να έχουμε πιο στρογγυλά νούμερα για πιο ομαλή κίνηση. Δίνοντας κλίση στη συσκευή μας δεξιά και αριστερά ο ήρωας κινείται αντίστοιχα.

Το παιχνίδι ξεκινάει μόλις ο χρήστης πατήσει μια φορά την οθόνη σε οποιοδήποτε σημείο, όπου ο ήρωας απογειώνεται πηδώντας πολύ ψηλά. Για να παραμείνει ο πιγκουίνος στον αέρα πρέπει να μαζεύει ότι αντικείμενα βρει στο δρόμο του, όπως νομίσματα, ψαράκια και μαγικά σμαράγδια. Τα αντικείμενα αυτά του δίνουν δύναμη για να πετάξει πιο ψηλά. Χωρίς αυτά δεν μπορεί να πετάξει, αλλά πέφτει και σκοτώνεται. Πέφτοντας η κάμερα δεν τον ακολουθεί, και καθώς βγαίνει από τα πλαίσια της οθόνης έχει μία ευκαιρία να μαζέψει άλλο

ένα αντικείμενο ώστε να μπορέσει να ξαναπετάξει. Αν δεν τα καταφέρει σκοτώνεται. Όσο είναι στον αέρα εξακολουθεί να κινείται δεξιά και αριστερά με την κλίση που δίνουμε στη συσκευή.

Πόντοι

Το κάθε νόμισμα που συλλέγει του προσθέτει ένα βαθμό στην τελική βαθμολογία, και από τα ψαράκια μπόνους εκατό βαθμούς το κίτρινο ψάρι, διακόσιους το μωβ και τριακόσιους το κόκκινο.

Ετικέτες

Στο πάνω μέρος της οθόνης αριστερά αναγράφεται ο αριθμός των νομισμάτων που έχει συλλέξει. Δεξιά φαίνονται τα τρία ψαράκια με γκρι χρώμα, και όταν συλλέξει κάποιο από αυτά παίρνει έντονο χρώμα.

Αντίπαλοι

Σε πίστες δυσκολότερου επιπέδου, ανάμεσα στα σύννεφα και τα δέντρα, κρύβονται οι βίκινγκς που ανακάλυψαν ότι ο πιγκουίνος προσπαθεί να ξεφύγει πετώντας. Αν τους ακουμπήσει με οποιοδήποτε τρόπο σκοτώνεται αμέσως και πέφτει προς τα κάτω.

Pick Ups

Αν όμως έχει πάρει το μαγικό σμαράγδι γίνεται διπλάσιος σε μέγεθος και είναι αθάνατος για οχτώ δευτερόλεπτα. Επίσης αυξάνεται και η ταχύτητά του.

Τέλος παιχνιδιού

Όταν πέσει και σκοτωθεί εμφανίζεται μια καρτέλα τέλος παιχνιδιού με τρία κουμπιά, ένα για να επιστρέψει στο μενού με τις πίστες αυτού του κόσμου, ένα για να επαναλάβει την πίστα και ένα για να προχωρήσει στην επόμενη αν και μόνο αν έχει τερματίσει την τωρινή επιτυχώς κάποια προηγούμενη φορά.

Στόχος

Για να κερδίσει πρέπει να φτάσει στα μεγάλα σύννεφα που βρίσκονται στο υψηλότερο σημείο της πίστας.

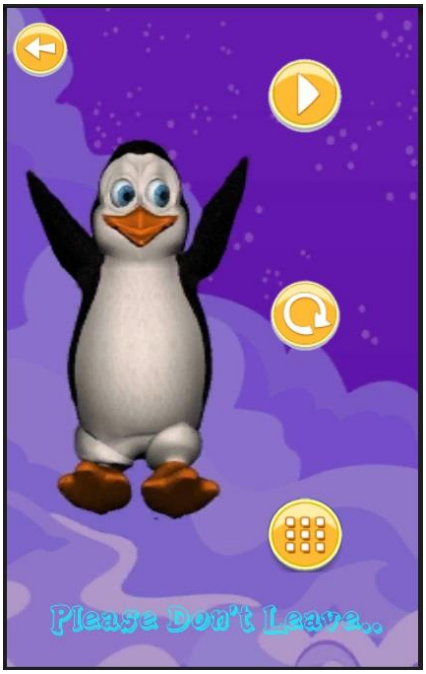
Νίκη

Τότε εμφανίζεται η καρτέλα νίκης στην οποία αναγράφεται η τελική βαθμολογία. Κάθε φορά αποθηκεύεται η μεγαλύτερη βαθμολογία που πετυχαίνει ο χρήστης ώστε να χρησιμοποιείται ως μέτρο σύγκρισης με το εκάστοτε σκορ. Όποτε κάνει νέο ρεκόρ εμφανίζεται η μεγαλύτερη βαθμολογία με γαλάζια γράμματα, αναγράφοντας ότι πέτυχε μεγαλύτερο σκορ, ενώ αν δεν είναι η μεγαλύτερή του βαθμολογία εμφανίζεται με κίτρινα γράμματα απλά το

σκορ. Η βαθμολογία προκύπτει από τα νομίσματα και τα ψάρια που συνέλεξε. Από κάτω φαίνονται τα νομίσματα που πήρε και με έντονα χρώματα τα ψαράκια μπόνους που συνέλεξε ενώ εκείνα τα οποία δεν κατάφερε να μαζέψει είναι με γκρι χρώμα. Στο κάτω μέρος της οθόνης είναι τρία κουμπιά, ένα για την επιστροφή στο μενού με τις πίστες αυτού του κόσμου, ένα για την επανάληψη της πίστας και ένα για την έναρξη της επόμενης.

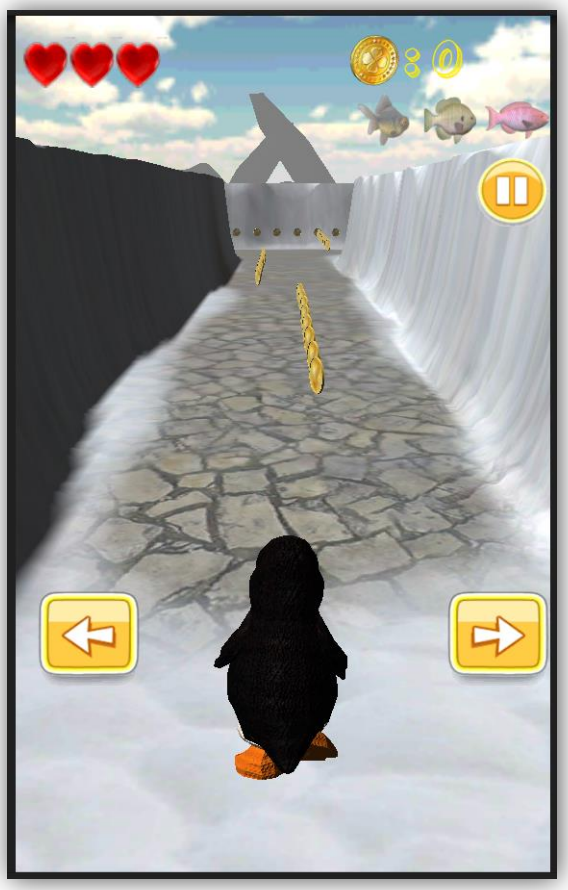
Παύση

Για τη διακοπή του παιχνιδιού διαθέτουμε τρεις διαφορετικούς τρόπους. Μπορούμε να επιλέξουμε το κουμπί παύσης που βρίσκεται στο πάνω μέρος της οθόνης δεξιά, είτε μπορούμε να ακουμπήσουμε δύο δάχτυλα ταυτόχρονα πάνω στην οθόνη σε οποιοδήποτε σημείο. Και στις δύο περιπτώσεις εμφανίζεται το μενού παύσης με τέσσερα κουμπιά, ένα για να επιστρέψουμε πίσω από εκεί που σταματήσαμε, ένα για να συνεχίσουμε το παιχνίδι από εκεί που το αφήσαμε, ένα για να επαναλάβουμε την πίστα από την αρχή και ένα για να πάμε στο μενού με τις πίστες αυτού του κόσμου.



Επίσης μπορούμε να αφήσουμε τη συσκευή κάτω ή να τις δώσουμε τέτοια κλίση σαν να την είχαμε αφήσει σε μια επιφάνεια και αμέσως θα διακοπεί το παιχνίδι. Για να συνεχιστεί αρκεί απλά να ξαναπάρουμε τη συσκευή στα χέρια μας.

4.3.5 RunLand

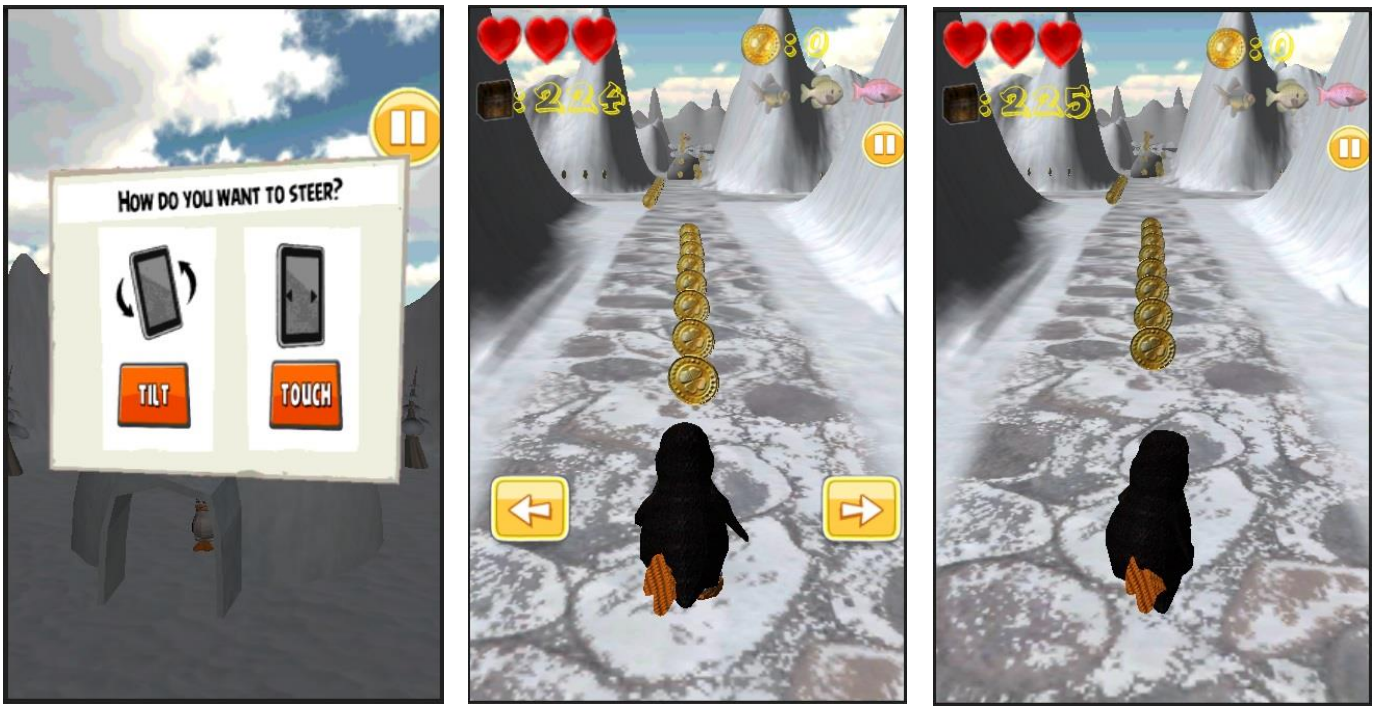


Σε αυτόν τον κόσμο ο πιγκουίνος τρέχει όλη την ώρα για να μην τον πιάσει ο βίκινγκ που είναι από πίσω του και τον κυνηγάει. Το παιχνίδι και σε αυτήν την περίπτωση είναι σχεδιασμένο μόνο για κινητές συσκευές. Για να ξεκινήσει πρέπει να έχουμε τη συσκευή σε όρθια θέση με το κεντρικό κουμπί κάτω. Αν κρατάμε τη συσκευή σε οποιαδήποτε άλλη (πλάγια ή ανάποδη όρθια) θέση όταν ξεκινάει το παιχνίδι δεν μπορούμε να παίξουμε καθώς μπαίνει σε αναμονή έως ότου περιστρέψουμε τη συσκευή σε όρθια θέση.

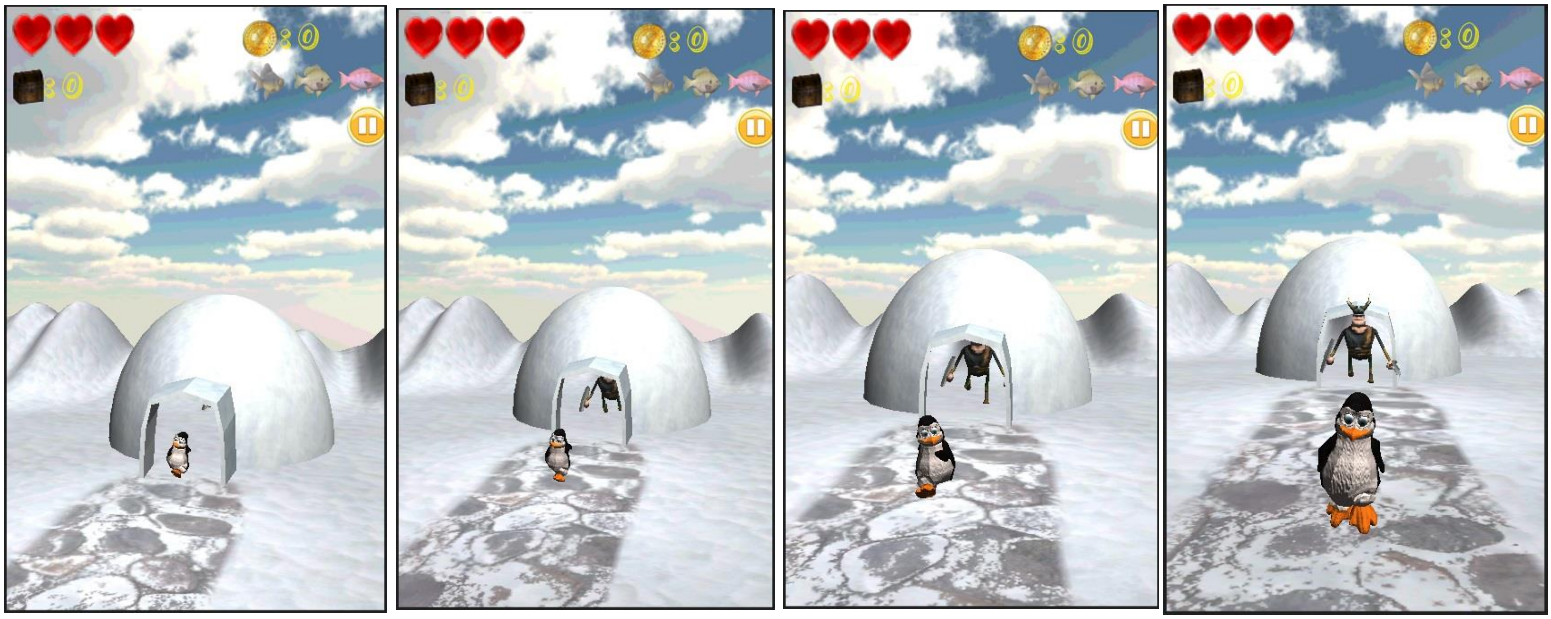
Ο πιγκουίνος σε αυτόν τον κόσμο μπορεί να τρέχει, να πηδάει, να πηδάει διπλά, να σκύβει και να κάνει τούμπες στο πάτωμα, να στρίβει απότομα κατά 90°, να πανηγυρίζει, να πέφτει, να χτυπάει και να σκοτώνεται.

Ξεκινώντας το παιχνίδι εμφανίζεται ένα μενού με το οποίο ρωτάμε το χρήστη με ποιον τρόπο επιθυμεί να ελέγχει την κίνηση του ήρωα μετακινώντας τον δεξιά και αριστερά. Η κίνηση ελέγχεται είτε με το επιταχυνσιόμετρο είτε με δύο κουμπιά αφής. Επιλέγοντας το επιταχυνσιόμετρο ο ήρωας κινείται δεξιά και αριστερά αναλόγως την κλίση που δίνουμε στη

συσκευή αντίστοιχα. Επιλέγοντας τα κουμπιά εμφανίζονται δύο βέλη δεξιά και αριστερά στην οθόνη, όσο τα πιέζουμε τόσο μετακινείται ο ήρωας προς την αντίστοιχη κατεύθυνση.



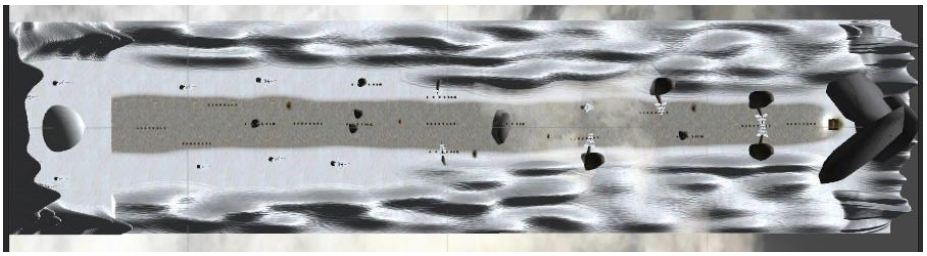
Αφού επιλέξουμε τον τρόπο ελέγχου της κίνησης, ξεκινάει μία ταινία μικρού μήκους μερικών δευτερολέπτων η οποία μας εισάγει στην υπόθεση του παιχνιδιού. Ο πιγκουίνος βγαίνει τρέχοντας έξω από το ιγκλού του και από πίσω του βρίσκεται ο βίκινγκ ο οποίος τον κυνηγάει. Αρχικά η κάμερα βρίσκεται ψηλά και μπροστά από το ιγκλού, καθώς βγαίνει ο πιγκουίνος από το ιγκλού η κάμερα κατευθύνεται προς το μέρος του κοιτώντας τον. Για να μπορέσει ο πιγκουίνος να ξεφύγει από τον βίκινγκ κατευθύνεται προς το λαβύρινθο που

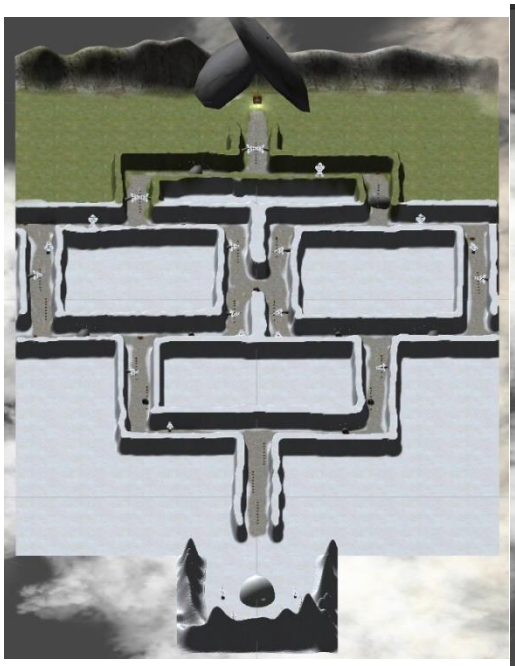


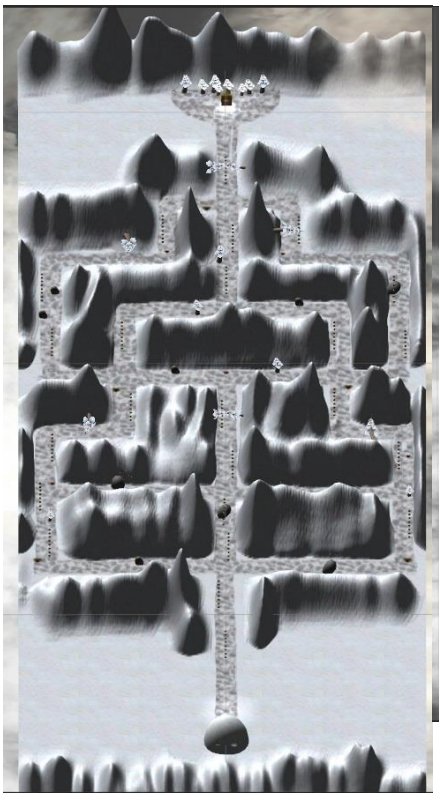
βίσκεται μπροστά του, σε εκείνο το σημείο αλλάζει η κάμερα η οποία τοποθετείται πίσω από τον πιγκουίνο πάνω από το κεφάλι του και τότε ξεκινάει το παιχνίδι.

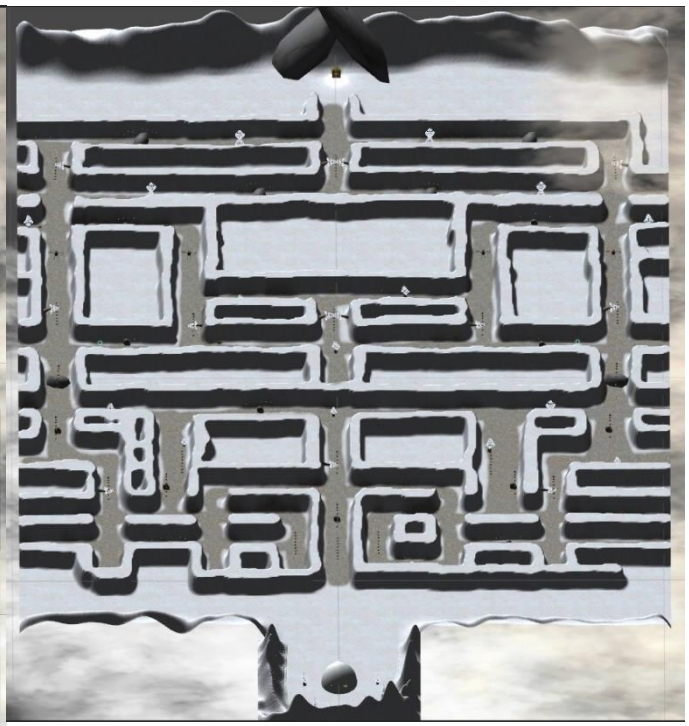
Πίστες

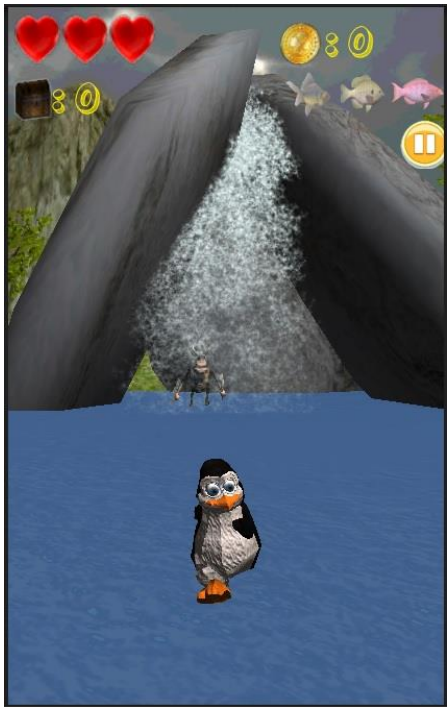
Η κάθε πίστα αποτελείται από έναν λαβύρινθο, στον οποίο ο ήρωας πρέπει να βρει την έξοδο για το θησαυρό, αποφεύγοντας τα εμπόδια που υπάρχουν στο δρόμο του, τον βίκινγκ που τον κυνηγάει, και τα αδιέξοδα που οδηγούν στο κενό. Ξεκινάει να τρέχει από το ιγκλού σε ένα περιβάλλον με χιόνια. Όσο προχωράει τις πίστες και απομακρύνεται από το αρχικό του σημείο, το περιβάλλον αλλάζει, και σιγά σιγά πηγαίνει σε χλωρό περιβάλλον με γρασίδι, καταπράσινα δέντρα και νερά, στο οποίο ξεκινάει να τρέχει από μια σπηλιά ή από έναν καταράκτη. Στο τέλος όμως, επιστρέφει στο κανονικό του περιβάλλον. Οι πίστες περιβάλλονται από σύννεφα στα οποία βρίσκεται ο πιγκουίνος αν πέσει από το έδαφος και σκοτωθεί. Στο κενό μπορεί να βρεθεί μόνο από τα αδιέξοδα και όχι όταν βρίσκεται στο νερό, καθώς τότε δεν του επιτρέπεται να προχωρήσει έξω από τα όρια του εδάφους ως βοήθεια λόγω περιορισμένης ορατότητας. Ο βαθμός δυσκολίας αυξάνεται με την πίστα.

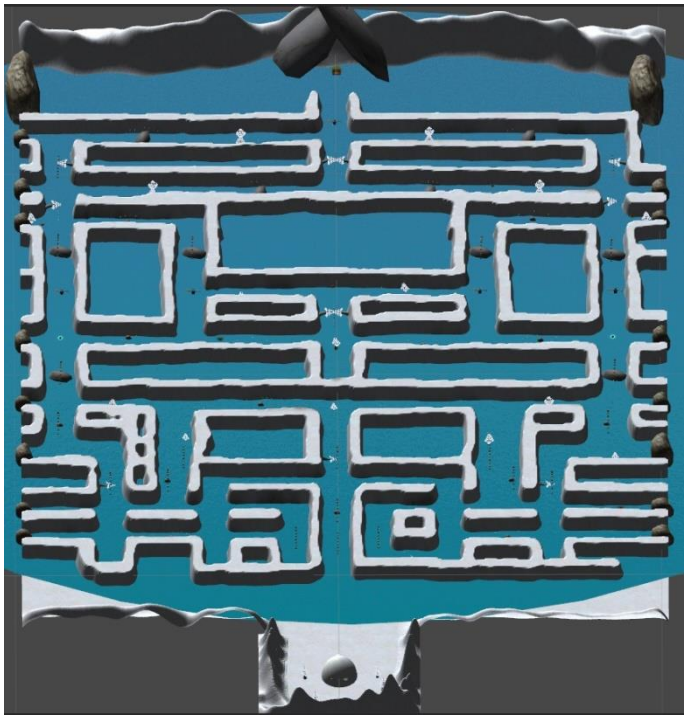


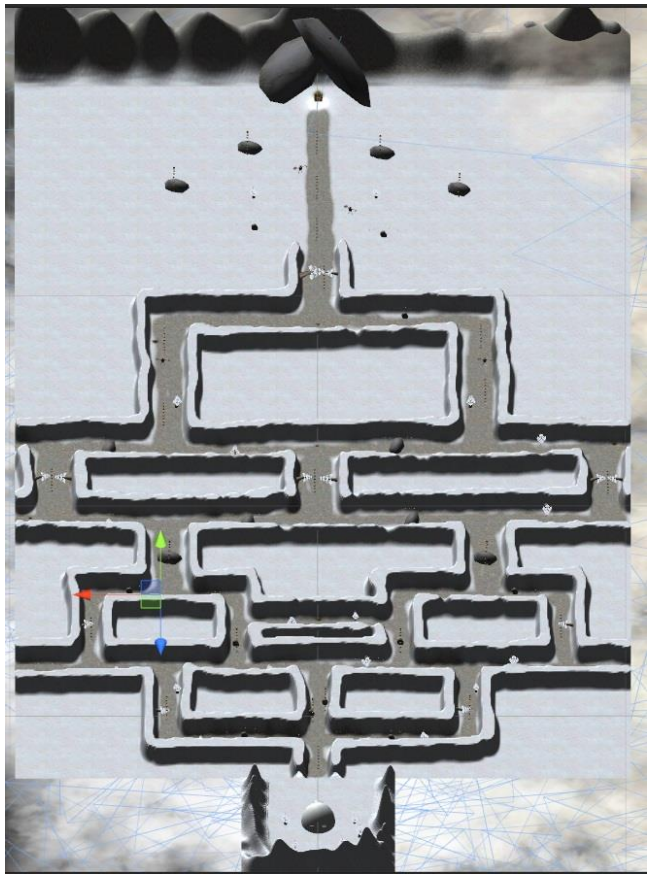


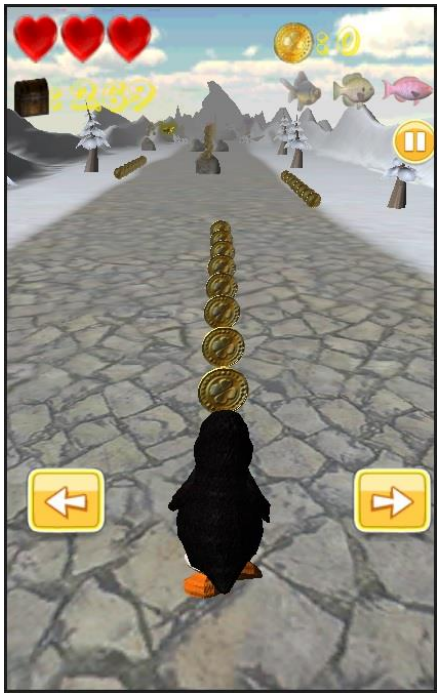
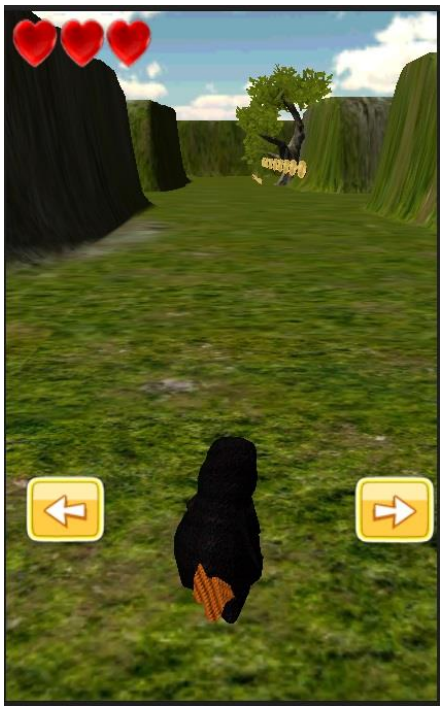











Αντίπαλοι

Εκτός από τον βίκινγκ που βρίσκεται από πίσω του, ο πιγκουίνος πρέπει να προσέχει να αποφεύγει τα εμπόδια που βρίσκονται στο δρόμο του. Μπορεί να τα αποφεύγει πηγαίνοντας δεξιά και αριστερά, να πηδάει πάνω από τους καφέ βράχους ή τα πεσμένα δέντρα, να πηδάει διπλά πάνω από τους μεγαλύτερους γκρι βράχους, και να σκύβει κάνοντας τούμπα κάτω από τα δέντρα που γέρνουν από πάνω του.

Στον βίκινγκ έχουμε προσθέσει τεχνητή νοημοσύνη ώστε να μπορεί να εντοπίζει τον πιγκουίνο μέσα στο λαβύρινθο και να τον κυνηγάει ακολουθώντας την πιο σύντομη διαδρομή ανεξαρτήτως τη διαδρομή που ακολούθησε ο ήρωας, περπατώντας μόνο πάνω στο επιτρεπτό μονοπάτι. Ότι εμπόδιο συναντήσει μπροστά του ο βίκινγκ το καταστρέφει για να περάσει, επομένως αν ξαναπεράσει από εκεί ο ήρωας δε θα το ξανασυναντήσει.

Ο βίκινγκ τρέχει πιο αργά από τον πιγκουίνο για αυτό δεν μπορεί να τον πιάσει, παρά μόνο αν ο πιγκουίνος καθυστερεί σε κάποιο σημείο και δεν προχωράει, όπως παραδείγματος χάρη αργεί να στρίψει ή έχει κολλήσει σε κάποιο εμπόδιο. Κάθε φορά που ο ήρωας σκοντάφτει σε κάποιο εμπόδιο και δεν το αποφύγει ο βίκινκγ τον προφτάνει. Την πρώτη φορά που θα πέσει σε ένα εμπόδιο θα χάσει μια ζωή και ο βίκινγκ θα εμφανιστεί στην κάμερα από πίσω του, αλλά αν ο πιγκουίνος συνεχίσει να προχωράει δεν θα τον πιάσει. Την δεύτερη φορα θα έρθει ακόμα πιο κοντά του, και την τρίτη φορά θα τον πιάσει και ο πιγκουίνος θα πεθάνει.

Σε πίστες δυσκολότερου επιπέδου υπάρχουν και άλλοι βίκινγκ μέσα στο λαβύρινθο οι οποίοι περιμένουν να περάσει από εκεί ο πιγκουίνος. Μόλις τον εντοπίσουν τρέχουν κατά πάνω του. Ο μόνος τρόπος για να τους αποφύγει ο ήρωάς μας, είναι να πηδήξει από πάνω

τους. Ακόμα και να πατήσει πάνω στο κεφάλι τους δεν σκοτώνεται, αν όμως τον ακουμπήσουν από μπροστά ο πιγκουίνος θα πεθάνει. Όταν απομακρυνθεί από αυτούς σταματάνε να τον κυνηγάνε εκτός από τον αρχικό βίκινγκ ο οποίος εξακολουθεί να βρίσκεται από πίσω του.

Pick Ups

Ως βοήθεια υπάρχουν και εδώ μαγικά σμαράγδια τα οποία δίνουν στον ήρωα μια φωτεινή ασπίδα γύρω από αυτόν που τον κάνει αθάνατο ως προς τα εμπόδια και τους βίκινγκς. Μόνο αν πέσει στο κενό πεθαίνει ακόμα και αν έχει την ασπίδα, της οποίας η διάρκεια είναι οχτώ δευτερόλεπτα.

Ζωές

Σε κάθε πίστα ο ήρωας έχει τρεις ζωές, όμως δεν έχει πιγκουινάκια να τον ακολουθούν ώστε να του δίνουν περισσότερες ευκαιρίες πριν χάσει μια ζωή. Όταν πέφτει πάνω σε κάποιο εμπόδιο κάνει μια μικρή παύση καθώς τραυματίζεται, χάνει μια ζωή, αφαιρείται το αντικείμενο με το οποίο συγκρούστηκε και στη συνέχεια συνεχίζει να παίζει από το ίδιο σημείο. Αν πέσει στο κενό ή τον πιάσει ο βίκινγκ χάνει όλες του τις ζωές και το παιχνίδι τελειώνει, ο πιγκουίνος πεθαίνει, εμφανίζεται μήνυμα τέλους παιχνιδιού, η κάμερα τον πλησιάζει από το πλάι ώστε να φαίνεται καλύτερα και εμφανίζεται η καρτέλα τέλους παιχνιδιού.

Τέλος παιχνιδιού

Όταν πεθάνει εμφανίζεται η καρτέλα τέλος παιχνιδιού με τρία κουμπιά, ένα για την επιστροφή στο μενού με τις πίστες αυτού του κόσμου, ένα για την επανάληψη της πίστας και ένα για την είσοδο στην επόμενη αν και μόνο αν έχει τερματίσει επιτυχώς την τωρινή κάποια άλλη φορά.

Πόντοι

Στο δρόμο του υπάρχουν παντού νομίσματα τα οποία του ανεβάζουν το σκορ κατά ένα βαθμό το κάθε νομισμα. Στα σημεία που υπάρχουν εμπόδια τα νομίσματα έχουν πάρει τέτοια θέση ώστε να του δείχνουν προς τα που πρέπει να πάει και να καταλάβει αν χρειάζεται να πηδήξει ή να σκύψει.

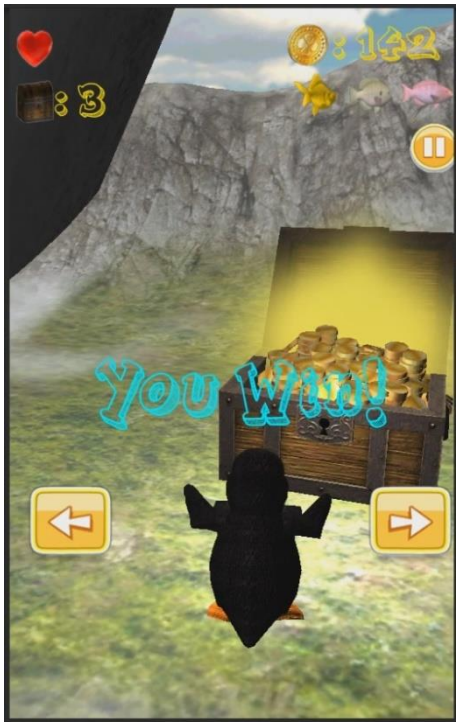
Τα ψαράκια μπόνους είναι κρυμμένα μέσα στο λαβύρινθο και δίνουν εκατό βαθμός το κίτρινο, διακόσιους το μωβ, και τριακόσιους το κόκκινο στην τελική βαθμολογία.

Ετικέτες

Στο πάνω μέρος της οθόνης δεξιά αναγράφεται ο αριθμός των νομισμάτων που έχει συλλέξει. Κάτω από αυτά, αρχικά φαίνονται τα τρία ψαράκια με γκρι χρώμα, και όταν συλλέξει κάποιο από αυτά παίρνει έντονο χρώμα.

Στο αριστερό μέρος υπάρχουν τρεις κόκκινες καρδούλες οι οποίες φανερώνουν τον αριθμό των ζωών του ήρωα. Με κάθε ζωή που χάνει αφαιρείται και μία καρδούλα. Όταν χάσει όλες τις ζωές παγώνουν όλες οι καρδιες και γίνονται γαλάζιες.

Κάτω από τις καρδούλες αναγράφεται η απόσταση που απομένει να διανύσει ο πιγκουίνος μέχρι να φτάσει στο θησαυρό. Με αυτόν τον τρόπο μπορεί να καταλάβει ο χρήστης πότε έχει πάρει το λάθος δρόμο στο λαβύρινθο, καθώς όταν απομακρύνεται από αυτόν η απόσταση ανάμεσά τους μεγαλώνει αντί να μικραίνει.



Στόχος

Για να κερδίσει ο πιγκουίνος πρέπει να βρει την έξοδο του λαβυρίνθου και να φτάσει στον θησαυρό. Μόλις τον πλησιάσει και ανοίξει το σεντούκι ο βίκινγκ σταματάει να τον κυνηγάει, ο πιγκουίνος πανηγυρίζει, η μουσική αλλάζει και το παιχνίδι τελειώνει.

Νίκη

Στη συνέχεια εμφανίζεται η τελική βαθμολογία και αποθηκεύεται πάντα η μεγαλύτερη ώστε να αναγράφεται με

γαλάζια γράμματα πότε ο ήρωας πέτυχε νέο ρεκόρ, ενώ με κίτρινα γράμματα όταν δεν έκανε καινούργιο σκορ. Η βαθμολογία προκύπτει από τα νομίσματα και τα ψαράκια που συνέλεξε τα οποία φαίνονται αναλυτικά από κάτω, όπου εμφανίζεται ο αριθμός των νομισμάτων και τα τρία ψαράκια από τα οποία έχουν έντονο χρώμα μόνο όσα κατάφερε να συλλέξει.

Χειρισμός

Εκτός από τον χειρισμό με το επιταχυνσιόμετρο ή τα κουμπιά αφής που ελέγχουν πόσο θα κινηθεί ο πιγκουίνος δεξιά και αριστερά έχουμε και άλλες δυνατότητες.

- Σέρνοντας το δάχτυλο στην οθόνη προς τα πάνω ο πιγκουίνος πηδάει, για να αποφύγει τα μικρά εμπόδια που βρίσκονται στο έδαφος.
- Για να εκτελέσει διπλό πήδημα αποφεύγοντας μεγαλύτερα εμπόδια στο έδαφος, όσο βρίσκεται στον αέρα ξανά σύρουμε το δάχτυλό μας προς τα πάνω.
- Σέρνοντας το δάχτυλο στην οθόνη προς τα κάτω ο πιγκουίνος σκύβει και να κάνει τούμπα, για να αποφύγει δέντρα που γέρνουν από πάνω του.
- Σέρνοντας το δάχτυλο πάνω στην οθόνη προς τα δεξιά ή προς τα αριστερά ο πιγκουίνος στρίβει αντίστοιχα κατά εννενήντα μοίρες (90°) για να μπορέσει να κινηθεί μέσα στο λαβύρινθο.

Παύση

Υπάρχουν πάλι τρεις τρόποι για να διακόψουμε το παιχνίδι. Πιέζοντας το κουμπί αφής που βρίσκεται στο πάνω μέρος της οθόνης δεξιά, ή τοποθετώντας δύο δάχτυλα ταυτόχρονα πάνω στην οθόνη. Και στις δύο περιπτώσεις σταματάει το παιχνίδι ακριβώς στο σημείο που διαλέξαμε και εμφανίζεται ένα μενού παύσης με τέσσερα κουμπιά αφής. Ένα κουμπί για την επιστροφή στο παιχνίδι, ένα για την συνέχιση του παιχνιδιού, ένα για την επανάληψή του, και ένα για την επιστροφή στο μενού με τις πίστες αυτού του κόσμου.

Άλλος ένας τρόπος να σταματήσει το παιχνίδι είναι να αφήσουμε κάτω τη συσκευή, ή να τις δώσουμε τέτοια κλίση σαν να την είχαμε αφήσει. Μόλις την σηκώσουμε πάλι το παιχνίδι συνεχίζει να παίζει στο σημείο που το αφήσαμε.

ΚΕΦΑΛΑΙΟ 5

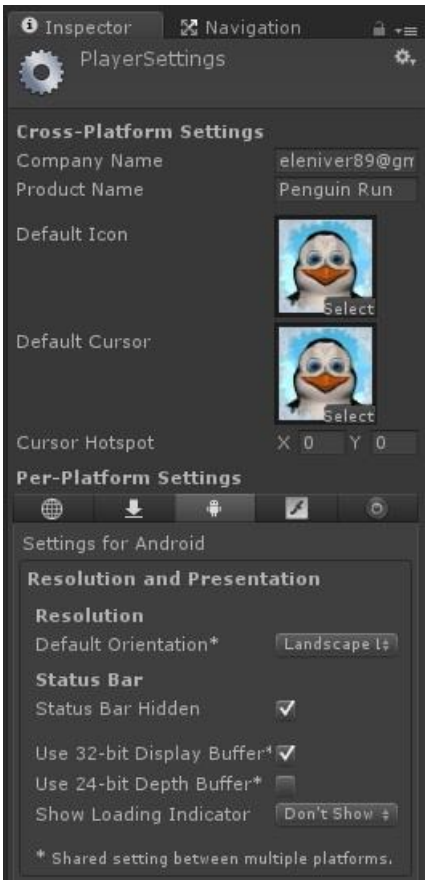
ΥΛΟΠΟΙΗΣΗ ΠΑΙΧΝΙΔΙΟΥ

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο αναλύουμε το προγραμματιστικό και τεχνικό κομμάτι του παιχνιδιού με το οποίο υλοποιήθηκε και έφερε ζωή σε όλα τα παραπάνω μέρη που αναλύσαμε.

Σε κάθε αντικείμενο της κάθε πίστας, έχουν προσαρτυθεί ένα ή περισσότερα συστατικά (components) της Unity, ώστε να αποκτήσει ιδιότητες και χαρακτηριστικά για να μπορέσει να ανταποκριθεί στις απαιτήσεις του παιχνιδιού και να αλληλεπιδράσει με τα γεγονότα.

Επίσης, σε κάθε αντικείμενο είναι συνδεδεμένα με ένα η περισσότερα αρχεία κώδικα, τα οποία ελέγχουν τα συστατικά του αντικειμένου και δημιουργούν καινούργια, ώστε να μπορέσουμε να ελέξγουμε πλήρως τη συμπεριφορά του με τον τρόπο που θέλουμε. Ο κώδικας είναι γραμμένος σε JavaScript και μεταγλωτίστηκε στο MonoDevelop της Unity.



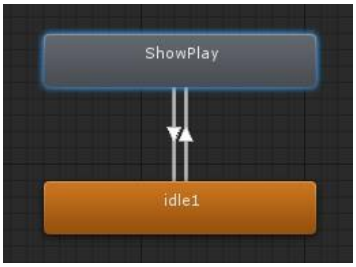
5.2 Οργάνωση της εφαρμογής

Στις Player Settings ορίσαμε διάφορες παραμέτρους για την πλατφόρμα Android και iOS για την τελευτική μορφή του παιχνιδιού. Ρυθμίσαμε το εικονίδιο της εφαρμογής του παιχνιδιού, και την ανάλυση και παρουσίαση της εικόνας που εμφανίζεται κατά τη διάρκεια της φόρτωσης του παιχνιδιού από τη συσκευή.

Η εικόνα αυτή εμφανίζεται σε θέση Landscape Left, δηλαδή σε όρθια θέση της συσκευής με το κεντρικό κουμπί στη δεξιά μεριά, και παραμένει σε αυτή τη θέση ανεξάρτητα την περιστροφή της συσκευής. Βρίσκεται στο κέντρο της οθόνης, καλύπτοντάς την ολόκληρη. Επίσης εμφανίζεται ο χρόνος φόρτωσης της εφαρμογής.

5.3 Οργάνωση του κεντρικού μενού

Το κεντρικό μενού περιλαμβάνει ένα απλό πλαίσιο για το φόντο στο πίσω μέρος στο οποίο έχουμε εισάγει την υφή της εικόνας. Μπροστά από αυτό βρίσκονται ο πιγκουίνος, ένας βίκινγκ και ένα κανόνι, για να δώσουμε στο χρήστη τα βασικά στοιχεία του παιχνιδιού, τα οποία κινούνται με ένα απλό animation.



Ο πιγκουίνος στέκεται, κοιτάει δεξιά και αριστερά όσο περιμένει τον χρήστη να ξεκινήσει το παιχνίδι, και όσο εκείνος καθυστερεί του δείχνει το κουμπί “Play” για να το επιλέξει. Οι κινήσεις του αποδίδονται μέσω του συστατικού Animator και του Controller που ορίσαμε, ο οποίος μεταβαίνει μεταξύ των καταστάσεων αναμονής και επίδειξης του κουμπιού με την πάροδο του χρόνου.

Ο βίκινγκ περπατάει προς το σημείο του πιγκουίνου, χωρίς να προχωράει. Η κίνηση αυτή αποδίδεται με το συστατικό Animation στο οποίο έχουμε εισάγει την συγκεκριμένη κίνηση. Το κανόνι δείχνει για πέντε δευτερόλεπτα το έδαφος και για πέντε δευτερόλεπτα σημαδεύει τον πιγκουίνο, μέσω ενός κώδικα με τον οποίο ελέγχεται η περιστροφή και η κατεύθυνσή του.

Η γραφική διεπαφή του μενού και οι λειτουργίες των κουμπιών ρυθμίζονται από το **MenuControllerMobile.js**. Με την συνάρτηση **OnGUI()** ρυθμίζουμε το χρώμα, το μέγεθος και τη θέση της γραμματοσειράς, των ετικέτων και των κουμπιών χρησιμοποιώντας **GUIStyle**. Με την κλάση **GUI.Label**, η οποία χρησιμοποιείται για τη δημιουργία κειμένων ή ετικετών στην οθόνη, δημιουργήσαμε τον τίτλο στο πάνω μέρος της οθόνης με μεγάλα κίτρινα γράμματα. Με την κλάση **GUI.Button** δημιουργήσαμε στο κέντρο το κυρίως κουμπί με ένα κίτρινο ημιδιάφανο χρώμα, στο οποίο αναγράφεται το «Play» με άσπρα γράμματα, τα οποία γίνονται κίτρινα όταν επιλεχθεί το κουμπί.

Μόλις ο χρήστης επιλέξει το Play, φορτώνεται η επόμενη πίστα που περιλαμβάνει το μενού των κόσμων, με την κλάση **Application.LoadLevel()** η οποία φορτώνει τα επίπεδα με βάση το όνομα ή τον αριθμό τους. Αν ο χρήστης έχει εισέλθει στο κυρίως μενού από το μενού των κόσμων, αρχικοποιούμε οποιαδήποτε κατάσταση υπήρχε στο μενού των κόσμων, ώστε να έχουμε πρώτο πλάνο τον κόσμο SnowLand όταν επιστρέψουμε.

Επίσης δημιουργήσαμε και άλλα τρία μικρότερα κουμπιά κάτω από το κεντρικό, για την πρόσβαση στις οδηγίες του παιχνιδιού, την ενεργοποίηση ή απενεργοποίηση του ήχου, και τον τερματισμό του παιχνιδιού. Με την επιλογή κάθε κουμπιού αναπαράγεται ένας μικρός ήχος με την **audio.PlayOneShot()**, ο οποίος παράγεται από την κύρια κάμερα στην οποία έχουμε συνδέσει το συστατικό Audio Source.

Αν ο χρήστης επιλέξει να απενεργοποιήσει τον ήχο, αλλάζει το εικονίδιο του κουμπιού ώστε να φαίνεται ότι είναι απενεργοποιημένος. Η επιλογή του χρήστη πρέπει να αποθηκευτεί και να είναι προσβάσιμη από ολές τις κλάσεις και όλες τις σκηνές του παιχνιδιού. Για αυτό χρησιμοποιούμε την **PlayerPrefs** η οποία αποθηκεύει και μας δίνει πρόσβαση στις προτιμήσεις του χρήστη μεταξύ των σκηνών του παιχνιδιού. Η **PlayerPrefs.SetInt()** ορίζει την τιμή προτίμησης που προσδιορίζεται από το κλειδί, και η **PlayerPrefs.GetInt()** επιστρέφει την τιμή που ανιστοιχεί στο κλειδί προτίμησης αν υπάρχει.

```
if(bShowButton)          //label for sound on
{
        if (GUI.Button (Rect ( widthBut3, heightBut2, totalWidth2, totalHeight2 ), soundButton,"")) {
                gameObject.Find("Main Camera").audio.Pause();          //stop the sound in the menu
                bShowButton = false;                                   //change the label sound
                PlayerPrefs.SetInt("Sound",0);                         //no music for the game
        }
}
else if(bShowButton2)  //label for sound off
{
      if (GUI.Button (Rect (widthBut3, heightBut2, totalWidth2, totalHeight2 ), soundOffButton,"")) {
                audio.PlayOneShot(PressedButtonSound);          //sound when the button is pressed
                gameObject.Find("Main Camera").audio.Play();           //start the sound in the menu
                bShowButton2 = false;                                  //change the label sound
                PlayerPrefs.SetInt("Sound",1);                         //music on
        }
}
```

Έτσι ελέγχουμε σε κάθε σκηνή αν ο χρήστης έχει ενεργοποιημένο τον ήχο, για να επιτρέψουμε την αναπαραγωγή του με την **audio.Play()** η οποία παίζει το κλιπ, διαφορετικά απενεργοποιούμε τον ήχο με την **audio.Stop()** η οποία σταματάει το κλιπ.

```
if(PlayerPrefs.GetInt("Sound")==1)              //sound is on from the main menu
{
        audio.Play();
}
else if(PlayerPrefs.GetInt("Sound")==0)         //sound is off from the main menu
{
        audio.Stop();
}
```

Το κυρίως μενού είναι σχεδιασμένο για την όρθια θέση της συσκευής με το κεντρικό κουμπί στη δεξιά ή αριστερή μεριά, και περιστρέφεται αναλόγως τη θέση με την οποία την κρατάμε. Ο έλεγχος της θέσης της συσκευής γίνεται μέσω του script **TiltControlMenu.js**.

Η κλάση **Input.deviceOrientation** αναγνωρίζει το φυσικό προσανατολισμό της συσκευής, όπως αναφέρεται από το λειτουργικό σύστημα.

Η κλάση **DeviceOrientation** περιγράφει το φυσική προσανατολισμό της συσκευής, όπως καθορίζεται από το λειτουργικό σύστημα. Εάν η συσκευή βρίσκεται μεταξύ διακριτών θέσεων, όπως όταν περιστρέφεται διαγωνίως, το σύστημα θα αναφέρει άγνωστο προσανατολισμό.

Η κλάση **Screen.orientation** καθορίζει το λογικό προσανατολισμό της οθόνης, και η **ScreenOrientation** περιγράφει τον προσανατολισμό της οθόνης.

Η συνάρτηση **Start()** καλείται στο καρέ που ενεργοποιείται ο κώδικας λίγο πριν από όλες τις άλλες συναρτήσεις και καλείται μόνο την πρώτη φορά. Σε αυτήν ελέγχουμε τον φυσικό προσανατολισμό της συσκευής τη στιγμή που εισαγόμαστε στο μενού, και περιστρέφουμε την οθόνη ανάλογα. Αν ο προσανατολισμός της συσκευής που δηλώνεται από την Input.deviceOrientation, είναι σε όρθια θέση με το κουμπί στο κάτω μέρος που δηλώνεται με την **DeviceOrientation.Portrait**, ή με το κουμπί στα δεξιά που δηλώνεται με την **DeviceOrientation.LandscapeLeft**, ή παράλληλα προς το έδαφος με την οθόνη προς τα πάνω που δηλώνεται με την **DeviceOrientation.FaceUp**, τότε η οθόνη περιστρέφεται στη θέση ScreenOrientation.LandscapeLeft την οποία καθορίζουμε με την Screen.orientation. Αν η συσκευή είναι σε όρθια θέση με το κουμπί στο πάνω μέρος που δηλώνεται με την **DeviceOrientation.PortraitUpsideDown**, ή με το κουμπί στα αριστερά που δηλώνεται με την **DeviceOrientation. LandscapeRight**, ή παράλληλα προς το έδαφος με την οθόνη προς τα κάτω που δηλώνεται με την **DeviceOrientation. FaceDown**, τότε η οθόνη περιστρέφεται στη θέση LandscapeRight που περιγράφεται από την ScreenOrientation, την οποία καθορίζουμε με την Screen.orientation.

Η **Update()** συνάρτηση καλείται κάθε καρέ, εάν ο κώδικας είναι ενεργοποιημένος. Η Update είναι η πιο συχνά χρησιμοποιούμενη συνάρτηση για την εφαρμογή κάθε είδους συμπεριφοράς παιχνιδιού. Σε αυτήν ελέγχουμε το φυσικό προσανατολισμό της συσκευής την κάθε στιγμή, σε περίπτωση που ο χρήστης την περιστρέψει ενώ βρίσκεται ήδη στο μενού, και περιστρέφουμε αντίστοιχα και την οθόνη όπως και στην περίπτωση της συνάρτησης Start().

```
#pragma strict
/*
        DeviceOrientation : Describes physical orientation of the device as determined by the OS.
```

```
        If device is physically situated between discrete positions, as when (for example) rotated
        diagonally, system will report Unknown orientation.
*/
static var widthScreen : float;
static var heightScreen : float;

function Start()
{
        //Portrait : The device is in portrait mode, with the device held upright and the home button
        //at the bottom.
        //LandscapeLeft : The device is in landscape mode, with the device held upright and the home
        //button on the right side.
        //FaceUp : The device is held parallel to the ground with the screen facing upwards.
        if (Input.deviceOrientation == DeviceOrientation.Portrait ||
                Input.deviceOrientation == DeviceOrientation.LandscapeLeft ||
                        Input.deviceOrientation == DeviceOrientation.FaceUp)
        {
                Screen.orientation = ScreenOrientation.LandscapeLeft;
        }
        //PortraitUpsideDown :The device is in portrait mode but upside down, with the device held
        //upright and the home button at the top.
        //LandscapeRight : The device is in landscape mode, with the device held upright and the
        //home button on the left side.
        //FaceDown : The device is held parallel to the ground with the screen facing downwards.
        else if (Input.deviceOrientation == DeviceOrientation.PortraitUpsideDown ||
                Input.deviceOrientation == DeviceOrientation.LandscapeRight ||
                Input.deviceOrientation == DeviceOrientation.FaceDown)
        {
                Screen.orientation = ScreenOrientation.LandscapeRight;
        }
}

function Update ()
{
        if (Input.deviceOrientation == DeviceOrientation.Portrait ||
                Input.deviceOrientation == DeviceOrientation.LandscapeLeft ||
                        Input.deviceOrientation == DeviceOrientation.FaceUp)
        {
                Screen.orientation = ScreenOrientation.LandscapeLeft;
        }
        else if (Input.deviceOrientation == DeviceOrientation.PortraitUpsideDown ||
                Input.deviceOrientation == DeviceOrientation.LandscapeRight ||
                Input.deviceOrientation == DeviceOrientation.FaceDown)
        {
```

```
                Screen.orientation = ScreenOrientation.LandscapeRight;
        }
}
```

5.4 Οργάνωση του μενού των κόσμων

Το μενού των κόσμων αποτελείται κατά κύριο λόγο από κουμπιά, κείμενα και ετικέτες, επομένως οι περισσότερες ρυθμίσεις γίνονται μέσω της συνάρτησης **OnGUI()** στην κλάση **LevelCtrlMobile.js**, καλώντας την κλάση **GUI.Label()** για τη δημιουργία κειμένων και ετικετών, και την **GUI.Button()** για τη δημιουργία κουμπιών.

Υπάρχουν καρτέλες για τον κάθε κόσμο, οι οποίες αποτελούνται από ένα πλαίσιο και μια εικόνα που αντιπροσωπεύει τον κόσμο. Το πλαίσιο αυτό, δημιουργήθηκε με την GUI.Button(), και λειτουργεί ως κουμπί, έτσι ώστε να μπορεί να το επιλέξει ο χρήστης με το δάχτυλό του και να έχουμε μια αντίδραση.

Κάθε φορά στο κέντρο εμφανίζεται μόνο μία καρτέλα, ξεκινώντας με τον κόσμο της SnowLand, και από πάνω της αναγράφεται ο τίτλος του κόσμου με μεγάλα άσπρα γράμματα μέσω της GUI.Label(). Δεξιά και αριστερά από αυτήν βρίσκονται οι υπόλοιπες καρτέλες, οι οποίες εναλλάσονται είτε με τα κουμπιά που βρίσκονται δεξιά και αριστερά, είτε σύροντας το δάχτυλο προς στην αντίστοιχη κατεύθυνση. Για να καταλάβει ο χρήστης ότι υπάρχουν και άλλες καρτέλες, στα άκρα της οθόνης αφήνουμε να φαίνονται ελάχιστα η επόμενη και η προηγούμενη καρτέλα.

Ο έλεγχος της αφής του δαχτύλου του χρήστη και ο υπολογισμός της κατεύθυνσης της κίνησής του γίνεται μέσω του script **SwipeMenu.js**. Αρχικά ορίζουμε μια Vector2 μεταβλητή **StartPos**, που αντιπροσωπεύει δύο σημεία, για τον ορισμό της θέσης του αγγίγματος, μια int μεταβλητή **SwipeID** ίση με -1 για να ξέρουμε πότε να σύρουμε την οθόνη (swipe), και μία float μεταβλητή **minMovement** για να ορίσουμε την ελάχιστη κίνηση που πρέπει να έχει το δάχτυλο για να πραγματοποιήθεί η αντίδραση. Επίσης ορίζουμε μεταβλητές τις οποίες ενεργοποιούμε όταν ο χρήστης σύρει το δάχτυλο του προς κάποια κατεύθυνση. Η boolen μεταβλητή **goLeft** ενεργοποιείται όταν κινήσει το δάχτυλό του προς τα αριστερά για να μετατοπίσουμε τις καρτέλες αριστερά, και η **goRight** όταν το σύρει προς τα δεξιά. Αρχικοποιούμε τις τιμές τους στην συνάρηση Start(). Οι μεταβλητές αυτές είναι static για να μπορεί να έχει πρόσβαση σε αυτές η κλάση LevelCtrlMobile ώστε να μετατοπίζει αντίστοιχα τις καρτέλες των κόσμων.

Η κλάση **Input.touches** επιστρέφει μια λίστα αντικειμένων που αντιπροσωπεύουν την κατάσταση όλων των αγγιγμάτων κατά τη διάρκεια του τελευταίου καρέ. Κάθε είσοδος περιγράφει την κατάσταση αγγίγματος του δαχτύλου.

Κάθε άγγιγμα δαχτύλου αντιπροσωπεύεται από μια **Input.Touch** δομή δεδομένων. Εμάς μας ενδιαφέρουν εξής δεδομένα:

- το **fingerId** που μας ενημερώνει αν βρέθηκε αφή,
- η **position** η οποία αναφέρει τη θέση στην οθόνη αφής, και
- το **phase** που περιγράφει την κατάσταση της αφής, η οποία μπορεί να προσδιορίσει αν το άγγιγμα μόλις ξεκίνησε, αν ο χρήστης μετακινηθεί με το δάχτυλο ή αν απλά σήκωσε το δάχτυλο.

Από τις φάσεις αγγίγματος που προσφέρονται μας ενδιαφέρουν οι:

- **Began**, που σημαίνει ότι ο χρήστης μόλις άγγιξε την οθόνη ένα δάχτυλο,
- **Moved**, που σημαίνει ότι κινήθηκε ένα δάχτυλο στην οθόνη,
- **Ended**, που σημαίνει ότι ένα δάχτυλο σηκώθηκε από την οθόνη και αποτελεί τη τελική φάση ενός αγγίγματος,
- **Canceled**, όταν το σύστημα ακυρώσει την παρακολούθηση της αφής, όπως όταν ο χρήστης τοποθετεί τη συσκευή στο πρόσωπό του ή τοποθετήσει περισσότερα από πέντε δάχτυλα ταυτόχρονα, το οποίο αποτελεί επίσης την τελική φάση της ενός αγγίγματος.

Στη συνάρτηση Update() ελέγχουμε κάθε καρέ την κατάσταση των αγγιγμάτων στην οθόνη. Για κάθε άγγιγμα της Input.touches, το οποίο αποθηκεύεται σε μια μεταβλητή **T**, αποθηκεύουμε τη θέση του στην οθόνη σε μια μεταβλητή **P**. Στη συνέχεια ελέγχουμε με την **TouchPhase.Began** αν ο χρήστης μόλις άγγιξε την οθόνη και αν δεν είχαμε κάποια αφή στην οθόνη ως τώρα (SwipeID == -1), αποθηκεύεουμε στην SwipeID την έρευση αφής μέσω της **fingerId**, και αποθηκεύουμε στην μεταβλητή StartPos τη θέση του αγγίγματος του δαχτύλου πάνω στην οθόνη.

Αλλιώς αν η αφή της οθόνης δεν ξεκίνησε τώρα και είναι η ίδια αφή που αποθηκεύσαμε στην SwipeID, αποθηκεύουμε στην μεταβλητή **delta** τη διαφορά της τωρινής θέσης του αγγίγματος στην οθόνη P, με αυτήν που είχε στην αρχή StartPos. Έπειτα ελέγχουμε αν το δάχτυλο μετακινήθηκε στην οθόνη μέσω της **TouchPhase.Moved** και αν το μήκος της μεταβλητής της διαφοράς θέσης **delta.magnitude** είναι μεγαλύτερο από την ελάχιστη απόσταση μετακίνησης δαχτύλου minMovement που έχουμε ορίσει για να ενεργοποιήσουμε την αντίδραση. Εφόσον ισχύουν και οι δύο αυτοί όροι, θέτουμε πάλι την SwipeID ίση με -1 για να εντοπίσουμε πάλι άλλη αφή αργότερα. Αν η απόλυτη τιμή **Mathf.Abs** της διαφοράς θέσης της αφής στον άξονα των x delta.x, είναι μεγαλύτερη από την απόλυτη τιμή της διαφοράς της θέσης της αφής στον άξονα των y delta.y, και αν διαφορά της θέσης στον άξονα των x είναι θετική τότε ο χρήστης έσυρε το δάχτυλό του προς τα δεξιά και πρέπει να μετακινηθούν οι καρτέλες προς τα δεξιά και θέτουμε την goRight ίση με true, ενώ αν η διαφορά της θέσης στον

άξονα των x είναι αρνητική σημαίνει πως ο χρήστης μετακίνησε το δάχτυλό του προς τα αριστερά και θέτουμε την goLeft ίση με true.

Αν όμως το δάχτυλο του χρήστη αντί να μετακινηθεί, ακυρωθεί, **TouchPhase.Canceled**, ή σηκωθεί από την οθόνη, **TouchPhase.Ended**, θέτουμε την SwipeID ίση με -1 για να εντοπίσουμε την επόμενη αφή.

```
#pragma strict
var StartPos : Vector2;
var SwipeID : int = -1;
var minMovement : float = 20.0f;
static var goLeft : boolean = false;
static var goRight : boolean = false;

// Use this for initialization
function Start ()
{
	goLeft = false;
	goRight = false;
}

// Update is called once per frame
function Update ()
{
	for (var T : Touch in Input.touches) {
		var P = T.position;
		if (T.phase == TouchPhase.Began && SwipeID == -1) {
			SwipeID = T.fingerId;
			StartPos = P;
		}
		else if (T.fingerId == SwipeID) {
			var delta = P - StartPos;
			if (T.phase == TouchPhase.Moved && delta.magnitude > minMovement) {
				SwipeID = -1;
				if (Mathf.Abs (delta.x) > Mathf.Abs (delta.y)) {
					if (delta.x > 0) {
					Debug.Log ("Swipe Right Found");
					goLeft = false;
					goRight = true;
					} else {
					Debug.Log ("Swipe Left Found");
					goLeft = true;
					goRight = false;
					}
```

```
                        }
                    } else if (T.phase == TouchPhase.Canceled || T.phase == TouchPhase.Ended)
                        SwipeID = -1;
                }
        }
}
```

Οι καρτέλες των κόσμων έχουν ίσες αποστάσεις μεταξύ τους και βρίσκονται η μία δίπλα στην άλλη, και πάνω από την κάθε μία βρίσκεται ο τίτλος του αντίστοιχου κόσμου. Με κάθε επιλογή του χρήστη για αλλαγή καρτέλας, μετατοπίζουμε την καρτέλα και τον τίτλο που βρίσκονται στο κέντρο, και μαζί με αυτήν μετατοπίζονται και όλες οι υπόλοιπες μέχρι να βρεθεί στο κέντρο η αμέσως επόμενη, όπου και σταματάει η μετατόπιση. Δηλαδή σε κάθε επιλογή μετατόπισης, μετατοπίζονται όλες οι καρτέλες κατά μία θέση προς την αντίστοιχη κατεύθυνση.

Η μετατόπιση γίνεται σταδιακά με ένα μικρό βήμα για να δημιουργήσουμε μια κίνηση των καρτελών των κόσμων. Κατά τη διάρκεια της μετατόπισης δεν είναι ενεργό κανένα κουμπί, και αν επιλεχθεί οτιδήποτε (καρτέλα κόσμου ή κουμπί) δεν θα έχει καμία αντίδραση μέχρι να βρεθεί στο κέντρο της οθόνης η καρτέλα του κόσμου και σταματήσει. Επίσης, για να φαίνεται ότι δεν μπορεί να επιλεχθεί κάποιο άλλο κουμπί, όση διάρκεια μετακινείται η καρτέλα το αντίστοιχο κουμπί που την μετατόπισε αλλάζει χρώμα και γίνεται πιο έντονο, ώστε να δίνεται η ψευδαίσθηση ότι είναι πιεσμένο. Μόλις βρεθεί η καρτέλα στο κέντρο, επανέρχεται το κουμπί στην κανονική του μορφή.

Η μετατόπιση προς μια συγκεκριμένη κατεύθυνση γίνεται εφόσον υπάρχουν επόμενες καρτέλες μετά από αυτήν. Αν ο χρήστης έχει φτάσει στην τελευταία δεξιά καρτέλα, τότε το κουμπί για την δεξιά μετατόπιση απενεργοποιείται και παίρνει ένα γκρι χρώμα και ο χρήστης μπορεί να κατευθυνθεί μόνο προς τα αριστερά. Αντίστοιχα, αν φτάσει στην τελευταία αριστερή καρτέλα απενεργοποιείται το κουμπί για την αριστερή μετατόπιση παίρνοντας ένα γκρι χρώμα και μπορεί να μεταβεί μονο προς τα δεξιά.

Στο πάνω μέρος της οθόνης δεξιά βρίσκεται ένα κουμπί για την επιστροφή στο κυρίως μενού (Home Button), ενώ στο πάνω αριστερό μέρος βρίσκεται ένα κουμπί για την ενεργοποίηση ή απενεργοποίηση του ήχου, όπου αποθηκεύεται η προτίμηση του χρήστη στην **PlayerPrefs** και αλλάζει ανάλογα και το εικονίδιο του κουμπιού, όπως είχαμε και στο κυρίως μενού.

Επιλέγοντας τον κάθε κόσμο, η καρτέλα αλλάζει, και εμφανίζονται οχτώ μικρά κουτάκια που αντιστοιχούν στις οχτώ πίστες του κόσμου, που πάνω τους αναγράφεται ο αριθμός της πίστας. Αρχικά ο χρήστης έχει πρόσβαση μόνο στην πρώτη πίστα, καθώς οι υπόλοιπες είναι κλειδωμένες, εμφανίζοντας ένα λουκέτο στο κουτάκι που τους αντιστοιχεί. Για να ξεκλειδώσει μια πίστα πρέπει να έχει τερματιστεί επιτυχώς η προηγούμενή της. Για την

εισαγωγή στην κάθε πίστα, αρκεί απλά να επιλέξει ο χρήστης το αντίστοιχο κουτάκι το οποίο λειτουργεί ως κουμπί μέσω της GUI.Button(). Σε κάθε ξεκλείδωτη πίστα αναγράφεται ο αριθμός των νομισμάτων, και απεικονίζονται με έντονο χρώμα τα ψαράκια που έχει συλλέξει ο χρήστης. Κάθε φορά αναγράφεται ο μεγαλύτερος αριθμός νομισμάτων και τα περισσότερα ψάρια που έχουν συλλεχθεί στην συγκεκριμένη πίστα μέσω της GUI.Label().

Για να έχουμε πρόσβαση σε όλα αυτά τα στοιχεία των πιστών, χρησιμοποιούμε την PlayerPrefs για να αποθηκεύουμε τα δεδομένα της κάθε πίστας κατά την ολοκλήρωσή της από τον χρήστη. Σε κάθε μία πίστα αποθηκεύουμε μια μεταβλητή **win**, η οποία είναι ίση με 1 αν έχει τερματιστεί επιτυχώς μέσω της **PlayerPrefs.SetInt()**. Επίσης αν είναι ίση με 1, αποθηκεύουμε πάλι μεσω της PlayerPrefs.SetInt() τον αριθμό των νομισμάτων και τα ψάρια μπόνους που συνέλεξε. Με την **PlayerPrefs.GetInt()**, μπορούμε να ελέγξουμε τη τιμή της μεταβητής win για να ξεκλειδώσουμε ή όχι την επόμενη πίστα, και να αναγράψουμε στο κάθε κουτάκι το μεγαλύτερο σκορ.

Όσο βρισκόμαστε στο μενού των πιστών, δίπλα στα αρχικά κουμπιά στο πάνω μέρος της οθόνης εμφανίζονται άλλα δύο. Το ένα κουμπί βρίσκεται δίπλα στο Home Button, και είναι για την επιστροφή στις καρτέλες των κόσμων εξαφανίζοντας τα κουτάκια των πιστών. Το άλλο είναι για τη διαγραφή όλων των δεδομένων και των σκορ των πιστών του συγκεκριμένου κόσμου, κλειδώνοντας πάλι όλες τις πίστες, για να ξεκινήσουμε από την αρχή. Επειδή τα δεδομένα είναι αποθηκευμένα στην PlayerPrefs, για τη διαγραφή τους θα χρησιμοποιήσουμε την **PlayerPrefs.DeleteKey()**, μία για κάθε μεταβλητή που θέλουμε να διαγράψουμε με το όνομά της μέσα στην παρένθεση.

Το μενού των κόσμων είναι σχεδιασμένο για την όρθια θέση της συσκευής με το κεντρικό κουμπί στη δεξιά ή αριστερή μεριά, και περιστρέφεται αναλόγως τη θέση με την οποία την κρατάμε. Αν την κρατάμε με το κεντρικό κουμπί προς τα κάτω, ή προς τα δεξιά, ή παράλληλα προς το έδαφος με την οθόνη προς τα πάνω, τότε η οθόνη περιστρέφεται σε θέση LandscapeLeft. Αν την κρατάμε με το κεντρικό κουμπί προς τα πάνω, ή προς τα αριστερά, ή παράλληλα προς το έδαφος με την οθόνη προς τα κάτω, τότε η οθόνη περιστρέφεται σε θέση LandscapeRight. Ο έλεγχος της θέσης της συσκευής γίνεται, όπως και στο κυρίως μενού, μέσω του script **TiltControlMenu.js**.

Οι διαστάσεις όλων των GUI είναι ανάλογες με την οθόνη της κάθε συσκευής, διαβάζοντας το ύψος της με την κλάση **Screen.height**, και το πλάτος με την κλάση **Screen.width**. Όταν όμως περιστρέφεται η συσκευή από τη θέση LandscapeLeft στην Portrait, το πλάτος με το ύψος αντιστρέφονται και οι αναλογίες των GUI χαλάνε, ενώ εμείς θέλουμε να παραμένει το μενού σε θέση LandscapeLeft καθώς δεν το περιστρέφουμε. Το ίδιο ισχύει όταν περιστρέφεται από τη θέση LandscapeRight στην PortraitUpsideDown. Για να διατηρήσουμε

φαίνομενικά τις διαστάσεις ελέγχουμε τη θέση της συσκευής και υπολογίζουμε αντίστοιχα και τις διαστάσεις.

Ο έλεγχος αυτός γίνεται στη συνάρτηση Start() καθώς θέλουμε να είναι έτοιμα τα GUI μόλις ενεργοποιείται το μενού των κόσμων. Αν η συσκευή βρίσκεται σε θέση LandscapeLeft, LandscapeRight, FaceUp ή FaceDown το πλάτος των GUI υπολογίζεται από την Screen.width και το ύψος από την Screen.height. Αν η συσκευή βρίσκεται σε θέση Portrait ή PortraitUpsideDown το πλάτος υπολογίζεται από την Screen.height και το ύψος από την Screen.width. Για οποιαδήποτε άλλη θέση οι διαστάσεις υπολογίζονται όπως την πρώτη περίπτωση.

Μέσα από το μενού αυτό, ο χρήστης μπορεί να μεταβεί ελεύθερα σε οποιαδήποτε ξεκλείδωτη πίστα θέλει, οποιουδήποτε κόσμου. Μόλις επιλέξει ο χρήστης μία πίστα, όσο περιμένει να φορτώσει και να ξεκινήσει το παιχνίδι, εμφανίζεται στην οθόνη μια εικόνα με οδηγίες της συγκεκριμένης πίστας, που αφορούν το στόχο, το χειρισμό και τους εχθρούς. Η εικόνα των οδηγιών εμφανίζεται στην οθόνη ανάλογα με τη θέση που κρατάμε τη συσκευή και τη θέση που πρέπει να έχει η συσκευή για την συκεκριμένη πίστα, ώστε να καταλάβει ο χρήστης αν θα πρέπει να την περιστρέψει η όχι. Οι πίστες των κόσμων SnowLand και WaterLand παίζονται καλύτερα σε θέση LandscapeLeft ή LandscapeRight, οπότε οι οδηγίες θα εμφανιστούν σε αντίστοιχη θέση στην οθόνη. Οι κοσμοι JumpLand και RunLand παίζονται σε θέση Portrait ή PortraitUpsideDown, οπότε αντίστοιχα θα εμφανιστούν και οι οδηγίες. Στο κάτω μέρος της οθόνης υπάρχει μια μπάρα φόρτωσης (**Loading Bar**), η οποία αυξάνεται κατά μήκος της οθόνης όσο μειώνεται ο χρόνος αναμονής της φόρτωσης της πίστας. Με αυτόν τον τρόπο ο χρήστης ξέρει πόσος χρόνος του απομένει μέχρι να ξεκινήσει το παιχνίδι.

Η κλάση **Application.LoadLevelAsync()** φορτώνει πλήρως το επίπεδο, και όλα τα αντικείμενα στη σκηνή, ασύγχρονα στο παρασκήνιο. Έτσι μπορούμε να φορτώσουμε νέα επίπεδα, ενώ παίζει ακόμη το τωρινό επίπεδο, παρουσιάζοντας μια γραμμή προόδου (progress bar).

```
var async : AsyncOperation;

//make the game button boxes
if(GUI.Button(Rect ( widthBut8, heightBut8, totalWidth8, totalHeight8 ), label1))
{
        async = Application.LoadLevelAsync(2);                  //Load the scene
        ShowInstructions();
}
```

Η συνάρτηση **ShowInstructions()** σηματοδοτεί την εμφάνιση της εικόνας των οδηγιών για την SnowLand και την WaterLand μέσω της κλάσης **InstructionsCtrl.js**, και καθυστερεί την φόρτωση της πίστας για πέντε δευτερόλεπτα, για να μπορέσει ο χρήστης να τις διαβάσει.

Αντίστοιχα η συνάρτηση ShowInstructions2() σηματοδοτεί την εμφάνιση οδηγιών της JumpLand, και η ShowInstructions3() της RunLand.

```
function ShowInstructions()
{
	showInstr1 = true;		//Show instructions for SnowLand or WaterLand
	async.allowSceneActivation = false;
	yield WaitForSeconds(5);
	async.allowSceneActivation = true;
}
function ShowInstructions2()
{
	showInstr2 = true;		//Show instructions for JumpLand
	async.allowSceneActivation = false;
	yield WaitForSeconds(5);
	async.allowSceneActivation = true;
}
function ShowInstructions3()
{
	showInstr3 = true;		//Show instructions for RunLand
	async.allowSceneActivation = false;
	yield WaitForSeconds(5);
	async.allowSceneActivation = true;
}
```

Στην κλάση InstructionsCtrl θέτουμε τις εικόνες των οδηγιών αναλόγως με την περιστροφή της συσκευής. Το μενού των πιστών είναι σχεδιασμένο για την πλάγια θέση της συσκευής. Όταν επιλέξει ο χρήστης πίστα της SnowLand ή της WaterLand η εικόνα των οδηγιών θέλουμε να εμφανίζεται για την ίδια θέση, οριζόντια. Όταν επιλέξει πίστα της JumpLand ή της RunLand θέλουμε να καταλάβει ότι οι πίστες αυτές παίζονται σε κάθετη θέση του κινητού. Έτσι αν κρατάει τη συσκευή οριζόντια εμφανίζονται οι οδηγίες στην πλάγια θέση περιστραμμένες. Αν περιστρέψει τη συσκευή κάθετα, χρησιμοποιούμε την ίδια εικόνα η οποία φαίνεται κανονικά σε αυτή τη θέση που είναι η σωστή.

Για να καταλάβει ο χρήστης πως πρέπει να περιμένει όσο εμφανίζονται οι οδηγίες, δημιουργούμε μια μπάρα φόρτωσης που ενημερώνει για το χρόνο που απομένει μέχρι την έναρξη της πίστας. Ορίζουμε μια float μεταβλητή **Timer** για να μετράμε το χρόνο από όταν επιλεχθεί μία πίστα μέχρι την εισαγωγή σε αυτήν την πίστα. Επίσης ορίζουμε τρεις Texture2D μεταβλητές στις οποίες θέτουμε τις εικόνες των οδηγιών για το χειρισμό τους. Στη συνάρτηση Start() αρχικά απενεργοποιούμε την εμφάνιση των εικονών, γιατί διαφορετικά θα εμφανίζονται με την εισαγωγή στο μενού των κόσμων, και θέτουμε την Timer ίση με μηδέν.

Στη συνάρτηση Update() όταν επιλεχθεί μία πίστα (και γίνει θετική η showInstr1, ή η showInstr2, ή η showInstr3), αρχίζουμε να υπολογίζουμε τον χρόνο προσθέτοντας στην Timer κάθε καρέ την **Time.deltaTime**, δηλαδή το χρόνο σε δευτερόλεπτα που χρειάζεται για να ολοκληρωθεί το τελευταίο καρέ. Επίσης υπολογίζονται οι διαστάσεις της εικόνας κάθε στιγμή για κάθε περιστροφή της συσκευής, ώστε να καλύβει όλη την οθόνη της συσκευής. Η εικόνα που χρησιμοποιείται είναι η ίδια, και βρίσκεται σε τέτοια θέση ώστε να δηλώνει πως πρέπει να τοποθετηθεί η συσκευή.

Μόλις η συνάρτηση ShowInstructions() σηματοδοτήσει την εμφάνιση των οδηγιών για τη SnowLand ή τη WaterLand μέσω της **showInstr1**, ενεργοποιούμε την εικόνα με τις διαστάσεις υπολογισμένες από την Update(). Αν σηματοδοτήσει η ShowInstructions2() την εμφάνιση των οδηγιών για τη JumpLand μέσω της **showInstr2**, ενεργοποιούμε την εικόνα, και εμφανίζουμε εκείνη με τις οδηγίες της JumpLand. Αντίστοιχα αν ShowInstructions3() θέσει την **showInstr3**, ενεργοποιούμε την εικόνα και εμφανίζουμε τις οδηγίες της RunLand.

Όσο εμφανίζονται οι οδηγίες της SnowLand ή της WareLand, στο κάτω μέρος της οθόνης, μπροστά από την εικόνα οδηγιών εμφανίζουμε τη μπάρα προόδου. Για τον υπολογισμό της μπάρας αυτής, χρησιμοποιούμε τη μεταβλητή Timer στην οποία έχουμε υπολογίσει το χρόνο. Μόλις ενεργοποιηθεί η εικόνα των οδηγιών, δημιουργούμε την μπάρα με τη κλάση **GUI.Box()**, η οποία χρησιμοποιείται για τη δημιουργία ενός πλαισίου-κουτιού. Ορίζουμε τη θέση της να είναι στο κάτω μέρος της οθόνης, και την υπολογίζουμε κατάλληλα για κάθε περιστροφή της συσκευής. Ξεκινάει από το αριστερό κάτω άκρο της οθόνη, και λίγο πιο ψηλά από το κάτω όριο της οθόνης. Στη τιμή του πλάτους της, έχουμε ορίσει την τιμή της μεταβλητής του χρόνου ώστε να αυξάνεται σταδιακά, και για το ύψος έχουμε ορίσει μια σταθερή τιμή. Ο χρόνος που συμβολίζει η μπάρα, διαρκεί πέντε δευτερόλεπτα, όσα δηλαδή έχουμε ορίσει την σκηνή να περιμένει πριν ενεργοποιηθεί, μέσω της **allowSceneActivation.**

Όσο εμφανίζονται οι οδηγίες της JumpLand ή της RunLand, στο δεξιό μέρος της οθόνης, μπροστά από την εικόνα οδηγιών εμφανίζουμε τη μπάρα προόδου. Ορίζουμε τη θέση της να είναι στο κάτω μέρος της εικόνας των οδηγιών ώστε να φαίνεται στο κάτω μέρος της οθόνης αφού περιστρέψει τη συσκευή ο χρήστης στη σωστή θέση, και την υπολογίζουμε κατάλληλα για κάθε περιστροφή της συσκευής. Ξεκινάει από το δεξιό άκρο της οθόνης, λίγο πριν το τέλος για να έχει κάποιο πλάτος η μπάρα, και ανεβαίνει προς το πάνω μέρος της οθόνης με την πάροδο του χρόνου. Στη τιμή του ύψους της, έχουμε ορίσει την αντίθετη τιμή της μεταβλητής του χρόνου ώστε να αυξάνεται σταδιακά προς τα πάνω, και για το πλάτος έχουμε ορίσει μια σταθερή τιμή.

```
#pragma strict
/*
        While waiting for the game to load we show the instructions of each world
```

```
*/
var instructions1 : Texture2D;
var instructions2 : Texture2D;
var instructions3 : Texture2D;

function Start () {

	guiTexture.enabled = false;
	Timer = 0;
}
private var Timer : float;	//to show the time we play

function Update()
{
	if(LevelCtrlMobile.showInstr1 == true || LevelCtrlMobile.showInstr2 == true ||
		LevelCtrlMobile.showInstr3 == true)
	{
		Timer += Time.deltaTime;
	}
	guiTexture.pixelInset.x = -Screen.width/2;
	guiTexture.pixelInset.y = -Screen.height/2;
	guiTexture.pixelInset.width = Screen.width;
	guiTexture.pixelInset.height = Screen.height;
}
function OnGUI ()
{
	if(LevelCtrlMobile.showInstr1 == true)
	{
		guiTexture.enabled = true;
		guiTexture.texture = instructions1;
		GUI.Box(new Rect(0,Screen.height - 40, Timer * Screen.width*0.1,40),"");
	}
	else if(LevelCtrlMobile.showInstr2 == true)
	{
		guiTexture.enabled = true;
		guiTexture.texture = instructions2;
		GUI.Box(new Rect(Screen.width - 40,Screen.height,40,
							-(Timer * Screen.width*0.1)),"");
	}
	else if(LevelCtrlMobile.showInstr3 == true)
	{
		guiTexture.enabled = true;
		guiTexture.texture = instructions3;
```

```
                GUI.Box(new Rect(Screen.width - 40,Screen.height,40,
                                                        -(Timer * Screen.width*0.1)),"");
        }
}
```

Στο τέλος του χρόνου της μπάρας, απενεργοποιούμε την εικόνα οδηγιών, αρχικοποιούμε όλες τις τιμές και μεταβλητές, και ξεκινάει η σκηνή της πίστας.

Όταν ο χρήστης επιστρέψει από μια πίστα στο μενού των κόσμων, θα βρεθεί στο μενού με τις πίστες του αντίστοιχου κόσμου, και όχι στην αρχική καρτέλα του SnowLand.

Στο πίσω μέρος υπάρχει ένα φόντο με τον ουρανό, το οποίο αποδίδεται μέσω ενός κύβου με την αντίστοιχη υφή.

5.5 Οργάνωση των SnowLand και WaterLand

5.5.1 Έλεγχος περιστροφής της συσκευής

Το παιχνίδι σε αυτούς τους κόσμους μπορεί να παίξει σε όρθια, ανάποδα όρθια και πλάγια θέση της συσκευής, είτε αριστερά είτε δεξιά. Είναι προτιμότερη όμως η πλάγια θέση λόγω πολυπλοκότητας της πίστας.

Αν αφήσουμε τη συσκευή κάτω, με την οθόνη προς τα πάνω ή προς τα κάτω, το παιχνίδι σταματάει και εμφανίζεται ένα μήνυμα παύσης μέχρι να την ξανασηκώσουμε, όπου το παιχνίδι συνεχίζει από το σημείο που το αφήσαμε.

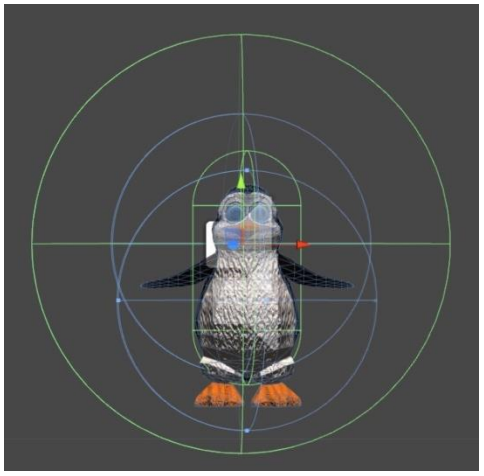
Σε ένα κενό GameObject έχουμε συνδέσει ένα συστατικό για το GUITexture για την εμφάνιση της εικόνας, και ένα συστατικό script για τον έλεγχο της θέσης της συκευής και την ενεργοποίηση της εικόνας μέσω του **TiltControlMoveAround.js**.

```
#pragma strict
/*
        DeviceOrientation: Describes physical orientation of the device as determined by the OS.
*/
var FaceUpTexture : Texture2D;
var FaceDownTexture : Texture2D;

function Start()
{
        guiTexture.enabled = false;
}

static var canPlay : boolean = false;

function Update ()
```

```
{
	//FaceUp: The device is held parallel to the ground with the screen facing upwards.
	if (Input.deviceOrientation == DeviceOrientation.FaceUp)
	{
		//Pause the game
		guiTexture.pixelInset.x = -Screen.width/2;
		guiTexture.pixelInset.y = -Screen.height*1.122;
		guiTexture.pixelInset.width = Screen.width;
		guiTexture.pixelInset.height = Screen.height*1.68;
		guiTexture.enabled = true;
		guiTexture.texture = FaceUpTexture;
		canPlay = false;
		Time.timeScale = 0;
	}
	//FaceDown: The device is held parallel to the ground with the screen facing downwards.
	else if (Input.deviceOrientation == DeviceOrientation.FaceDown)
	{
		//Pause the game
		guiTexture.pixelInset.x = -Screen.width/2;
		guiTexture.pixelInset.y = -Screen.height*1.122;
		guiTexture.pixelInset.width = Screen.width;
		guiTexture.pixelInset.height = Screen.height*1.68;
		guiTexture.enabled = true;
		guiTexture.texture = FaceDownTexture;
		canPlay = false;
		Time.timeScale = 0;
	}
	else
	{
		guiTexture.enabled = false;
		canPlay = true;
		if(PauseTouch.pauseNow == false)
		{
			Time.timeScale = 1;
		}
	}
}
```

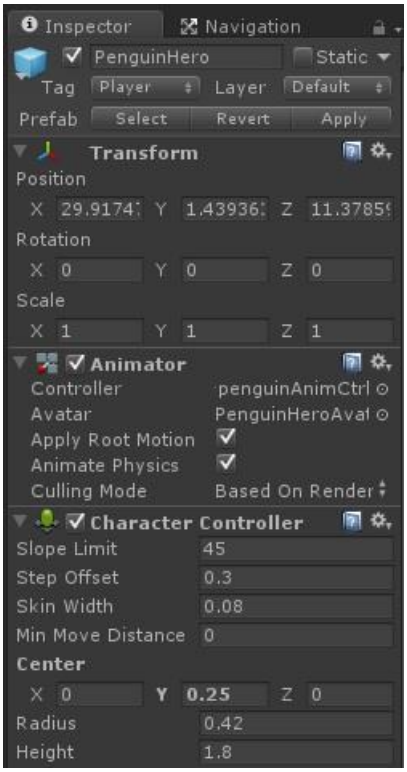
Στις μεταβλητές Texture2D ορίζουμε την εικόνα που θέλουμε να εμφανίζεται σε κάθε περίπτωση, και μόνο η συγκεκριμένη η κλάση έχει πρόσβαση σε αυτές. Η static μεταβλητή **canPlay** μπορεί να διαβαστεί και από άλλες κλάσεις, και χρησιμοποιείται για να διακόπτεται το παιχνίδι όταν ο χρήστης αφήνει τη συσκευή, και να συνεχίζεται όταν είναι στη σωστή θέση. Στη συνάρτηση **Start()** θέτουμε ανενεργή την εικόνα όταν ξεκινάει το παιχνίδι γιατί δεν μπορούμε να γνωρίζουμε εξαρχής την κλίση της συσκευής.

Η **Update()** συνάρτηση καλείται κάθε καρέ, εάν ο κώδικας είναι ενεργοποιημένος. Η Update είναι η πιο συχνά χρησιμοποιούμενη συνάρτηση για την εφαρμογή κάθε είδους συμπεριφοράς παιχνιδιού. Σε αυτήν ελέγχουμε αν ο φυσικός προσανατολισμός της συσκευής είναι ίσος με αυτόν με την παράλληλη θέση της συσκευής ως προς το έδαφος με την οθόνη προς τα πάνω που δηλώνεται με την **DeviceOrientation.FaceUp**, ή με την οθόνη προς τα κάτω που δηλώνεται με την **DeviceOrientation.FaceDown**. Σε μία τέτοια περίπτωση ενεργοποιούμε την εικόνα ώστε να φαίνεται, ορίζουμε την εικόνα που θέλουμε να εμφανιστεί στην οθόνη, ρυθμίζουμε το μέγεθός της και η θέση της ώστε να εμφανίζεται σε όλη την έκταση της οθόνης οποιασδήποτε συσκευής διαβάζοντας το ύψος και το πλάτος της εκάστοτε οθόνης ορίζοντας αντίστοιχα και τις διαστάσεις της εικόνας, σταματάμε το χρόνο του παιχνιδιού και θέτουμε τη μεταβλητή canPlay ίση με false για να μην μπορεί να συνεχιστεί το παιχνίδι. Σε οποιαδήποτε άλλη θέση της συσκευής απενεργοποιούμε την εικόνα, θέτουμε πάλι τη μεταβλητή canPlay ίση με true για τη συνέχιση του παιχνιδιού, και εφόσον δεν έχει επιλεχθεί το κουμπί Παύσης, συνεχίζεται ο χρόνος του παιχνιδιού.

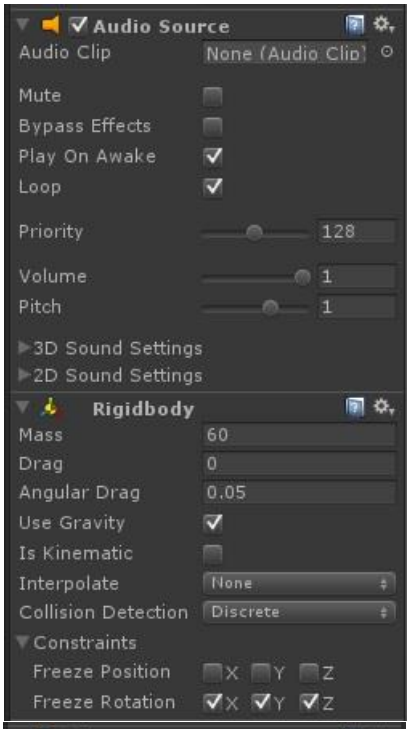


5.5.2 Υλοποίηση του ήρωα

Η κίνηση και η περιστροφή του ήρωα γίνεται μέσω ενός χειριστηρίου αφής (joystick), οι οποίες συνδέονται άμεσα με αυτό. Για τον λόγο αυτό ο ήρωας και το χειριστήριο είναι παιδιά ενός κενού GameObject. Ο πιγκουίνος αποτελείται από πολλά διαφορετικά συστατικά, όπου το καθένα είναι υπεύθυνο για μια διαφορετική δράση.

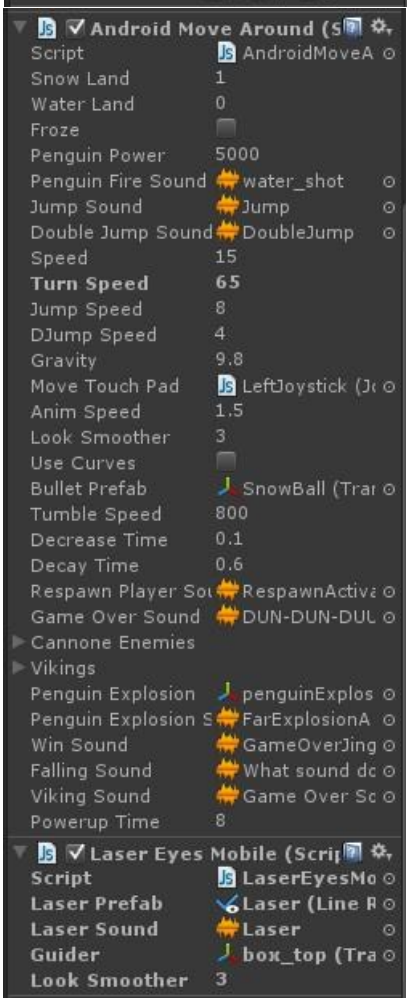


- Το Transform καθορίζει τη θέση του ήρωα στο χώρο, την περιστροφή και το μέγεθός του.
- Ο Animator είναι η διεπαφή για τον έλεγχο του συστήματος animation Mecanim, που καθορίζει τα animations της κίνησής του Avatar, αναλόγως την κατάσταση στην οποία βρίσκεται. Οι αλλαγές μεταξύ των καταστάσεων αυτών γίνονται μέσω του Animation Controller που έχουμε ορίσει, τον **penguinAnimCtrl**.
- Ο Character Controller του δίνει φυσική υπόσταση στα πλαίσια του πράσινου κυλίνδρου που τον περιβάλλει, ώστε να μπορεί να αλληλεπιδρά με το περιβάλλον και να κινείται. Φροντίζουμε ώστε ο κύλινδρος να καλύπτει όλη την επιφάνεια του σώματος του πιγκουίνου για να προσωμοιωθεί σωστά η συμπεριφορά της φυσικής. Ορίσαμε μια τιμή στο Slope Limit, ο οποίος περιορίζει τον επιταχυντή (collider) να ανέβει μόνο πλαγιές που είναι ίσες ή μικρότερες από την αναγραφόμενη τιμή, έτσι ώστε να ανεβαίνει πλαγιές ανάλογα το μεγεθός του.



- Η Audio Source παράγει τους ήχους του πηγάζουν από αυτόν, και συμβολίζεται με το σήμα του ηχείου.
- Το Rigidbody ελέγχει τη κίνησή του μέσω της προσομοίωσης της φυσικής, και δηλώνεται με τον μπλε κύκλο που τον περιβάλλει. Ορίσαμε τη μάζα του ήρωα για την προσωμοίωση της φυσικής, η οποία είναι ανάλογη του μεγέθους του.
- Το script **AndroidMoveAround.js** είναι ο κώδικας που ελέγχει όλη την κίνηση και δράση του, ελέγχοντας τα παραπάνω συστατικά.
- Το **LayserEyesMobile.js** είναι για τον έλεγχο του κεφαλιού του όταν κοιτάει το θησαυρό ή τη φάλαινα οδηγό.

Ο πιγκουίνος είναι γονέας ενός «μαγνήτη» για τη συλλογή των αντικειμένων, και κάποιων κενών GameObjects που χρησιμοποιούμε για τον προσδιορισμό της θέσης κάποιων Check Points.



Για να δημιουργήσουμε την ψευδαίσθηση ότι τα αντικείμενα συλλογής (νομίσματα, ψαράκια μπόνους, μαγικό σμαράγδι, ζωές και πιγκουινάκια) μαγνητίζονται από τον ήρωα και κατευθύνονται προς αυτόν όταν τα πλησιάζει, δημιουργήσαμε ένα κενό GameObject, πάνω στον πιγκουίνο που τον ακολουθεί παντού. Του προσαρτήσαμε έναν Sphere Collider, ο οποίος είναι ο μεγάλος πράσινος κύκλος γύρω από τον πιγκουίνο, για να ξέρουμε πότε έρχεται σε επαφή με κάθε αντικείμενο, στα πλαίσια αυτού του κύκλου. Επίσης αποτελείται από το συστατικό Audio Source, που χρησιμοποιείται για την απόδοση ήχων των αντικειμένων που έρχεται σε επαφή. Ελέγχεται από το script **MagnetControl.js**, και ενημερώνεται για την αναπαραγωγή των ήχων από Boolean μεταβλητές, και από τον ίδιο τον collider.

Έχουμε εισάγει μια μεγάλη ποικιλία ήχων για να δώσουμε όσο το δυνατόν περισσότερο ενδιαφέρον στο παιχνίδι. Οι ήχοι που ελέγχει ο μαγνήτης είναι ο ήχος με την συλλογή κάθε νομίσματος, καρδούλας, μικρού πιγκουίνου, ψαριού μπόνους, μαγικού σμαραδιού, ήχος με την ρίξη χιονόμπαλας, ήχος για το απλό και διπλό άλμα, ήχος για την επαναφορά του στον αρχικό σημείο με το χάσιμο μιας ζωής, ήχος τέλος παιχνιδιού όταν χάνει και όταν κερδίζει, ήχος όταν τον χτυπάει ένα βλήμα κανονιού και δημιουργείται μια έκρηξη σε κάθε πιγκουίνο, ένας ήχος κραυγής όταν πέφτει στο κενό, και ένας ήχος που πεθαίνει όταν τον πιάσει ο βίκινγκ.

Τα Check Points τα χρησιμοποιούμε για πολλούς σκοπούς στο παιχνίδι:

- Τρία για την αλλαγή της θέσης της κύριας κάμερας. Ένα για όταν πεθαίνει ο ήρωας και θέλουμε να φαίνεται από την μπροστινή του μεριά, ένα για όταν κοιτάει το θησαυρό ή τη φάλαινα οδηγό που η κάμερα πηγαίνει στο ύψος του κεφαλιού του, και ένα για όταν θέλει να κοιτάξει την πίστα από ψηλά.
- Ένα για το σημείο εκτόξευσης των χιονόμπαλων του πιγκουίνου μπροστά από αυτόν, στο ύψος των χεριών του.
- Δύο Check Points για τον προσδιορισμό της θέσης των ματιών του ήρωα, ώστε όταν κινήσει το κεφάλι του και κοιτάξει τον θησαυρό ή τη φάλαινα οδηγό, να ξέρουμε από ποιο σημείο να ξεκινάει το λέιζερ που θα δείχνει προς στο σημείο που θα κοιτάει μέσω της LayserEyesMobile.
- Δύο για τον ορισμό του σημείου πίσω από τον ήρωα, με το οποίο ακουλουθεί το πρώτο μικρό πιγκουινάκι τον ήρωα. Το ένα χρησιμοποιείται για να ορίσουμε το ύψος με το οποίο θα τον ακολουθεί κατά τη διάρκεια του παιχνιδιού, ώστε να παραμένει το πιγκουινάκι χαμηλά στο έδαφος. Το δεύτερο χρησιμοποιείται όταν ο πιγκουίνος κολυμπάει σε βαθιά νερά και πρέπει τα πιγκουινάκια να πάνε πιο ψηλά για να φαίνονται.

Όλη η συμπεριφορά του πιγκουίνου καθορίζεται μέσα στην κλάση AndroidMoveAround. Στην συνάρτηση **Start()** αρχικοποιούμε όλες τις μεταβλητές, κάθε φορά που ξεκινάει μία πίστα, και έχουμε πρόσβαση στα συστατικά του πιγκουίνου από τα οποία αποτελείται ώστε να μπορέσουμε να τα χειριστούμε, όπως το συστατικό Transform για τον έλεγχο της θέσης, το CharacterController για την κίνηση και τον Animator για τον έλεγχο των animations. Επειδή έχουμε δύο στρώματα animations, ορίζουμε το βάρος ίσο με την τιμή 1 ώστε να ξεκινάμε από το πρώτο στρώμα μέσω της **Animator.SetLayerWeight**. Επίσης ελέγχουμε αν θα γίνει αναπαραγωγή του ήχου στο παιχνίδι αναλόγως με την επιλογή του χρήστη στα αρχικά μενού.

```
function Start ()
{
        // Cache component lookup at startup instead of doing this every frame
        thisTransform = GetComponent( Transform );
        controller = GetComponent(CharacterController);
        anim = GetComponent(Animator);
        if(anim.layerCount ==2)                         //start always from Base Layer
                anim.SetLayerWeight(1, 1);
}
```

Για να ενεργοποιηθεί οποιαδήποτε λειτουργία της σκηνής, πρέπει να βρίσκεται η συσκευή σε σωστή θέση, δηλαδή να μην είναι παράλληλη προς το έδαφος, το οποίο το ελέγχουμε από

την μεταβλητή canPlay της TiltControlMoveAround. Η κίνηση του ήρωα καθορίζεται κυρίως στην συνάρτηση **Update()**, εφόσον βρίσκεται η συσκευή στη σωστή θέση.

Αν ο χρήστης τοποθετήσει τρία δάχτυλα ταυτόχρονα στην οθόνη τότε το παιχνίδι σταματάει και εμφανίζεται το μενού Παύσης. Στη μεταβλητή **fingerCount** αποθηκεύουμε τον αριθμό των δαχτύλων που βρίσκονται στην οθόνη το κάθε καρέ. Για κάθε άγγιγμα στην οθόνη ελέγχουμε αν η φάση αφής τελείωσε **TouchPhase.Ended** ή ακυρώθηκε **TouchPhase.Canceled**. Σε περίπτωση που δεν ισχύει καμία από αυτές τις δύο περιπτώσεις προσθέτουμε στην μεταβλητή fingerCount τον αριθμό των αγγιγμάτων που εντοπίστηκαν στο συγκεκριμένο καρέ. Αν ο αριθμός των αγγιγμάτων είναι ίσος με τρία, σταματάμε το χρόνο του παιχνιδιού μέσω της **Time.timeScale** και θέτουμε τη μεταβλητή PauseFingers ίση με true η οποία ενεργοποιεί το μενού Παύσης.

Η κίνηση και η περιστροφή του ήρωα ελέγχεται από τις εισόδους του χειριστηρίου, τις οποίες διαχειριζόμαστε στην Update(). Το χειριστήριο είναι ενεργό και μόνο κατά της διάρκεια του παιχνιδιού, δηλαδή εφόσον η συσκευή βρίσκεται στη σωστή θέση και ο χρήστης δεν έχει χάσει ή κερδίσει, και δεν έχει επιλέξει παύση του παιχνιδιού.

Θέτουμε σε μια μεταβλητή **moveTouchPad** το χειριστήριο για να έχουμε πρόσβαση σε αυτό. Για την περιστροφή του ήρωα, θέτουμε στη μεταβλητή **horizontal** την κίνηση του χειριστηρίου στον άξονα x, **moveTouchPad.position.x**, ενώ για την κάθετη κίνηση θέτουμε στη μεταβλητή **vertical** την κίνηση του χειριστηρίου στον άξονα των y, **moveTouchPad.position.y**.

Σε περιπτώσεις που θέλουμε να παγώσουμε την κίνηση του ήρωα, όπως όταν πεθαίνει η επανέρχεται στο αρχικό του σημείο, έχουμε ορίσει μια Boolean μεταβλητή **Froze**, η οποία όταν είναι αληθής μηδενίζουμε τις τιμές των μεταβλητών της οριζόντιας και κάθετης κίνησης.

Η **Transform.Rotate** εφαρμόζει μία περιστροφή (eulerAngles.z) γύρω από τον άξονα z, (eulerAngles.x) γύρω από τον άξονα x, και (eulerAngles.y) γύρω από τον άξονα y (με αυτή τη σειρά). Εμάς μας ενδιαφέρει η περιστροφή μόνο γύρω από τον άξονα των y. Οπότε μηδενίζουμε την περιστροφή στους άλλους δύο άξονες, ενώ στον άξονα των y, πολλαπλασιάζουμε την είσοδο περιστροφής από το χειριστήριο με την ταχύτητα περιστροφής ομοιόμορφα με το χρόνο Time.deltaTime.

Το **Time.deltaTime** δίνει το χρόνο σε δευτερόλεπτα που χρειάστηκε για να ολοκληρωθεί το τελευταίο καρέ. Χρησιμοποιείται για να γίνει το παιχνίδι ανεξάρτητο από τον ρυθμό των καρέ. Όταν προσθέτουμε ή αφαιρούμε μια τιμή κάθε πλαίσιο, πρέπει να την πολλαπλασιάσουμε με Time.deltaTime, ώστε να γίνεται η κίνηση που θέλουμε ανά δευτερόπλεπτο και όχι ανά πλαίσιο.

Η **TransformDirection** μεταμορφώνει την κατεύθυνση x, y, z από τον τοπικό χώρο στον παγκόσμιο χώρο και δεν επηρεάζεται από την κλίμακα ή τη θέση του μετασχηματισμού, καθώς το διάνυσμα που επιστρέφει έχει το ίδιο μήκος με την κατεύθυνση. Αποθηκεύουμε την κατεύθυνση της κίνησης στον άξονα x και y στο διάνυσμα **vel**, καθώς μας ενδιαφέρει να κινείται ο πιγκουίνος μόνο μπροστά και πίσω. Ομαλοποιούμε τη μεταβλητή αυτή με την **Normalize** ώστε να έχει μέγεθος 1 διατηρώντας την ίδια κατεύθυνση.

Για να εφαρμόσουμε την κίνηση του ήρωα από την κίνηση του χειριστηρίου, αποθηκεύουμε σε ένα δισδιάστατο διάνυσμα **absJoyPos** τις απόλυτες τιμές της οριζόντιας και κάθετης κίνησής του. Αν η απόλυτη τιμή του διανύσματος αυτού στον άξονα των y, που αντιστοιχεί στην κάθετη κίνηση του χειριστηρίου, είναι μεγαλύτερη από την απόλυτη τιμή του διανύσματος στον άξονα των x, που αντιστοιχεί στην οριζόντια κίνηση του χειριστηρίου, τότε θα κινηθεί ο πιγκουίνος κάθετα. Αν η μεταβλητή vertical είναι θετική θα κινηθεί προς τα εμπρός, ενώ αν είναι αρνητική θα κινηθεί προς τα πίσω. Στη μεταβλητή που έχουμε ορίσει την κίνηση του πιγκουίνου vel, πολλαπλασιάζουμε την ταχύτητα κίνησης και την απόλυτη τιμή κίνησης του χειριστηρίου στον άξονα y. Αν όμως το διάνυσμα absJoyPos δεν έχει μεγαλύτερη τιμή στον άξονα των y από τον άξονα των x, τότε ο πιγκουίνος θα κινηθεί οριζόντια, οπότε στη μεταβλητή κίνησης πολλαπλασιάζουμε την ταχύτητα κίνησης με την απόλυτη τιμή της κίνησης του χειριστηρίου στον άξονα των x.

Ο πιγκουίνος μπορεί να πηδήξει μόνο αν βρίσκεται στο έδαφος, δηλαδή αν ο κύλινδρος του Character Controller ακουμπάει το terrain. Το **CharacterController.isGrounded** μας ενημερώνει αν ο Character Controller άγγιξε το έδαφος κατά την τελευταία κίνηση. Όσο βρίσκεται στο έδαφος η ταχύτητα άλματος είναι μηδέν, την οποία την ορίζουμε στη μεταβλητή **vSpeed**. Αν ο χρήστης πατήσει το κουμπί άλματος, θέτουμε στην vSpeed την ταχύτητα άλματος που θέλουμε, και θέτουμε αληθή τη μεταβλητή **Jump**, μέσω της **Animator.SetBool,** η οποία μας εισάγει στην κατάσταση του animation του άλματος, ενεργοποιούμε τον ήχο του άλματος, και θέτουμε τη μεταβλητή **canDJump** ίση με true για να μπορέσει να πηδήξει δεύτερη φορά όσο είναι στον αέρα αν το επιλέξει.

Αν βρίσκεται στον αέρα, και ο Character Controller δεν ακουμπάει το έδαφος, μπορεί να ξαναπηδήξει και ο χρήστης πατήσει το κουμπί άλματος, θέτουμε στην μεταβλητή vSpeed την ταχύτητα του διπλού άλματος, θέτουμε αληθή τη μεταβλητή **DoubleJump** η οποία μας εισάγει στην κατάσταση του animation του διπλού άλματος, ενεργοποιούμε τον ήχο του διπλού άλματος, και θέτουμε πάλι τη μεταβλητή canDJump με false καθώς δεν μπορεί να ξαναπηδήξει.

Αν βρίσκεται στον αέρα, μπορεί να ξαναπηδήξει αλλά ο χρήστης δεν πατήσει το κουμπί άλματος, τότε πέφτει προς το έδαφος και θέτουμε αληθή τη μεταβλητή **Fall** η οποία μας

εισάγει στην κατάσταση του animation της προσγείωσης. Αν ο χρήστης επιλέξει το κουμπί άλματος όσο πέφτει ο πιγκουίνος αλλλά πριν προσγειωθεί, θα ενεργοποιηθεί το διπλό άλμα.

Από τη ταχύτητα άλματος αφαιρούμε ανά δευτερόλεπτο μια τιμή **gravity** που έχουμε ορίσει ως βαρύτητα προσομοιώνοντας τη φυσική, ώστε να προσειωθεί πάλι στο έδαφος και να μην παραμείνει στον αέρα. Την ταχύτητα αυτή την θέτουμε στον άξονα y της μεταβλητής vel που καθορίζει την κίνηση στον άξονα αυτό.

Η **CharacterController.Move()** πραγματοποιεί την κίνηση μέσω του Character Controller, εισάγοντας ως είσοδο τη μεταβλητή κίνησης vel πολλαπλασιασμένη με Time.deltaTime για να πραγματοποιείται η κίνηση ανά δευτερόλεπτο.

Αν ο χρήστης πατήσει το κουμπί πυροβολισμού, δημιουργείται μια χιονόμπαλα από ένα αόρατο σημείο μπροστά από τον πιγκουίνο στο ύψος των χεριών του και εκτοξεύεται με δύναμη προς την κατεύθυνση που κοιτάει. Έχουμε φτιάξει ένα Prefab μιας χιονόμπαλας για να μπορέσουμε να την δημιουργήσουμε όσες φορές θέλουμε. Με την **Object.Instantiate** κλωνοποιούμε το αρχικό αντικείμενο και επιστρέφουμε τον κλώνο, στο οποίο μπορούμε να καθορίσουμε τη θέση με την **Vector3.zero** και την περιστρφή με την **Quaternion.identity**. Την χιονόμπαλα που έχουμε δημιουργήσει την εντοπίζουμε με την **GameObject.Find** η οποία εντοπίζει το αντικείμενο από το όνομα που εισάγουμε και το επιστρέφει.

Σε κάθε χιονόμπαλα που δημιουργείται προσθέτουμε μια ετικέτα μέσω της **GameObject.tag** για να μπορέσουμε να την εντοπίσουμε στο παιχνίδι. Με την **Rigidbody.AddForce** προσθέτουμε δύναμη στη χιονόμπαλα για να αρχίσει να κινείται, την οποία την ορίζουμε με την **transform.forward** καθώς θέλουμε να κινηθεί μόνο προς τα εμπρός και την πολλαπλασιάζουμε με μια ταχύτητα. Όταν εκτοξεύεται η χιονόμπαλα ενεργοποιούμε και το animation κίνησης του χεριού του πιγκουίνου.

Αν ο πιγκουίνος χτυπηθεί από τα κανόνια δύο φορές και χάσει όλα τα πιγκουινάκια, ενεργοποιείται το animation με το οποίο περπατάει πληγωμένος θέτοντας αληθή τη μεταβλητή **IsWounded**.

Αν η συσκευή βρεθεί παράλληλη προς το έδαφος, με την οθόνη προς τα πάνω ή προς τα κάτω απενεργοποιούνται όλες οι λειτουργίες μέχρι να βρεθεί στη σωστή θέση.

```
var moveTouchPad : Joystick;
function Update ()      //function which is read in every frame.it must listen to the character
                        //controller and move the character around
{
if(TiltControlMoveAround.canPlay)
{
        //PAUSE IF THREE FINGERS ARE ON THE SCREEN
```

```
var fingerCount = 0;
for (var touch : Touch in Input.touches) {
        if (touch.phase != TouchPhase.Ended && touch.phase != TouchPhase.Canceled)
                fingerCount++;          //Each entry represents a status of a finger touching
                                        //the screen.
}
if (fingerCount == 3)                           //User has 3 fingers touching the screen
{
        Time.timeScale = 0;
        PauseFingers = true;                    //Pause
}

if( HealthControl.GameOver == true || HealthControl.WinGame == true ||
        PauseTouch.pauseNow == true || AndroidMoveAround.PauseFingers == true )
{
        moveTouchPad.active = false;
}
if(HealthControl.GameOver == false && HealthControl.WinGame == false &&
                PauseTouch.pauseNow == false && TiltControlMoveAround.canPlay == true
                                && AndroidMoveAround.PauseFingers == false )
{
        moveTouchPad.active = true;
}

var horizontal = moveTouchPad.position.x;
var vertical = moveTouchPad.position.y;
if(Froze){
        horizontal = 0;
        vertical = 0;
}
// Rotate around y - axis .
//We are only interested on the y axis and we multiply it with the speed vector
transform.Rotate(0, horizontal * turnSpeed * Time.deltaTime, 0);

// Move forward / backward
var vel = thisTransform.TransformDirection( Vector3(horizontal, 0, vertical ) );
vel.Normalize();

// Apply movement from move joystick
var absJoyPos = Vector2( Mathf.Abs( horizontal ), Mathf.Abs( vertical ) );
if ( absJoyPos.y > absJoyPos.x )
{
        if ( vertical > 0 )
                vel *= speed * absJoyPos.y;
```

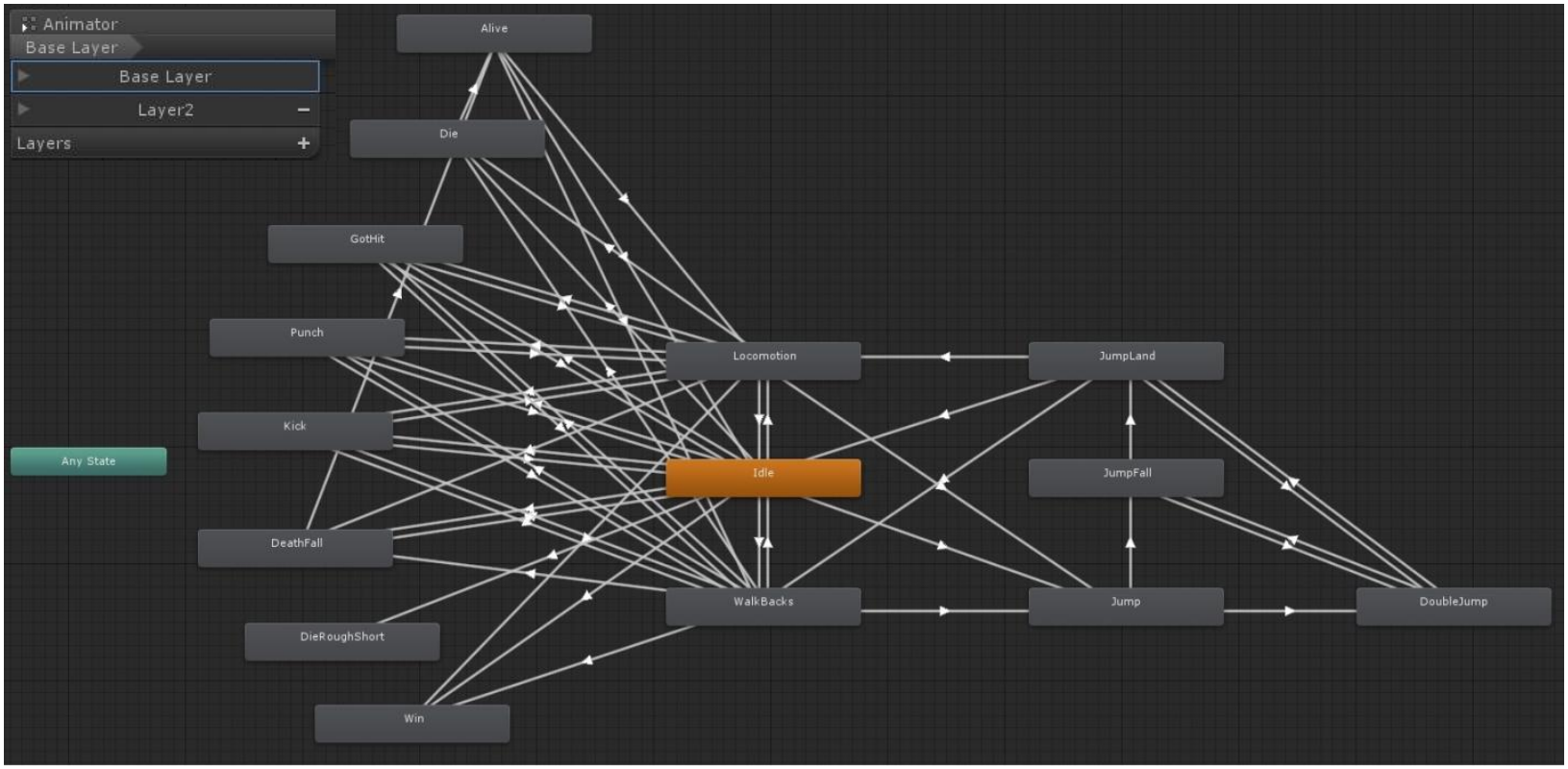
```
            else
                vel *= speed * absJoyPos.y;
        }
        else
            vel *= speed * absJoyPos.x;

        // Check for JUMP
        if (controller.isGrounded)                  // We are grounded, so recalculate
        {
            vSpeed = 0;                             // grounded character has vSpeed = 0...
            if (TouchJump.jumpNow)                  //JUMP
            { // unless it jumps:
                vSpeed = jumpSpeed;
                anim.SetBool("Jump", true);     //SetBool : Sets the value of a bool parameter.
                playJumpSound = true;
                canDJump = true;
            }
        }
        if (!controller.isGrounded && canDJump && (TouchJump.jumpNow))
            //he has no time to  jump higher,he just jumps with the doubleJump clip
        {
            vSpeed = dJumpSpeed;
            anim.SetBool("DoubleJump", true);               //DOUBLE JUMP
            playDoubleJumpSound = true;
            anim.SetBool("Fall", false);
            canDJump = false;
        }
        else if (!controller.isGrounded && canDJump && (!TouchJump.jumpNow))
            //he jumps higher if we           press the jump button after a while since the first one
        {
            anim.SetBool("Fall", true);                 //FALLING
            if (!controller.isGrounded && canDJump && (TouchJump.jumpNow))
            {
                vSpeed = dJumpSpeed;
                anim.SetBool("DoubleJump", true);       //DOUBLE JUMP after FALLING
                playDoubleJumpSound = true;
                canDJump = false;
            }
        }
        // apply gravity acceleration to vertical speed:
        vSpeed -= gravity * Time.deltaTime;
        vel.y = vSpeed; // include vertical speed in vel

        // convert vel to displacement and Move the character:
```

```
        controller.Move(vel * Time.deltaTime );

        // SHOOTING! we need to check if we are shooting, we will use the CTRL button to shoot
        if(TouchButtons.shootNow)
        {       // we need to start creating the bullet in our world
                var bullet = Instantiate(bulletPrefab,
                        transform.Find("spawnPoint").transform.position,Quaternion.identity);
                        // Instantiate function creates the prefab in the world
                // it needs 3 variables (what we are creating-the prefab, the position where we want
                //to create it, the angles that it will look towards)
                // GameObject.Find will look to all the object to the current scene to find the
                //"GivenName", and transform.positon will give its position
                // Quaternion.identity finds the angles of the worm
                // we need to add some force to the bullet
                bullet.tag = "penguinProjectile";       //we tag the snowballs from penguin as
                                                        //penguinProjectiles
                bullet.rigidbody.AddForce(transform.forward * PenguinPower);
                        //Adds a force to the rigidbody. As a result the rigidbody will start moving.
                anim.SetBool("Punch", true);    //SHOOTING by throwing the snowball
        }
        if(HealthControl.HITS == 2)
        {
                anim.SetBool("IsWounded", true);
        }
        else if(HealthControl.HITS == 0 ||HealthControl.HITS == 1 || HealthControl.HITS == 3)
        {
                anim.SetBool("IsWounded", false);
        }
}
else if(TiltControlMoveAround.canPlay == false)
{
        moveTouchPad.active = false;
}
}
```

Έχουμε δημιουργήσει δύο στρώματα για να αναπαραχθούν τα animation, το βασικό στρώμα που ονομάσαμε **Base Layer**, και ένα δεύτερο στρώμα που ονομάσαμε **Layer2** και πανωγράφει τα animations του βασικού στρώματος. Οι μεταβάσεις μεταξύ των καταστάσεων των κινουμένων σχεδίων γίνονται μέσω των παραμέτρων που έχουμε ορίσει (βέλη) και τις τιμές που τους δίνουμε μέσω του κώδικα.

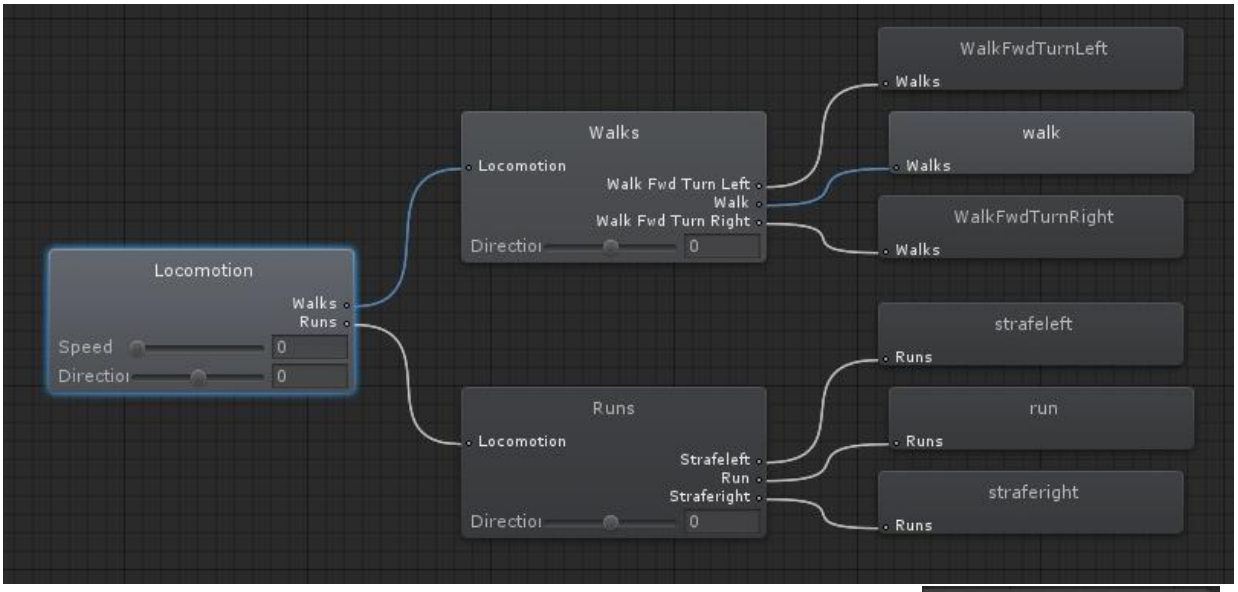
Ο πιγκουίνος αρχικά παραμένει ακίνητος με το animation **Idle** περιμένοντας κάποια ενέργεια από το χρήστη. Αν προχωρήσει προς τα εμπρός μπαίνει στην κατάσταση **Locomotion**, η οποία περιλαμβάνει άλλες δύο καταστάσεις **Walks** και **Runs** που αναλόγως την ταχύτητα

Speed που αναπτύσσεται αναπαράγεται το περπάτημα **walk** ή το τρέξιμο **run**, και αναλόγως την κατεύθυνση **Direction** της κίνησης στρίβοντας δεξιά ή αριστερά ενεργοποιείται για το περπάτημα το **WalkFwdTurnLeft** ή το **WalkFwdTurnRight** για να στρίψει αντίστοιχα, ενώ στο τρέξιμο το **strafeleft** ή το **straferight**. Αν προχωρήσει προς τα πίσω, η ταχύτατα Speed παίρνει αρνητικές τιμές, οπότε μπαίνει στην κατάσταση **WalkBacks** όπου πηγαίνει προς τα πίσω με το animation **WalkBack**, και αν στρίβει παράλληλα ενεργοποιούνται τα animations **WalkBackTurnLeft** και **WalkBackTurnRight** μέσω της Direction. Αναλόγως την ενέργεια την οποία πραγματοποιεί και την κατάσταση στην οποία βρίσκεται εκείνη τη στιγμή (είναι ακίνητος ή προχωράει προς τα μπροστά ή προς τα πίσω) θα μεταβεί στην κατάσταση με το αντίστοιχο animation, τα περισσότερα εκ των οποίων γυρίζουν αμέσως στην ίδια θέση με την ολοκλήρωση του κλιπ τους. Αυτό συμβαίνει με το animation **Hit** όταν τον χτυπάει ένα κανόνι, με το **Punch** με το οποίο πετάει τις χιονόμπαλες, με το **DeathFall** όταν πέφτει στο κενό, με το **DieRoughShort** όταν χάσει όλες τις ζωές και πεθάνει, και με το **Win** όταν βρει το θησαυρό και πανηγυρίζει. Άλλα animation διαδέχονται άλλες καταστάσεις με άλλα animation και μετά γυρίζουν στην κατάσταση που βρισκόντουσαν όταν ενεργοποιήθηκαν. Αυτό συμβαίνει όταν πεθάνει από έναν βίκινγκ, όπου πέφτει κάτω και πεθαίνει με το animation **Die**, και μετά ενεργοποιείται το **Alive** με το οποίο ανασηκώνεται από το έδαφος όταν επανέρχεται στο αρχικό του σημείο. Επίσης όταν πηδάει ενεργοποιείται μια σειρά από animations, πρώτα το **Jump** με το οποίο εκτελείται το άλμα, στη συνέχει πέφτει προς το έδαφος **JumpFall** και προσγειώνεται με το **JumpLand**. Σε οποιαδήποτε από αυτές τις καταστάσεις, μπορεί να εκτελεστεί διπλό άλμα ενεργοποιώντας το animation **DoubleJump** και έπειτα επιστρέφει στην

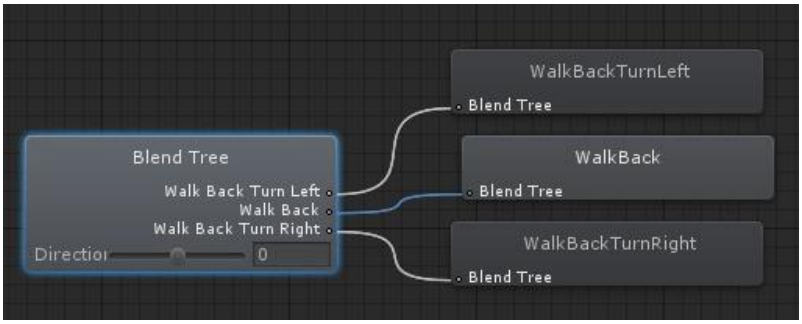


κατάσταση από την οποία ενεργοποιήθηκε και συνεχίζει η ίδια σειρά των animation μέχρι να προσγειωθεί και να επιστρέψει στην αρχική του κατάσταση. Η κάθε κατάσταση ενεργοποιείται με τις κατάλληλες τιμές των παραμέτρων που μας εισάγουν σε αυτές μέσω του κώδικα, ή με την ολοκλήρωση του χρόνου του αντίστοιχου κλιπ.

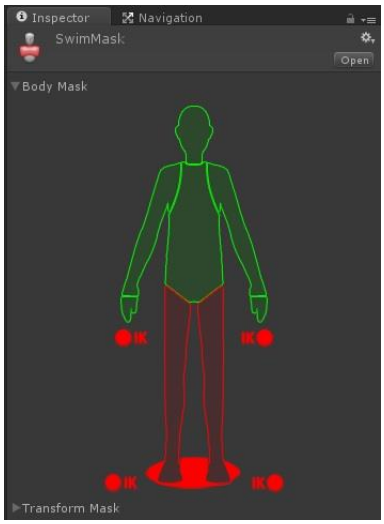
Η κατάσταση για την περιστροφή και την κίνηση του ήρωα προς τα εμπρός είναι η **Locomotion**, η οποία είναι ένα Blend Tree, δηλαδή συνδυάζει δύο ή περισσότερες παρόμοιες κινήσεις. Για την κίνηση προς τα εμπρός έχουμε αναμείξει το περπάτημα και το τρέξιμο ανάλογα με την ταχύτητα **Speed** του χαρακτήρα. Για την περιστροφή του χαρακτήρα που κλίνει προς τα αριστερά ή προς τα δεξιά καθώς γυρίζει στη διάρκεια μιας περιόδου **Direction**, έχουμε αναμείξει τα animations στροφής κατά το περπάτημα και κατά το τρέξιμο. Την ταχύτητα κίνησης η οποία καθορίζει πότε μεταβαίνουμε από το animation του περπατήματος σε αυτό του τρεξίματος, την ορίζουμε με μια μεταβλητή **v** στην οποία θέτουμε την κάθετη κίνηση του χειριστηρίου. Την κατεύθυνση που προσδιορίζει την περιστροφή κατά το τρέξιμο ή το περπάτημα την ορίζουμε με μια μεταβλητή **h** στην οποία θέτουμε την οριζόντια κίνηση του χειριστηρίου. Ο ορισμός των μεταβλητών αυτών γίνεται μέσω της **Animator.SetFloat**.

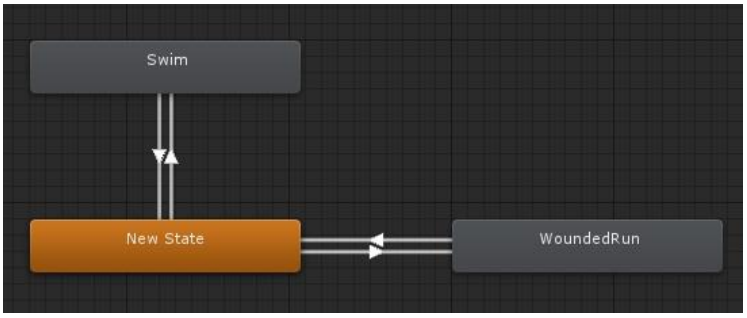


Αν ο πιγκουίνος κινείται προς τα πίσω, το animation της κίνησης ορίζεται από το Blend Tree **WalkBacks**, στο οποίο συνδυάζεται το περπάτημα με την στροφή αναλόγως τις εισόδους των παραμέτρων.



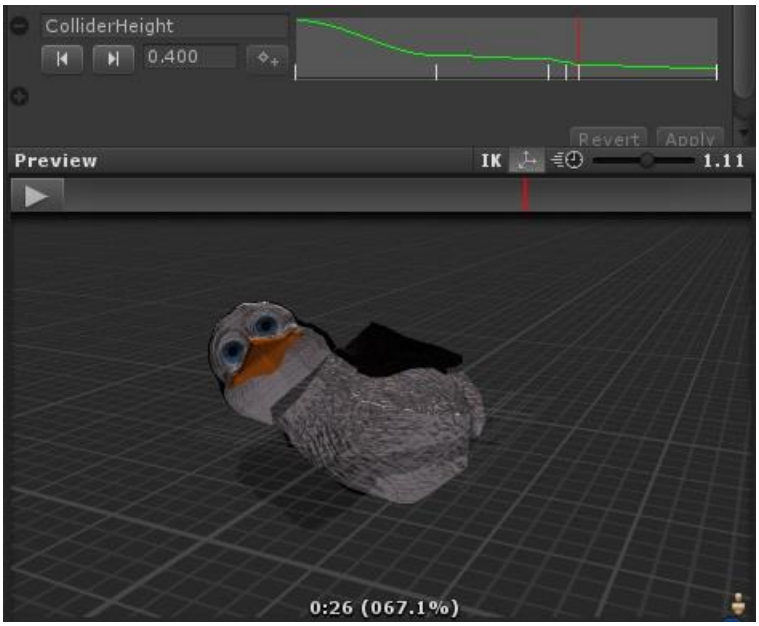
Το στρώμα του δεύτερου επιπέδου περιλαμβάνει τα animation για το κολύμπι **Swim** και του τρεξίματος ενώ είναι πληγωμένος **WoundedRun**, καθώς θέλουμε να πανωγραφεί μόνο η κίνηση του σώματος από τη μέση και πάνω, ενώ των ποδιών να παραμείνει όπως έχει. Αυτό γίνεται με τη χρήση μιας μάσκας Avatar που ενεργοποιεί τα σημεία του σώματος που μας ενδιαφέρουν.





Ορίζουμε στη μεταβλητή **currentBaseState** την κατάσταση του Animator στην οποία βρισκόμαστε στο πρώτο στρώμα, και στη μεταβλητή **layer2CurrentState** για το δεύτερο στρώμα, μέσω της **AnimatorStateInfo** η οποία μας δίνει πληροφορίες σχετικά με την τωρινή και επόμενη κατάσταση. Κάθε κατάσταση που θέλουμε να ελέγξουμε την ορίζουμε σε μια μεταβλητή μέσω της **Animator.StringToHash.** Την τωρινή κατάσταση στην οποία βρισκόμαστε σε ένα συγκεκριμένο στρώμα το γνωρίζουμε με την **Animator.GetCurrentAnimatorStateInfo**.

Σε κάποια animations θέλουμε να αλλάξουμε το μέγεθος και το ύψος του Collider του ήρωα, στη συγκεκριμένη περίπτωση του CharacterController, ώστε να μπορέσει να πλησιάσει το έδαφος, πχ όπως όταν πεθαίνει και ξαπλώνει. Ο κύλινδρος του CharacterController δεν αλλάζει θέση ή ύψος με τις κινήσεις του ήρωα, οπότε αν επιχειρήσει να ξαπλώσει ο ήρωας θα βρεθεί ξαπλωμένος στον αέρα. Για να ακουμπήσει στο έδαφος, πρέπει να αλλάξουμε το μέγεθος και τη θέση του collider. Αυτό γίνεται με τη χρήση καμπυλών του Mecanim ή με τις αλλαγές των παραμέτρων του CharecterController μέσω κώδικα. Έπειτα επαναφέρουμε τον κύλινδρο στην αρχική του θέση, διαφορετικά θα πέσει από το έδαφος ο ήρωας.



Χρησιμοποιώντας τις καμπύλες του Mecanim, μπορούμε να ορίσουμε κλειδιά-στιγμιότυπα σε σημεία που θέλουμε να αλλάξει το μέγεθος του collider και να ορίσουμε την τιμή που θέλουμε να έχει σε

κάθε κλειδί. Η καμπύλη αυτή αποθηκεύεται σε μια παράμετρο, την οποία χρησιμοποιούμε στον κώδικα. Μέσω της **Animator.GetFloat** παίρνουμε την τιμή της float παραμέτρου **ColliderHeight** και ορίζουμε αντίστοιχα το ύψος του CharacterCollider.

Για να εισαχθούμε σε κάθε κατάσταση animation πρέπει η αντίστοιχη παράμετρος που μας οδηγεί σε αυτήν να πάρει την κατάλληλη τιμή. Όσο είμαστε μέσα σε μια κατάσταση, και δεν βρισκόμαστε στο στάδιο της μετάβασης το οποίο το γνωρίζουμε μέσω της **Animator.IsInTransition**, αρχικοποιούμε την τιμή της παραμέτρου για να βγούμε από αυτήν και μπορέσουμε να ξαναεισέλθουμε όταν χρειαστεί.

Ενδεικτικά αναφέρουμε ένα μέρος του κώδικα σχετικά με τον ορισμό των animation και τις καταστάσεις τους, καθώς δεν έχουν διαφορές. Η αρχικοποίηση των παραμέτρων των καταστάσεων γίνεται με τη συνάρτηση **FixedUpdate()**, η οποία καλείται με ένα σταθερό ρυθμό πλαισίου.

```
//ANIMATION
//AnimatorStateInfo : Information about the current or next state.
private var currentBaseState : AnimatorStateInfo;        // a reference to the current state of the
                                                         //animator, used for base layer
private var layer2CurrentState : AnimatorStateInfo;      // for layer 2
//StringToHash : Generates an id parameter from a string. Ids are used for optimized setters and
//getters on parameters. these integers are references to our animator's states and are used to check
//state for various actions to occur
static var locoState : int = Animator.StringToHash("Base Layer.Locomotion");
static var jumpState : int = Animator.StringToHash("Base Layer.Jump");
static var walkBackState : int = Animator.StringToHash("Base Layer.WalkBacks");
static var deadState : int = Animator.StringToHash("Base Layer.DieRoughShort");
static var swimState : int = Animator.StringToHash("Layer2.Swim");       //second layer for swimming
static var woundedState : int = Animator.StringToHash("Layer2.WoundedRun");    //when is twice hit

public var useCurves : boolean;          // a setting for teaching purposes to show use of curves

function FixedUpdate ()
{
if(TiltControlMoveAround.canPlay)
{
        var h : float = moveTouchPad.position.x;         // setup h variable as our horizontal input axis
        var v : float= moveTouchPad.position.y;          // setup v variables as our vertical input axis
        //SetFloat : Sets the value of a float parameter. parameters (name:The name of the parameter,
        // value:The new value for the parameter)
        anim.SetFloat("Speed", v);       // set our animator's float parameter 'Speed' equal to the
                                         //vertical input axis
```

```
        anim.SetFloat("Direction", h);  // set our animator's float parameter 'Direction' equal to the
                                        //horizontal input axis
        //GetCurrentAnimatorStateInfo : Gets the current State information on a specified
        //AnimatorController layer
        currentBaseState = anim.GetCurrentAnimatorStateInfo(0);      // set our currentState variable
        // to the current state of the Base Layer (0) of animation

        if(anim.layerCount ==2)
                layer2CurrentState = anim.GetCurrentAnimatorStateInfo(1);
        // set our layer2CurrentState variable to the current state of the second Layer (1) of animation

        if (currentBaseState.nameHash == jumpState)   // if we are in the jumping state...
        {       //  ..and not still in transition..
                if(!anim.IsInTransition(0))                 //Animation Transitions define what happens
                        // when you switch from one Animation State to another. There can be only
                        // one transition active at any given time.
                {
                        anim.SetBool("Jump", false);
                }
        }
        if (currentBaseState.nameHash == deadState)
        {
                if(!anim.IsInTransition(0))
                {
                        controller.height = anim.GetFloat("ColliderHeight");   //set the collider height
                                                // to a float curve in the clip called ColliderHeight
                        anim.SetBool("Dead", false);
                }
        }
}
}
```

Η συνάρτηση **LateUpdate()** καλείται κάθε καρέ, μετά που κληθούν όλες οι λειτουργίες της Update(). Αυτό είναι χρήσιμο για να ορίσουμε μια σειρά στην εκτέλεση του κώδικα, ορίζοντας γεγονότα που θέλουμε να γίνουν μετά την Update.

Στην συνάρτηση αυτή επαναφέρουμε τον ήρωα στην αρχική του θέση και κατεύθυνση κάθε φορά που χάνει μια ζωή, είτε επειδή τον έπιασε ένας βίκινγκ, είτε επειδή χτυπήθηκε τρεις φορές από βλήμματα κανονιών, το οποίο σηματοδοτείται από την μεταβλητή **dead**. Αν είναι ίση με μηδέν αρχικοποιούμε κάθε κατάστασή του που αφορά το μέγεθος, τα κινούμενα σχέδια, την κίνηση και μηδενίζεται ο αριθμός των χτυπημάτων που μπορεί να υποστεί πριν χάσει μια ζωή. Επίσης ενεργοποιείται ένα animation με το οποίο σηκώνεται από το έδαφος για να συνεχίσει το παιχνίδι και αναπαράγεται ένας ήχος που ειδοποιεί για την επαναφορά του. Μέχρι να ολοκληρωθεί το animation με το οποίο σηκώνεται όρθιος, δεν μπορεί να

προχωρήσει καθώς είναι απενεργοποιημένη η κίνηση. Μαζί με αυτόν επανέρχονται στην αρχική τους θέση η κύρια κάμερα και τα πιγκουινάκια που τον ακολουθούν.

Κάθε φορά που πεθαίνει ο ήρωας και χάσει μια ζωή δονείται το κινητό μέσω της συνάρτησης **Handheld.Vibrate()**. Αν η συσκευή που αναπαράγει το παιχνίδι δεν την υποστηρίζει, απλά αγνοείται.

Σε κάθε επαναφορά του ήρωα στο αρχικό σημείο, πριν σηκωθεί από το έδαφος, ελέγχουμε αν του έχουν μείνει και άλλες ζωές για να μπορέσει να συνεχίσει το παιχνίδι, διαφορετικά πεθαίνει ενεργοποιώντας ένα animation με το οποίο πέφτει στο έδαφος και μένει ξαπλωμένος, αλλάζοντας το μέγεθος του collider.

Επίσης ελέγχουμε αν ο πιγκουίνος έχει χτυπηθεί από κάποιο κανόνι, το οποίο σηματοδοτείται από την μεταβλητή **gotHit**. Αν είναι αληθής περιστρέφεται με κάποια ταχύτητα γύρω από τον εαυτό του και ενεργοποιείται το animation στο οποίο πληγώνεται. Ο ήρωας περιστρέφεται για να είναι πιο δύσκολο για το χρήστη να προσανατολιστεί και να αντιμετωπίσει τον εχθρό. Για την περιστροφή ορίζουμε στη μεταβλητή **tumbleSpeed** μια τιμή για την ταχύτητα περιστροφής, στην μεταβλητή **decreaseTime** μια τιμή με την οποία θα μειώνουμε σιγά σιγά τον χρόνο περιστροφής μέχρι να σταματήσει να κινείται ώστε να φτιάξουμε το animation μέσω του κώδικα, και σε έναν πίνακα **backup** κρατάμε τις αρχικές τιμές των μεταβλητών αυτών γιατί αυξάνονται με το χρόνο κατά την περιστροφή.

Αν έχει χτυπηθεί, και αν έχει σταματήσει να περιστρέφεται, δηλαδή αν η ταχύτητα περιστροφής του είναι μικρότερη του 1, αρχικοποιούμε τις παραπάνω μεταβλητές μέσω του πίνακα backup και θέτουμε ψευδή την gotHit για να μπορέσουμε να ξαναχτυπηθούμε. Αν όμως η ταχήτητα περιστροφής είναι μεγαλύτερη του 1, τότε περιστρέφεται και ορίζουμε την περιστροφή μέσω της transform.Rotate εισάγοντας στον άξονα y, την τιμή της tumbleSpeed πολλαπλασιασμένη με την Time.deltaTime για να έχουμε κίνηση ανά δευτερόλεπτο. Κατά την περιστροφή αφαιρούμε κάθε φορά από την tumbleSpeed το χρόνο decreaseTime με τον οποίο θέλουμε να σταματήσει σιγά σιγά η περιστροφή, ο οποίος κάθε φορά διπλασιάζεται.

```
//Getting hit
var tumbleSpeed = 800;          //when we get hit we spin the character around
var decreaseTime = 0.01;        //it slowly stops the rotation to start moving
static var gotHit = false;                  //globaly accessable by all scripts
private var backup = [tumbleSpeed, decreaseTime, decayTime]; //array, we need a backup from these
                                                    //variables because they increase by time
function LateUpdate()  //basically the same with the Update() but it is called after the Update
                       //function so this way our collision script doesn't interfere with the moving
                       //collider around script
{
if(TiltControlMoveAround.canPlay)
```

```
{
        if(gotHit)        //we check if we got hit
        {
                anim.SetBool("GotHit", true);   //GOT HIT
                if(tumbleSpeed < 1)      //if we stop spinning around we need to set everything back to
                {                        //default (safety net)
                        //we're not hit anymore -> reset and get back in the game!
                        tumbleSpeed = backup[0];
                        decreaseTime = backup[1];
                        decayTime = backup[2];
                        gotHit = false;             //because we have already got hit
                }
                else      //if we are still spinning around then we need to rotate the character
                {
                        //we're hit! Spin our character around
                        transform.Rotate(0,tumbleSpeed * Time.deltaTime,0, Space.World);
                        //rotate in the world space using only the y axis
                        tumbleSpeed = tumbleSpeed - decreaseTime;//slowly stops rotating over time
                        decreaseTime += decreaseTime;           //it makes the decreaseTime bigger,
                      //so each time it goes faster down to zero and that gives an illusion of animation
                }
        }
}
}
```

Επιπλέον, στη συνάρτηση LateUpdate(), ελέγχουμε στο τέλος αν κέρδισε ο ήρωας και αναλόγως τον κόσμο και την πίστα που βρισκόμαστε κάθε φορά, αποθηκεύουμε όλα τα δεδομένα που χρειάζεται να εμφανίσουμε στο μενού των πιστών. Μέσω της **PlayerPrefs** αποθηκεύουμε σε μεταβλητές ότι η συγκεκριμένη πίστα τερματίστηκε επιτυχώς, τον μέγιστο αριθμό των νομισμάτων και των ψαριών που συλλέχθηκαν και τα αποθηκεύουμε μέσω της **PlayerPrefs.Save()**.

Για να κρατήσουμε τη μεγαλύτερη τιμή, αποθηκεύουμε σε μία μεταβλητή το δεδομένο που θέλουμε, πχ τον αριθμό των νομισμάτων στη μεταβλητή **coins**, με την **PlayerPrefs.SetInt**. Κάθε φορά που τερματίζεται η ίδια πίστα ανακαλούμε τα δεδομένα αυτά μέσω της **PlayerPrefs.GetInt** και τα συγκρίνουμε με την τωρινή μεταβλητή coins η οποία περιλαμβάνει τον αριθμό των νομισμάτων του τωρινού παιχνιδιού. Αν τα δεδομένα που είχαμε αποθηκεύσει έχουν μικρότερη τιμή από την τωρινή coins, πετύχαμε μεγαλύτερο σκορ, οπότε αποθηκεύουμε τα καινούργια δεδομένα στη θέση των παλιών μέσω της PlayerPrefs.SetInt. Παρακάτω αναφέρουμε την αποθήκευση δεδομένων για την πρώτη πίστα της SnowLand, και ακολουθούμε την ίδια μέθοδο και για τις υπόλοιπες πίστες.

```
//WIN
if(hasWon)                //we save the data for each level
{
        //SNOWLAND
        if(snowLand == 1)
        {
                PlayerPrefs.SetInt("WinMove1", 1);
                //it will automatically check if this value is more then the previous and it will save
                //automatically on save points.
                 if (PlayerPrefs.GetInt("Coin1")<coins)
                {
                        PlayerPrefs.SetInt("Coin1", coins);
                }
                if(PlayerPrefs.GetInt("FishBonus1")<FishControl.fishCounter)
                {       //save the number of fishes
                        PlayerPrefs.SetInt("FishBonus1", FishControl.fishCounter);
                }
                //to find which of the fishes we have collected
                if(Fishy1)      //if the first fish has been collected
                {
                        PlayerPrefs.SetInt("1GotFish1",1);      //make intense its colour
                }
                else
                {
                        PlayerPrefs.SetInt("1GotFish1",0);      //otherwise leave it with a fade colour
                }
                if(Fishy2)
                {
                        PlayerPrefs.SetInt("1GotFish2",1);
                }
                else
                {
                        PlayerPrefs.SetInt("1GotFish2",0);
                }
                if(Fishy3)
                {
                        PlayerPrefs.SetInt("1GotFish3",1);
                }
                else
                {
                        PlayerPrefs.SetInt("1GotFish3",0);
                }
                PlayerPrefs.Save();     //save all data to disk
        }
```

```
}
```

Η συνάρτηση **Awake()** καλείται όταν το script φορτώνεται, χρησιμοποιείται για την αρχικοποίηση μεταβλητών ή οποιαδήποτε κατάσταση του παιχνιδιού πριν από την έναρξη του παιχνιδιού. Καλείται μόνο μία φορά κατά τη διάρκεια του κώδικα και μετά που όλα τα αντικείμενα έχουν αρχικοποιηθεί, έτσι ώστε να μπορέσουμε να τα χρησιμοποιήσουμε.

Τη χρησιμοποιήσαμε για να υπολογίζουμε σε κάθε έναρξη της πίστας τον αριθμό των κανονιών και των βίκινγκ, για να τους εμφανίσουμε στην οθόνη δίπλα σε ένα εικονίδιο που τους αναπαριστά για να ξέρει ο χρήστης τους εχθρούς που πρέπει να αντιμετωπίσει. Για κάθε περίπτωση εχθρού, αποθηκεύουμε σε μια μεταβλητή των αριθμό των GameObjects που βρέθηκαν κατά την έναρξη του παιχνιδιού με την ετικέτα (tag) που θέλουμε μέσω της **GameObject.FindGameObjectsWithTag**, και παίρνουμε το μήκος της μέσω της **GameObject.Length**, το οποίο αντιπροσωπεύει τον αριθμό των αντικειμένων αυτών.

```
//we need to count how many cannones exist in the game
static var CANNONES = 0;       static var VIKINGS = 0;
var cannoneEnemies : GameObject[];   var vikings : GameObject[];
function Awake()
{
        cannoneEnemies = GameObject.FindGameObjectsWithTag("Cannone");
        CANNONES = cannoneEnemies.Length;
        vikings = GameObject.FindGameObjectsWithTag("Viking");
        VIKINGS = vikings.Length;
}
```

Η συνάρτηση **OnTriggerEnter()** καλείται όταν ο Collider εισέρχεται σε trigger. Το μήνυμα αποστέλλεται στον επιταχυντή trigger και στο rigidbody (ή στον επιταχυντή, εφόσον δεν υπάρχει rigidbody) που αγγίζει το trigger. Κάθε φορά δηλαδή που ο επιταχυντής του πιγκουίνου, ο CharacterCollider ή ο SphereCollider που βρίσκεται στο μαγνήτη-παιδί του πιγκουίνου, έρθει σε επαφή με έναν άλλον επιταχυντή, η συνάρτηση αυτή επιστρέφει το αντικείμενο με του οποίου τον επιταχυντή βρέθηκε σε επαφή. Διαβάζοντας το όνομα ή την ετικέτα του αντικειμένου το οποίο επιστρέφει η συνάρτηση γνωρίζουμε με ποιο GameObject αλληλεπιδρά ο ήρωας κάθε στιγμή.

Στη συνάρτηση αυτή υπολογίζουμε τον αριθμό των νομισμάτων που συλλέγει ο ήρωας, κάθε φορά που έρχεται σε επαφή με το Sphere Collider του νομίσματος αυξάνουμε τη μεταβλητή αριθμού νομισμάτων **coins** κατά ένα. Σε κάθε GameObject νομίσματος έχουμε ορίσει την ετικέται **Coin** για να μπορέσουμε να τα αναγνωρίσουμε.

Αν συλλέξει το μαγικό σμαράγδι με την ετικέτα **PowerUp**, τον μετατοπίζουμε λίγο πιο ψηλά στον άξονα των y για να μπορέσουμε να διπλασιάσουμε το μέγεθός του με την

συνάρτηση **DoubleSize()** και να μην πέσει κάτω από το έδαφος, και τον κάνουμε αθάνατο θέτοντας αληθή την **cannotDie**.

```
function OnTriggerEnter( hit : Collider )        // this function is called whenever we hit something
{                                                 // and it tells us in "hit" variable what we are hitting
        if(TiltControlMoveAround.canPlay)
        {       //COIN
                if (hit.tag == "Coin")     //if the penguin collects coins we make a sound
                {
                        coins++;
                }
                //POWER UP
                if(hit.tag == "PowerUp")
                {
                        gameObject.transform.position.y = 3.5;
                        DoubleSize();
                        cannotDie = true;//he cannot die by enemies, only if he fell off the ground
                }
        }
}
```

Η συνάρτηση DoubleSize του διπλασιάζει το μέγεθος με την **Transform.localScale**, προσθέτοντας στο τωρινό μέγεθος του ήρωα, το οποίο ισούται σε όλους τους άξονες με 1, ένα τρισδιάστατο διάνυσμα με την τιμή 1. Ο διπλασιασμός διαρκεί για οχτώ δευτερόλεπτα με την **WaitForSeconds()** η οποία αναστέλλει την εκτέλεση της υπορουτίνας (coroutine) για την δεδομένη ποσότητα δευτερολέπτων. Μετά το χρόνο αυτό επαναφέρουμε το αρχικό μέγεθος και θέτουμε την cannotDie ίση με false για να μπορεί να πεθάνει πάλι.

```
var powerupTime : int = 8;
function DoubleSize()
{
        gameObject.transform.localScale += Vector3(1,1,1);      //he will have double size
        yield WaitForSeconds(powerupTime);
        gameObject.transform.localScale = Vector3(1,1,1);
        cannotDie = false;                                      //he is vulnerable again
}
```

Αν έρθει σε επαφή με έναν βίκινγκ, αν δεν έχει πάρει το μαγικό σμαράγδι, η συσκευή δονείται με την **Handheld.Vibrate()** για να καταλάβει ο χρήστης ότι τον έπιασε ο εχθρός. Παγώνουμε την κίνησή του, ενεργοποιούμε το animation με το οποίο πεθαίνει και περιμένουμε μέχρι να ολοκληρωθεί για να τον επαναφέρουμε στο αρχικό του σημείο θέτοντας την dead αληθή.

Αν πέσει στο κενό, θα χάσει μια ζωή οπότε δονείται η συσκευή καθώς πέφτει. Παγώνουμε την κίνησή του, χάνει μια ζωή και επανέρχεται στο αρχικό του σημείο ενεργοποιώντας την

dead. Αν δεν έχει χάσει ακόμα, ενεργοποιούμε ένα animation για να σηκωθεί από το έδαφος όταν βρεθεί στο αρχικό του σημείο, αλλιώς πεθαίνει.

Αν χτυπηθεί από βλήμμα κανονιού και αν δεν είναι υπό την επίδραση του μαγικού σμαραγδιού, αυξάνουμε τα χτυπήματά του κατά ένα και καταστρέφουμε το βλήμμα. Αν είναι το πρώτο χτύπημα δημιουργούμε μία έκρηξη στο σημείο που βρίσκεται το δεύτερο πιγκουινάκι το οποίο εξαφανίζεται (όπως δείχνουμε παρακάτω), αν είναι το δεύτερο χτύπημα η έκρηξη γίνεται στο σημείο που είναι το πρώτο πιγκουινάκι, και αν είναι το τρίτο χτύπημα στη θέση του πιγκουίνου πριν πεθάνει. Η έκρηξη διαρκεί λίγα δευτερόλεπτα και τη δημιουργούμε μέσω της Instantiate. Στο τρίτο χτύπημα θα χάσει μια ζωή, οπότε ενεργοποιείται η δόνηση της συσκευής.

```
//DIEING from Viking
if(hit.gameObject.tag == "Viking" )
{
        if(!cannotDie)                                  //he can die
        {
                Froze=true;
                if(isHit == false)
                {
                        Handheld.Vibrate ();            // vibrate the device
                        isHit = true;
                        anim.SetBool("Die", true);      //die animation
                        yield WaitForSeconds(1.5);      //assuming it takes 1.5 seconds to play
                                                        //the animation
                        dead = true;
                }
        }
}
//FALLING. we need to check what we are hitting
if(hit.gameObject.tag == "fallout2")            //it means we have fall off the platform
{
        Handheld.Vibrate ();                    // vibrate the device
        Froze = true;                           //the player is frozen, he cannot move
        anim.SetBool("DeadFall", true);         //die animation
        if(HealthControl.LIVES == 3 || HealthControl.LIVES == 2)        //this is not our last life
        {
                yield WaitForSeconds(1);
                anim.SetBool("GetUp", true);
        }
        else if(HealthControl.LIVES == 1)
        {
                anim.SetBool("GetUp", false);
```

```
        }
}
//GOT HIT
if(hit.gameObject.tag == "enemyProjectile")     //if we get hit by the enemy
{
if(!cannotDie)                                  //he can die
{
        gotHit = true;
        HealthControl.HITS += 1;        //we need to count the hits
        Destroy(hit.gameObject);        //we need to destroy the enemyProjectile after it hit us
        if(HealthControl.HITS == 1)     //with one hit we explode the second small penguin
        {
                for (var i : int = 0;i < 10; i++) //the explosion last for some seconds
                {
                        var boom1 = Instantiate(penguinExplosion,
                          gameObject.Find("littlePenguin2").transform.position, Quaternion.identity);
                }//the Instantiate clones the particles of the exolosion at the smallPenguin2 position
        }
        if(HealthControl.HITS == 2)     //with two hits we explode the first small penguin
        {
                for (var j : int = 0;j < 10; j++)
                {
                        var boom2 = Instantiate(penguinExplosion,
                          gameObject.Find("littlePenguin1").transform.position, Quaternion.identity);
                }
        }
        if(HealthControl.HITS == 3)     //with three hits we explode the Main penguin
        {
                for (var k : int = 0;k < 10; k++)
                {
                        Handheld.Vibrate ();                            // vibrate the device
                        var boom3 = Instantiate(penguinExplosion, transform.position,
                                        Quaternion.identity);
                }
                dead = true;
        }
}
}
}
}
```

Για κάθε ψαράκι μπόνους που συλλέγει προστίθονται οι βαθμοί που αντιστοιχούν στο καθένα (100, 200, 300) σε μια μεταβλητή **FishBonus** για το σκορ, και αναλόγως το ψάρι που σύλλεξε σηματοδοτούμε τη μεταβλητή **Fishy1**, **Fishy2**, και **Fishy3** αντίστοιχα για να ξέρουμε ποιο ψάρι θα πάρει έντονο χρώμα στο σκορ.

Αν ακουμπήσει το θησαυρό, ελέγχουμε αν έχει τερματίσει επιτυχώς (σε πίστες δυσκολότερου επιπέδου πρέπει να έχει σκοτώσει όλους τους εχθρούς πρώτα), και ενεργοποιούμε τη μουσική τερματισμού, το animation πανηγυρισμού και θέτουμε αληθή τη μεταβλητή νίκης της πίστας που χρησιμεύει στο ξεκλείδωμα των κόσμων.

Αν ακουμπήσει το νερό ενεργοποιείται το animation με το οποίο κολυμπάει και αναπαράγεται ένας ήχος πλατσουρίσματος.

Η συνάρτηση **OnTriggerExit()** καλείται όταν ο Collider έχει σταματήσει να αγγίζει το trigger κάποιου αντικειμένου, και μας επιστρέφει το αντικείμενο αυτό. Όταν σταματάει ο ήρωας να αγγίζει το νερό, σταματάει το animation για το κολύμπι και ο αντίστοιχος ήχος.

```
function OnTriggerExit(other : Collider)
{
	if(other.gameObject.tag == "Water")
	{
		anim.SetBool("CanSwim", false);		//stop swimming
		isSwimming = false;
	}
}
```

Στο τέλος του παιχνιδιού απενεργοποιούμε το χειριστήριο.

```
function OnEndGame()
{
	moveTouchPad.Disable();	// Disable joystick when the game ends
}
```

Όταν ο χρήστης επιλέξει το κατάλληλο κουμπί, ο ήρωας κοιτάει προς το θησαυρό ή τη φάλαινα γυρίζοντας το κεφάλι του. Ο έλεγχος του κεφαλιού και των ματιών του γίνεται μέσω του script **LayserEyesMobile.js**.

Η **LineRenderer** χρησιμοποιείται για να δημιουργήσουμε ακτίνες λέιζερ ως ελεύθερες κυμαινόμενες γραμμές στον 3D χώρο. Ορίζουμε τρεις μεταβλητές LineRenderer, τη **laserPrefab** για να θέσουμε το prefab της ακτίνας λέιζερ που δημιουργήσαμε, και δύο για τη θέση με την οποία ξεκινάει το λέιζερ από το κάθε μάτι του πιγκουίνου (**laserL** και **laserL**). Για να γνωρίζουμε το σημείο που βρίσκονται τα μάτια του, δημιουργήσαμε ένα κενό GameObject για το κάθε μάτι, θέτοντάς το σε μία **Transform** μεταβλητή (**EyeL** και **EyeR**). Στη μεταβλητή **guider** θέτουμε το GameObjetc που θέλουμε να κοιτάξει ο πιγκουίνος, και με μια float μεταβλητή **lookWeight** ελέγχουμε πότε θα γίνεται η κίνηση του κεφαλιού.

Στη συνάρτηση Start() γίνεται ο ορισμός και η αρχικοποιήση των τιμών, ενώ στην FixedUpdate() ορίζουμε τη τιμή της lookWeight για να καθορίσουμε αν θα περιστραφεί το κεφάλι του ή όχι.

Με την **Animator.SetLookAtWeight()** ρυθμίζουμε το βάρος lookWeight το οποίο χρησιμοποιούμε με IK animation για το κεφάλι ώστε να κοιτάξει ο πιγκουίνος. Όσο ο χρήστης πιέζει το κουμπί για να κοιτάξει το θησαυρό ή τη φάλαινα, η μεταβλητή **seeWhaleNow** είναι αληθής. Όσο παραμένει αληθής, ρυθμίζουμε τον πιγκουίνο να κοιτάει προς τη θέση του αντικειμένου guider με την **Animator.SetLookAtPosition()**. Επίσης με την **Mathf.Lerp()** ρυθμίζουμε την τιμή της lookWeight, δίνοντάς της μια float τιμή μεταξύ του 0 και 1 με την πάροδο του χρόνου Time.deltaTime τον οποίο έχουμε πολλαπλασιάσει με μια τιμή για να έχουμε πιο ομαλή μετάβαση. Αν ο χρήστης σταματήσει να πατάει το κουμπί, θέτουμε την lookWeight ίση με μηδέν για να επανέλθει το κεφάλι στην αρχική του θέση.

```
function FixedUpdate ()
{
	//SetLookAtWeight : Set look at weights.
	anim.SetLookAtWeight(lookWeight);  // set the Look At Weight - amount to use look at IK
	                                   // vs using the head's animation
	// LOOK AT ENEMY
	if( SeeWhaleTouchButton.seeWhaleNow)
	{
	// ...set a position to look at with the head, and use Lerp to smooth the look weight from
	//animation to IK
		anim.SetLookAtPosition(guider.position);	//SetLookAtPosition : Sets the look at
		                                        //position(The position to lookAt).
		lookWeight = Mathf.Lerp(lookWeight,1f,Time.deltaTime*lookSmoother);
	}
	else	// else, return to using animation for the head by lerping back to 0 for look at weight
	{
		lookWeight = Mathf.Lerp(lookWeight,0f,Time.deltaTime*lookSmoother);
	}
}
```

Στη συνάρτηση Update() ενεργοποιούμε το λέιζερ αναλόγως την τιμή της lookWeight. Αν η τιμή της είναι μεγαλύτερη από το 0,9 και δεν έχει ενεργοποιηθεί το λέιζερ ακόμη, τότε κλωνοποιούμε δύο λέιζερ με την Instantiate ως LineRenderer, ενημερώνουμε ότι ενεργοποιήθηκε το λέιζερ θέτοντας αληθή τη μεταβλητή **shot** και αναπαράγουμε έναν αντίστοιχο ήχο λέιζερ. Αν η τιμή της lookWeight είναι μικρότερη του 0,9 επαναφέρουμε την αρχική κατάσταση καταστρέφοντας τα λείζερ με την **Object.Destroy**, αρχικοποιούμε την shot και απενεργοποιούμε τον ήχο. Όσο υπάρχει το λέιζερ καθορίζουμε τη θέση από την οποία θα ξεκινάνε τα λέιζερ, και την κατεύθυνση στην οποία θα πηγαίνουν με την **LineRenderer.SetPosition()**.

```
function Update ()
{
	//LASER EYES
```

```
        if(lookWeight >= 0.9f && !shot) // if the look weight has been increased to 0.9, and we have
                                        // not yet shot..
        {
                // instantiate our two lasers
                laserL = Instantiate(laserPrefab) as LineRenderer;
                laserR = Instantiate(laserPrefab) as LineRenderer;
                shot = true;                            // register that we have shot once
                audio.PlayOneShot(LaserSound);          // play the laser beam effect
        }
        else if(lookWeight < 0.9f)      // if the look weight returns to normal
        {
                // Destroy the laser objects
                Destroy(laserL);
                Destroy(laserR);
                shot = false;           // reset the shot toggle
                audio.Stop();           // stop audio playback
        }
        if(laserL != null)                      // if our laser line renderer objects exist..
        {
        // set positions for our line renderer objects to start at the eyes and end at the enemy position,
        //registered in the bot control script
                laserL.SetPosition(0, EyeL.position);   //SetPosition : Set the position of the vertex in
                                                        //the line.
                laserL.SetPosition(1, guider.position);
                laserR.SetPosition(0, EyeR.position);
                laserR.SetPosition(1, guider.position);
        }
}
```

5.5.3 Υλοποίηση κουμπιών αφής

Τα κουμπιά άλματος, πυροβολισμού, κοιτάγματος θυσαυρού ή φάλαινας, και κοιτάγματος του κόσμου από ψηλά είναι το καθένα ένα GUITexture GameObject με την εικόνα του αντίστοιχου κουμπιού και ένα script λειτουργίας του.

Η κλάση για την λειτουργία του κουμπιού άλματος είναι η **TouchJump**. Στη συνάρτηση Start() ενεργοποιούμε την εμφάνιση της εικόνας του κουμπιού. Ορίζουμε μια στατική Boolean μεταβλητή **jumpNow** για να ενημερώνουμε πότε πρέπει ο πιγκουίνος να πηδήξει και να μπορούν άλλες κλάσεις να έχουν πρόσβαση σε αυτήν. Στη συνάρτηση Update() απενεργοποιούμε την εικόνα του αν τελειώσει το παιχνίδι, είτε κερδίζοντας είτε χάνοντας, ή αν ο χρήστης επιλέξει την παύση του παιχνιδιού, είτε πιέζοντας δύο δάχτυλα ταυτόχρονα στην οθόνη είτε επιλέγοντας το κουμπί παύσης, ή αν η συσκευή δεν είναι τοποθετημένη στη σωστή θέση. Αν κανένα από τα παραπάνω δεν ισχύει τότε η εικόνα του κουμπιού είναι ενεργοποιημένη και ελέγχουμε αν επιλεχθεί το συγκεκριμένο κουμπί.

Ελέγχουμε αν υπάρχει έστω και ένα άγγιγμα στην οθόνη με την **Input.touchCount** η οποία μας δίνει τον αριθμό των αγγιγμάτων που δεν αλλάζουν κατά τη διάρκεια του καρέ. Για κάθε άγγιγμα ελέγχουμε την κατάσταση στην οποία βρίσκεται με την **Input.touches** η οποία μας ενημερώνει για τις καταστάσεις των αγγιγμάτων κατά τη διάρκεια του τελευταίου καρέ. Ελέχουμε αν ο χρήστης μόλις άγγιξε την οθόνη με την **TouchPhase.Began** και αν ακούμπησε την περιοχή της θέσης του κουμπιού άλματος, την οποία βρίσκουμε με την **GUIElement.HitTest()** που είναι ένα σημείο στην οθόνη στο εσωτερικό του στοιχείου και επιστρέφει true αν η θέση στην οθόνη περιέχεται σε αυτό το GUIElement, τότε θέτουμε αληθή τη μεταβλητή jumpNow για να πηδήξει ο πιγκουίνος μέσω της Update() στην AndroidMoveAround κλάση. Αλλιώς η jumpNow είναι ψευδής και ο πιγκουίνος δεν πηδάει.

```
#pragma strict
function Start () {
	guiTexture.enabled = true;
}

static var jumpNow : boolean = false;
function Update () {
	//disable the button when we die or we have pussed pause
	if( HealthControl.GameOver == true || HealthControl.WinGame == true ||
		AndroidMoveAround.PauseFingers == true || PauseTouch.pauseNow == true ||
		TiltControlMoveAround.canPlay == false)
	{
		guiTexture.enabled = false;
	}
	if(HealthControl.GameOver == false && HealthControl.WinGame == false &&
		AndroidMoveAround.PauseFingers == false && PauseTouch.pauseNow == false &&
		TiltControlMoveAround.canPlay == true)
	{
		guiTexture.enabled = true;
		if (Input.touchCount>0)
		{
			for (var touch : Touch in Input.touches)
			{
				if (touch.phase == TouchPhase.Began &&
					guiTexture.HitTest(touch.position))
				{
					jumpNow = true;		//this can handle many touches :D
				}
				else
				{
					jumpNow = false;
				}
```

```
                        }
                }
        }
}
```

Ίδια διαδικασία ακολουθείται για το κουμπί πυροβολισμού στο script **TouchButtons.js** θέτοντας αντίστοιχα αληθή ή ψευδή τη μεταβλητή **shootNow** η οποία ενημερώνει την Update της AndroidMoveAround για το πότε πραγματοποιείται πυροβολισμός. Και τα δύο αυτά κουμπιά λειτουργούν με τον ίδιο τρόπο, καθώς μας ενδιαφέρει μόνο το πότε τα πατάει ο χρήστης και πόσες φορές, και όχι για πόσο χρόνο. Για αυτό χρησιμοποιήσαμε στο κουμπί άλματος και πυροβολισμού την TouchPhase.Began.

Στα κουμπιά όμως κοιτάγματος του θυσαυρού ή της φάλαινας, και του κόσμου από ψηλά, μας ενδιαφέρει η χρονική διάρκεια της επιλογής του κουμπιού καθώς τόση ώρα διαρκεί και η αντίστοιχη λειτουργία. Για αυτό το λόγο αντί την TouchPhase.Began, χρησιμοποιούμε την **TouchPhase.Stationary**, η οποία μας ενημερώνει αν ένα δάχτυλο αγγίζει την οθόνη και δεν έχει μετακινηθεί. Οι υπόλοιποι έλεγχοι που πραγματοποιούνται είναι ίδιοι με τα άλλα δύο κουμπιά με διαφορετικές μεταβλητές. Στον κώδικα **SeeWhaleTouchButton.js** θέτουμε αληθή τη μεταβλητή **seeWhaleNow** όσο είναι επιλεγμένο το κουμπί για το κοίταγμα της φάλαινας ή του θυσαυρού, για να ενεργοποιηθεί η αντίστοιχη λειτουργία κοιτάγματος από την LayserEyesMobile. Στον κώδικα **SeeWorldTouch.js** θέτουμε αληθή τη μεταβλητή **seeWorldNow** όσο επιλεγμένο το κουμπί για το κοίταγμα του κόσμου από ψηλά, για να ενεργοποιηθεί η αντίστοιχη λειτουργία από την AndroidMoveAround.

```
if (Input.touchCount>0)
{
        for (var touch : Touch in Input.touches)
        {
                if (touch.phase == TouchPhase.Stationary && guiTexture.HitTest(touch.position))
                {
                        seeWhaleNow = true;
                }
                else
                {
                        seeWhaleNow = false;
                }
        }
}
```

Για το κουμπί παύσης του παιχνιδιού, δεν μας ενδιαφέρει η φάση αγγίγματος στην οποία βρισκόμαστε καθώς δεν πραγματοποιείται καμία ενέργεια κατά τη διάρκεια του παιχνιδιού, απλά διακόπτεται. Επομένως, στον κώδικα **PauseTouch.js**, μας ενδιαφέρει απλά να ελέγξουμε αν πατήθηκε το συγκεκριμένο κουμπί, δηλαδή αν το δάχτυλο ακούμπησε τη θέση στην οποία

βρίσκεται το κουμπί. Αν το ακούμπησε, θέτουμε αληθή τη μεταβλητή **pauseNow** για να σημάνουμε την παύση του παιχνιδιού και σταματάμε το χρόνο με την **Time.timeScale.**

```
if (Input.touchCount>0)
{
	for (var touch : Touch in Input.touches)
	{
		if (guiTexture.HitTest(touch.position))
		{
			Time.timeScale = 0;
			pauseNow = true;
		}
	}
}
```

5.5.4 Υλοποίηση κύριας κάμερας

Η κύρια κάμερα βρίσκεται πίσω και πάνω από το κεφάλι του ώστε να τον ακολουθεί παντού στο παιχνίδι και να βλέπει ο χρήστης τον ήρωα αλλά και το περιβάλλον μπροστά από αυτόν (third person shooter). Η κάμερα αποτελείται από τα συστατικά GUILayer για να μπορεί να βλέπει τα GUI του παιχνιδιού, από Flare Layer για να μπορούμε να βλέπουμε λάμψεις, τον Audio Listener ο οποίος μπορεί να είναι μόνο ένας και χρησιμοποιείται για να ακούμε όλους τους ήχους στο παιχνίδι, και τέλος το script **CameraControl** που καθορίζει την κίνησή της στο παιχνίδι.

Η κίνηση της κάμερας είναι ομαλή και ακολουθεί τον πιγκουίνο με μια μικρή καθυστέρηση, για αυτό την ελέγχουμε μέσω της συνάρτηση LateUpdate(). Όσο ο χρήστης πιέζει το κουμπί κοιτάγματος θυσαυρού ή φάλαινας και η seeWhaleNow είναι θετική αλλάζουμε τη θέση της κάμερας **transform.position** και την βάζουμε κοντά και πίσω από το κεφάλι του πιγκουίνου. Για να βρούμε τη θέση αυτή, τοποθετούμε ένα κενό GameObject στο σημείο που θέλουμε του οποίου η θέση μεταβιβάζεται στην κάμερα μέσω της **Vector3.Lerp** ομαλά με την πάροδο του χρόνο ανά δευτερόλεπτο, καθώς πολλαπλασιάζουμε την Time.deltaTime με μια τιμή ομαλοποίησης. Με τον ίδιο τρόπο αλλάζουμε και την κατεύθυνση της κάμερας **transform.forward** ώστε να κοιτάει προς το στόχο. Το ίδιο γίνεται όταν ο ήρωας πεθαίνει και έχει μηδέν ζωές, η κάμερα πηγαίνει μπροστά του και κοιτάει προς αυτόν. Στη θέση εκείνη έχουμε άλλο ένα κενό GameObject του οποίου τη θέση και κατεύθυνση την ορίζουμε στην κάμερα με την Vector3.Lerp. Επίσης, όσο ο χρήστης πιέζει το κουμπί για να κοιτάξει το κόσμο από ψηλά και η seeWorldNow είναι θετική, και η κάμερα παίρνει τη θέση και την κατεύθυνση ενός κενού GameObject που έχουμε τοποθετήσει ψηλά και πάνω από τον πιγκουίνο για να βλέπει καλύτερα.

Αν δεν ισχύει καμία από τις παραπάνω περιπτώσεις, τότε η κάμερα απλά ακολουθεί ομαλά τον ήρωα από πίσω. Στη Transform μεταβλητή **target** θέτουμε τον στόχο που ακολουθεί η κάμερα, δηλαδή τον πιγκουίνο. Ορίζουμε στη μεταβλητή **wantedRotationAngle** την περιστροφή ως Euler γωνίες σε μοίρες του ήρωα στον άξονα y, και στη **wantedHeight** τη θέση του στον άξονα των y στην οποία προσθέτουμε μια τιμή για να πάρουμε το ύψος που επιθυμούμε να έχει η κάμερα που βρίσκεται πάνω από αυτόν. Επίσης ορίζουμε στην **currentRotationAngle** την περιστροφή ως Euler γωνίες και στην **currentHeight** τη θέση της της κάμερας στον άξονα των y. Έπειτα θέτουμε την περιστροφή της κάμερας currentRotationAngle ίση με την επιθυμητή περιστροφή του στόχου wantedRotationAngle με μία ομαλή μετάβαση κατά τη διάρκεια του χρόνο, μέσω της **Mathf.LerpAngle** η οποία είναι ίδια με τη Lerp αλλά φροντίζει να παρεμβάλονται σωστά οι τιμές στους 360 βαθμούς. Το ύψος της κάμερας currentHeight το αντικαθιστούμε με το επιθυμητό ύψος wantedHeight ομαλά με την πάροδο του χρόνου μέσω της Mathf.Lerp. Ορίζουμε μια μεταβλητή currentRotation στην οποία μετατρέπουμε την γωνία Euler του άξονα y σε περιστροφή μέσω της **Quaternion.Euler**, η οποία επιστρέφει μια περιστροφή με currentRotationAngle μοίρες γύρω από τον άξονα y. Για να βρούμε τη θέση της κάμερας πίσω από τον πιγκουίνο, αφαιρούμε από τη θέση του ήρωα την currentRotation περιστροφή της στον άξονα των y, κρατώντας μια απόσταση **distance**. Η θέση είναι προσδιορισμένη μόνο ως προς τον άξονα z, το οποίο εξασφαλίζεται με την **Vector3.forward** η οποία πολλαπλασιάζεται στην τιμή της θέσης της κάμερας και μηδενίζει τις τιμές στους άλλους άξονες αφού είναι στην ουσία Vector3(0, 0, 1). Στην θέση της κάμερας στον άξονα των y θέτουμε το ύψος της. Και τέλος, με την **Transform.LookAt()** την περιστρέφουμε ώστε να κοιτάει πάντα μπροστά προς τον στόχο.

```
function LateUpdate () {
        if (!target)        // Early out if we don't have a target
                return;
        if((SeeWhaleTouchButton.seeWhaleNow ) && lookAtPos){
        // lerp the camera position to the look at position, and lerp its forward direction to match
                transform.position = Vector3.Lerp(transform.position, lookAtPos.position,
                        Time.deltaTime * smooth);
                transform.forward = Vector3.Lerp(transform.forward, lookAtPos.forward,
                        Time.deltaTime * smooth);
        }
        else if ( HealthControl.LIVES == 0 )     //if we die the camera needs to chane place at the die
                                                 // position
        {
                // lerp the camera position to the die position, and lerp its forward direction to match
                transform.position = Vector3.Lerp(transform.position, diePos.position,
                                                 Time.deltaTime * smooth);
                transform.forward = Vector3.Lerp(transform.forward, diePos.forward,
                                                 Time.deltaTime * smooth);
```

```
        }
        else if ((SeeWorldTouch.seeWorldNow) && mapPos)  //if we press "x" we can see the world
                                                         //from above
        {
                transform.position = Vector3.Lerp(transform.position, mapPos.position,
                                                  Time.deltaTime * smooth);
                transform.forward = Vector3.Lerp(transform.forward, mapPos.forward,
                                                  Time.deltaTime * smooth);
        }
        else
        {
                // Calculate the current rotation angles
                var wantedRotationAngle = target.eulerAngles.y;
                var wantedHeight = target.position.y + height;
                var currentRotationAngle = transform.eulerAngles.y;
                var currentHeight = transform.position.y;

                // Damp the rotation around the y-axis
                currentRotationAngle = Mathf.LerpAngle (currentRotationAngle,
                                        wantedRotationAngle, rotationDamping * Time.deltaTime);
                // Damp the height
                currentHeight = Mathf.Lerp (currentHeight, wantedHeight, heightDamping *
                                                  Time.deltaTime);

                // Convert the angle into a rotation
                var currentRotation = Quaternion.Euler (0, currentRotationAngle, 0);
                // Set the position of the camera on the x-z plane to: distance meters behind the target
                transform.position = target.position;
                transform.position -= currentRotation * Vector3.forward * distance;
                transform.position.y = currentHeight;            // Set the height of the camera
                transform.LookAt (target);       // Always look at the target
        }
}
```

5.5.5 Οργάνωση των ετικετών GUI

Ετικέτες στην οθόνη του παιχνιδιού

Για την εμφάνιση των εικόνων και των ετικετών στην οθόνη που αφορούν πληροφορίες του παιχνιδιού, όπως ζωές, νομίσματα, ψαράκια, εχθροί που απομένουν, δημιουργήσαμε ένα GUIText το οποίο διαχειριζόμαστε με τον κώδικα **ScoreTextMobile.js**. Έχουμε προσαρτήσει επίσης ένα συστατικό Audio Source για να αναπαράγουμε την μουσική (background) υπόκρουση του παιχνιδιού. Ο ήχος ακούγεται μόνο από τον Audio Listener που βρίσκεται στην

κύρια κάμερα, η οποία ακολουθεί τον ήρωα. Για να μπορέσει να ακουστεί η μουσική του παιχνιδιού καθόλη τη διάρκεια κίνησης του ήρωα, πρέπει η πηγή να βρίσκεται στην εμβέλεια ακουστικής της κάμερας. Έτσι συνδέουμε το GameObject στην κάμερα ως παιδί της για να την ακολουθεί παντού και να ακούγεται πάντα η μουσική.

Στοιχεία που διαχειρίζεται, όπως ο χρόνος μέχρι την ολοκλήρωση της πίστας, χρησιμοποιούνται για την απόδοση της βαθμολογίας στο τέλος του παιχνιδιού. Ορίζουμε τη float μεταβλητή **Timer** στην οποία κάθε καρέ που δεν έχει πεθάνει ο ήρωας και δεν έχει επιλεχθεί παύση του παιχνιδιού, είτε με τα δύο δάχτυλα είτε με το κουμπί παύσης, και η συσκευή είναι στη σωστή θέση, της προσθέτουμε την Time.deltaTime για να υπολογίσουμε τον συνολικό χρόνο δράσης.

Όταν ο ήρωας πεθάνει και εμφανιστεί η καρτέλα τέλους, ή όταν κερδίσει και εμφανιστεί το σκορ, ή όταν επιλεχθεί παύση και εμφανιστεί το αντίστοιχο μενού, η μουσική του παιχνιδιού δεν σταματάει αλλά χαμηλώνει την έντασή της μέσω της **AudioListener.volume**. Όταν το παιχνίδι συνεχιστεί δυναμώνει πάλι η ένταση στην αρχική της τιμή.

```
private var Timer : float;          //to show the time we play
function Update ()
{
        if( HealthControl.LIVES != 0 && (PauseTouch.pauseNow == false &&
          AndroidMoveAround.PauseFingers == false) && TiltControlMoveAround.canPlay == false )
        {       //if we have not died or the pause button is pushed, stop the time
                Timer += Time.deltaTime;        //incease the time
        }
        if( HealthControl.GameOver || HealthControl.WinGame || PauseTouch.pauseNow == true ||
            AndroidMoveAround.PauseFingers == true || TiltControlMoveAround.canPlay == false )
        {
                AudioListener.volume = 0.3;             //all AudioSources playing will be paused
        }
        else if(  HealthControl.GameOver == false && HealthControl.WinGame == false &&
                PauseTouch.pauseNow == false && AndroidMoveAround.PauseFingers == false
                &&TiltControlMoveAround.canPlay == true)
        {//if we have not died or the pause button is pushed, stop the time
                AudioListener.volume = 1;
        }
}
```

Στην συνάρτηση OnGUI() ρυθμίζουμε τη θέση, το μέγεθος και την εικόνα των ψαριών μπόνους, τα οποία αρχικά είναι με αχνά χρώματα, και όταν ο ήρωας τα συλλέξει παίρνουν έντονο χρώμα. Οι ετικέτες είναι ενεργές μόνο εφόσον ο ήρωας δεν έχει χάσει ή κερδίσει, δεν έχει επιλεχθεί η παύσης και η συσκευή είναι στη σωστή θέση. Για να γνωρίζουμε πότε έχουν

συλλεχθεί ελέχουμε πότε είναι αληθείς οι μεταβλητές Fishy1, Fishy2, Fishy3 από την OnTriggerEnter της AndroidMoveAround. Από αυτήν επίσης διαβάζουμε τη μεταβλητή coins που μας ενημερώνει για τον τωρινό αριθμό νομισμάτων που έχει συλλέξει ο χρήστης στη συγκεκριμένη πίστα, και αναγράφουμε την τιμή της κάθε φορά δίπλα σε μια ετικέτα με ένα νόμισμα.

Υπάρχουν δύο μικρές ετικέτες, μία ενός βίκινγκ και μία με ένα κανόνι, όπου δίπλα αναγράφεται ο αριθμός του συγκεκριμένου εχθρού στην πίστα. Κάθε φορά που σκοτώνεται ένας εχθρός, η τιμή αυτή μειώνεται κατά ένα, ώστε να ξέρει κάθε φορά ο χρήστης πόσοι εχθροί του απομένουν.

Υπολογίζουμε επίσης το χρόνο σε λεπτά και δευτερόλεπτα, για να τον χρησιμοποιήσουμε στην απόδοση της βαθμολογίας. Για να βρούμε τα λεπτά, διαιρούμε την τιμή του Timer με το 60 και παίρνουμε τον μεγαλύτερο ακέραιο που είναι μικρότερος ή ίσος με το αποτέλεσμα, μέσω της **Mathf.FloorToInt**, και το θέτουμε στη μεταβλητή **minutes**. Για να βρούμε τα δευτερόλεπτα, αφαιρούμε από τον Timer την minutes πολλαπλασιασμένη με 60 και κρατάμε τον μεγαλύτερο ακέραιο αριθμό, και το αποθηκεύουμε στην μεταβλητή **seconds**. Για να δώσουμε τη μορφή του χρόνου 0:00 στην τελική βαθμολογία, χρησιμοποιούμε την **String.Format** θέτοντας ως εισόδους τις μεταβλητές minutes και seconds και το αποθηκεύουμε στην **niceTime**.

```
minutes = Mathf.FloorToInt(Timer / 60F);
var seconds : int = Mathf.FloorToInt(Timer - minutes * 60);

niceTime = String.Format("{0:0}:{1:00}", minutes, seconds);
//This will give times in the 0:00 format, {indexOfParameter:formatting}
```

Έλεγχος ζωών και χτυπημάτων

Οι ζωές στο παιχνίδι αντιπροσωπεύονται από τρεις κόκκινες καρδούλες στο πάνω αριστερό μέρος της οθόνης και ελέγχονται από ένα GUITexture στο οποίο ρυθμίζουμε τις καταστάσεις μέσω του κώδικα **HealthControl.js**. Έχουμε τέσσερις διαφορετικές εικόνες οι οποίες εναλλάσονται αναλόγως τις ζωές του ήρωα, μία εικόνα με τρεις κόκκινες καρδούλες όταν έχει τρεις ζωές, μία εικόνα με δύο καρδούλες όταν έχει δύο ζωές, μια εικόνα με μια καρδούλα όταν του έχει μείνει μία ζωή, και μια εικόνα με τρεις γαλάζιες καρδούλες όταν πεθαίνει και δεν έχει καμία ζωή. Για να αφαιρεθεί μια ζωή πρέπει να χτυπηθεί δύο φορές από ένα κανόνι, έχει δηλαδή δύο χτυπήματα, ή να τον πιάσει ένας βίκινγκ όπου πεθαίνει αμέσως. Αυτά τα δύο χτυπήμα συμβολίζονται από τα δύο πιγκουινάκια που τον ακολουθούν. Με το πρώτο χτύπημα χάνεται το τελευταίο πιγκουινάκι, με το δεύτερο χτύπημα χάνεται και το πρώτο, ενώ με το τρίτο χτύπημα πεθαίνει. Αν πεθάνει και του απομένουν κ άλλες ζωές, επαναφέρεται στο αρχικό του σημείο με τα δύο πιγκουινάκια στη θέση τους.

Όταν χτυπηθει ο ήρωας, δεν τα καταστρέφουμε τα πιγκουινάκια, γιατί θέλουμε να τα επαναφέρουμε πάλι αργότερα, απλά τα κάνουμε αόρατα με την **Renderer.enabled** και όταν τα χρειαστούμε τα κάνουμε πάλι ορατά. Ο έλεγχος και η ρύθμιση των ζωών και των πιγκουίνων γίνεται στην Update() καθώς πρέπει να γίνονται άμεσα.

```
function Update ()
{	//it displays how many lives and hits we have in our current game
	switch(LIVES)  //we need to check if the status of our lives has changed
	{
		case 3:
			guiTexture.texture = health3;
		break;
		case 2:
			guiTexture.texture = health2;
		break;
		case 1:
			guiTexture.texture = health1;
		break;
		case 0:
			guiTexture.texture = health0;		//GAME OVER
		break;
	}
	//another switch-case statement for hits but now we are counting up
	switch(HITS)
	{
		case 0:  //enable all penguins
			smallPenguin1.renderer.enabled = true;
			smallPenguin2.renderer.enabled = true;
		break;
		case 1:  //disable penguin2

			smallPenguin1.renderer.enabled = true;
			smallPenguin2.renderer.enabled = false;
			//this will turn off the renderer of the penguin2 so we cannot see it anymore,
			//we do not destroy it because we want to use it again and turn it on
		break;
		case 2:  //disable penguin1
			smallPenguin1.renderer.enabled = false;
			smallPenguin2.renderer.enabled = false;		//for safety, in case penguint2
									//is still there
		break;
		case 3:  //we need to respawn and subtract a life
			smallPenguin1.renderer.enabled = true;		//enable the little penguins to
									// be visible again
```

```
                    smallPenguin2.renderer.enabled = true;
                break;
        }
}
```

Στη συνάρτηση Start() αρχικοποιούμε τις ζωές, τα χτυπήματα και το χρόνο του παιχνιδιού. Στη συνάρτηση OnGUI() όταν οι ζωές του τελειώσουν και πεθάνει, εμφανίζουμε στο κέντρο της οθόνη ένα μήνυμα "Game Over" το οποίο αναβοσβήνει για κάποια δευτερόλεπτα και μετά μένει σταθερό μέχρι να εμφανιστεί η καρτέλα τέλους παιχνιδιού.

```
if (LIVES == 0 )
{
        if (Time.time % 2 < 1 && count> -290)
        {
                GUI.Label (Rect ( widthBut, heightBut, totalWidth, totalHeight ), "Game Over",
                                        GameOverStyle);
                count = count - 1;
        }
        if( count <= -290 && count > -292 )
        {
                guiText.enabled = true;          //turns on the text game over on screen
                count = count - 1;
        }
        if ( count <= -292 )
        {
                guiText.enabled = false;
                GameOver = true;                 //game over menu
                Time.timeScale = 0.0;            //stop time
                LIVES = 3;
        }
  }
```

Καρτέλα βαθμολογίας

Όταν κερδίσει εμφανίζεται η βαθμολογία μέσω του GUITexture που ρυθμίζουμε με τον κώδικα **WinGameGUIMobile.js**. Στη συνάρτηση OnGUI() ενεργοποιούμε την εικόνα που βρίσκεται στο πίσω μέρος, και μέσω της **GUI.Label** εμφανίζουμε στο κατάλληλο μέγεθος, χρώμα και θέση το σκορ, το χρόνο, τα νομίσματα και τα ψάρια μπόνους που συλλέχθηκαν με τα αντίστοιχα εικονίδια. Στην GUI.Label και στην GUI.Button θέτουμε τις συντεταγμένες του χώρου στις οποίες θέλουμε να βρίσκεται η ετικέτα ή το κουμπί, καθώς και την εικόνα ή και το μήνυμα που θέλουμε να εμφανιστεί. Η θέση καθορίζεται με τη συνάρτηση **Rect ()** που ορίζει τέσσερις ιδιότητες: αριστερή θέση, ανώτερη θέση, συνολικό πλάτος, συνολικό ύψος. Όλες αυτές οι τιμές παρέχονται σε ακέραιους αριθμούς, που αντιστοιχούν σε τιμές εικονοστοιχείων.

Η βαθμολογία διαμορφώνεται από το άθροισμα των νομισμάτων των ψαριών, όπου κάθε νόμισμα είναι ένας βαθμός και τα ψάρια είναι 100, 200 ή 300 βαθμοί αναλόγως το χρώμα του, από το οποίο αφαιρούμε δέκα βαθμούς για κάθε λεπτό που χρειάστηκε για να τερματίσει το παιχνίδι.

Κάτω από τα αποτελέσματα αυτά, υπάρχουν τρία κουμπιά αφής τα οποία δημιουργήθηκαν μέσω της **GUI.Button**. Το πρώτο κουμπί μας επιστρέφει στο μενού των πιστών αυτού του κόσμου, στην καρτέλα με τις πίστες και όχι με τους κόσμους, θέτοντας την οθόνη σε LandscapeLeft θέση, το δεύτερο κουμπί είναι για την επανάληψη της πίστας, και το τρίτο κουμπί είναι για την εισαγωγή στην αμέσως επόμενη πίστα. Αναλόγως την επιλογή του χρήστη φορτώνεται η αντίστοιχη σκηνή μέσω της **Application.LoadLevel()** και συνεχίζεται πάλι ο χρόνος, καθώς είχε σταματήσει μόλις τερματίσαμε.

```
if (GUI.Button (Rect (replayWidth, heightBut3, totalWidth3, heightButtons), replayButton,""))
{
        Time.timeScale = 1.0;                   //restart the time
        audio.PlayOneShot(PressedButtonSound);  //sound when the button is pressed
        HealthControl.WinGame = false;          //initialize
        Application.LoadLevel (SceneNumber);            //Load the scene
}
```

Στο κάτω μέρος της οθόνης εμφανίζουμε ένα μήνυμα με την GUI.Label με το οποίο παροτρύνουμε τον χρήστη να μην εγκαταλείψει το παιχνίδι.

Καρτέλα ήττας

Όταν ο χρήστης χάσει εμφανίζεται η καρτέλα τέλους με το GUITexture που ρυθμίζεται από τον κώδικα **GameOverGUIMobile.js**. Μέσω της συνάρτησης OnGUI() ενεργοποιούμε την εικόνα της καρτέλας όπου στο κέντρο υπάρχουν τρία κουμπία ίδια με το μενού της βαθμολογίας. Η διαφορά είναι ότι στο κουμπί που μας εισάγει στην επόμενη πίστα, γίνεται έλεγχος αν η τωρινή πίστα έχει τερματιστεί επιτυχώς κάποια προηγούμενη φορά για να μπορέσουμε να προχωρήσουμε. Διαφορετικά απενεργοποιείται και εμφανίζεται με ένα αχνό γκρι χρώμα. Στον κώδικα παρακάτω δείχνουμε τον έλεγχο αυτόν για τις δύο πρώτες πίστες της SnowLand, και ακολουθείται ο ίδιος τρόπο για τις υπόλοιπες πίστες. Στο κάτω μέρος εμφανίζουμε πάλι ένα μήνυμα παρότρυνσης του χρήστη για να συνεχίσει το παιχνίδι.

```
//we can't go to the nexr scene until we won the current scene
if( (SceneNumber == 2 && PlayerPrefs.GetInt("WinMove1") == 0) || (SceneNumber == 3 &&
        PlayerPrefs.GetInt("WinMove2") == 0))  // ……..more cases
{
        GUI.Label(Rect (nextWidth, heightBut3, totalWidth3, heightButtons), nonextButton,"");
}
else    //we have next scenes
{
```

```
        // Make the third button. If it is pressed, we are going to the NEXT SCENE
        if (GUI.Button (Rect (nextWidth, heightBut3, totalWidth3, heightButtons), nextButton,"")) {
                Time.timeScale = 1.0;                   //restart the time
                audio.PlayOneShot(PressedButtonSound);  //sound when the button is pressed
                HealthControl.GameOver = false;         //initialize
                Application.LoadLevel (NextScene);      //Load the scene
        }
}
```

Για να ξέρουμε σε ποιά πίστα βρισκόμαστε κάθε φορά, χρησιμοποιούμε μεταβλητές στις οποίες αποθηκεύουμε τον αριθμό της κάθε πίστας. Για τον κόσμο της SnowLand έχουμε ορίσει μια μεταβλητή **snowland** στην οποία θέτουμε τους αριθμούς από 1 έως 8 για την κάθε πίστα του κόσμου. Αντίστοιχα έχουμε τις μεταβλητές **waterland**, **jumpland** και **runland** από οχτώ τιμές η κάθε μία που αντιπροσωπεύουν τις πίστες του κάθε κόσμου ώστε να τις ελέγχουμε. Η μεταβλητή **SceneNumber** περιλαμβάνει τον αριθμό της κάθε σκηνής με την σειρά με την οποία έχει τοποθετηθεί στο project ώστε να μπορούμε να οδηγηθούμε στην αμέσως επόμενη αριθμητικά.

Καρτέλα παύσης

Όταν ο χρήστης επιλέξει παύση παιχνιδιού, είτε με το αντίστοιχο κουμπί είτε τοποθετώντας τρία δάχτυλα ταυτόχρονα στην οθόνη, και η συσκευή είναι τοποθετημένη στη σωστή θέση, εμφανίζεται η καρτέλα παύσης GUITexture που ελέγχεται μέσω του **GUIbuttons.js**. Εμφανίζεται η εικόνα της καρτέλας και τέσσερα κουμπιά αφής που υλοποιήθηκαν με τη GUI.Button. Δύο κουμπιά, ένα μικρότερο στην πάνω αριστερή άκρη και ένα στο κέντρο, με διαφορετικό εικονίδιο είναι για την επιστροφή στο σημείο που σταματήσαμε, ένα είναι για την επανάληψη της πίστας, και ένα για την επιστροφή στο μενού των πιστών και όχι των κόσμων θέτοντας την οθόνη σε LandscapeLeft θέση. Στο κάτω μέρος εμφανίζουμε πάλι ένα μήνυμα συνέχισης του παιχνιδιού που αναφέρεται στο χρήστη.

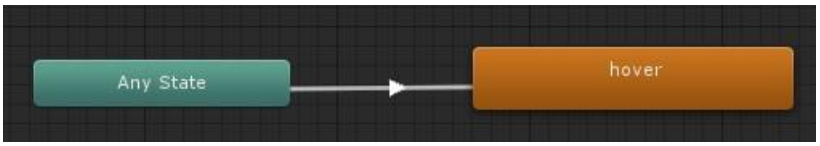
```
// Make the first button. If it is pressed, the game will be continued
if (GUI.Button (Rect ( widthBut, heightBut - Screen.height * 1/3, totalWidth, totalHeight ),
                playButton,"")) {
        audio.PlayOneShot(PressedButtonSound);      //sound when the button is pressed
        Time.timeScale = 1.0;                       //restart the time
        PauseTouch.pauseNow = false;                //initialize
        AndroidMoveAround.PauseFingers = false;
}
// Make the second button. If it is pressed, the game will be restart
if (GUI.Button (Rect ( widthBut, heightBut, totalWidth, totalHeight ), replayButton, "")) {
        audio.PlayOneShot(PressedButtonSound);      //sound when the button is pressed
        Application.LoadLevel (SceneNumber);                //Load the scene
        Time.timeScale = 1.0;                       //restart the time
        PauseTouch.pauseNow = false;                //initialize for mobile game
```

```
        AndroidMoveAround.PauseFingers = false;
}
```

5.5.6 Οργάνωση των μικρών πιγκουίνων

Ο πρώτος πιγκουίνος ακολουθεί με κάποια απόσταση και με ομαλή κίνηση ένα κενό GameObject στο κάτω και πίσω μέρος του ήρωα, με το οποίο ρυθμίζουμε το ύψος με το οποίο θέλουμε να τον ακολουθεί κάθε φορά, ενώ ο δεύτερος πιγκουίνος ακολουθεί τον πρώτο με κάποια απόσταση. Στην περίπτωση που ο ήρωας κολυμπάει σε βαθια νερά, για να φαίνονται τα πιγκουινάκια αλλάζουμε το σημείο που ακολουθούν, και έχουν ως στόχο ένα κενό GameObject που βρίσκεται πιο ψηλά ώστε να βρίσκονται στην επιφάνεια του νερού. Το GameObject αυτό το ακολουθεί μόνο ο πρώτος πιγκουίνος, καθώς ο δεύτερος μεταφέρεται αυτόματα πιο ψηλά ακολουθώντας τον πρώτο. Για να ακολουθεί το κάθε πιγκουινάκι το στόχο του, χρησιμοποιήσαμε τον ίδιο τρόπο με τον οποίο η κύρια κάμερα ακολουθεί τον ήρωα όσο δεν είναι επιλεγμένο κάποιο κουμπί. Την μεμονωμένη αυτή λειτουργία περιέχει ο κώδικας **SmoothFollow.js** τον οποίο έχουμε προσαρτήσει στο δεύτερο πιγκουινάκι ορίζοντας ως στόχο του τον πρώτο πιγκουίνο. Η κίνηση του πρώτου πιγκουίνου περιλαμβάνει δύο στόχους και πρέπει να ελέγχει πότε βρίσκεται στο νερό και πότε όχι για να εναλλάσονται οι στόχοι τους οποίους ακολουθεί ακριβώς με τον ίδιο τρόπο μέσω του **PenguinFollowCtrl.js**.

Καθόλη τη διάρκεια του παιχνιδιού, και τα δύο μικρά πιγκουινάκια κάνουν την ίδια κίνηση με την οποία φαίνεται σαν να αιωρούνται στον αέρα. Για την απόδοση αυτής της κίνησης τους έχουμε προσαρτήσει το συστατικό Animator με μία μόνο κατάσταση animation στην οποία βρίσκονται συνεχώς.



5.5.7 Οργάνωση των αντικειμένων συλλογής

Τα αντικείμενα που μπορεί να συλλέξει ο ήρωας, αιωρούνται στον αέρα, και περιστρέφονται γύρω από τον εαυτό τους. Μόλις ο collider του μαγνήτη του πιγκουίνου, έρθει σε επαφή με τον collider του αντικειμένου θέτουμε αληθή την σχετική μεταβλητή μέσω της συνάρτησης OnTriggerEnter() για να ξέρουμε πότε πρέπει να γίνει η αντίστοιχη λειτουργία, και καταστρέφουμε το αντικείμενο με την Object.Destroy() μετά από 0,1 δευτερόλεπτο. Τα νομίσματα τα ελέγχουμε από το **CoinControl.js**, τα ψαράκια μπόνους από το **FishControl.js**, το μαγικό σμαράγδι με το **PowerUpControl.js**, τις καρδούλες με το **HeartControl.js**, και πιγκουινάκια για την αύξηση των χτυπημάτων που μπορεί να δεχθεί ο ήρωας από το **PenguinLifeCtrl.js**. Πριν καταστραφεί η καρδούλα, αυξάνουμε κατά ένα των αριθμό των ζωών του ήρωα.

Τα νομίσματα αιωρούνται στον αέρα σε ένα σταθερό σημείο και περιστρέφονται γύρω από τον εαυτό τους, ενώ όλα τα άλλα αντικείμενα εκτός από την περιστροφή γύρω από τον εαυτό τους ανεβοκατεβαίνουν στον αέρα εντός συγκεκριμένων ορίων. Έχουμε ρυθμίσει τα αντικείμενα, εκτός το μαγικό σμαράγδι, έτσι ώστε όταν τα πλησιάζει σχετικά κοντά ο ήρωας να μαγνητίζονται από αυτόν και να πηγαίνουν προς το μέρος του, ώστε να είναι πιο εύκολο να τα συλλέξει. Η κατεύθυνση προς τον μαγνήτη γίνεται μέσω της **Vector3.MoveTowards** η οποία μετακινεί ένα τρέχον σημείο σε μια ευθεία γραμμή προς ένα σημείο-στόχο, στην οποία εισάγουμε τη θέση του στόχου που θέλουμε και την θέτουμε στη θέση του νομίσματος, και καθορίζουμε μέχρι ποιά απόσταση θέλουμε να πλησιάσει και να μην υπερβεί.

Την περιστροφή του αντικειμένου γύρω από τον εαυτό του την ρυθμίζουμε με την **Transform.Rotate** εφαρμόζοντας μία περιστροφή (eulerAngles.y) γύρω από τον άξονα y που αναπαραστάται με την **Vector3.up**. Η περιστροφή γίνεται με μια ομαλή κίνηση ανά δευτερόλεπτο.

```
function Update ()
{       //rotates the coin in the air
        transform.Rotate(Vector3.up * Time.deltaTime*100, Space.World);

        if(target) {
                transform.position = Vector3.MoveTowards(transform.position,
                                                        target.transform.position, 1.0f);
        }
}
```

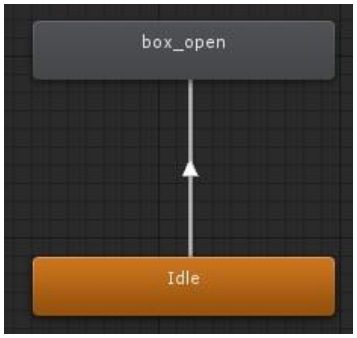
Για τα αντικείμενα που ανεβοκατεβαίνουν στον αέρα θέτουμε στην μεταβλητή **angle** αρχικά την τιμή -90 για να καθορίσουμε το ύψος χρησιμοποιώντας τα ημίτονα. Στη μεταβλητή **toDegrees** μετατρέπουμε τα ακτίνια σε μοίρες διαιρώντας το π **Mathf.PI** με το 180, και στη μεταβλητή **startHeight** ορίζουμε το ύψος το οποίο θα έχει το αντικείμενο όταν αρχίσει ο κώδικας. Στην συνάρτηση FixedUpdate() προσθέτουμε στην angle την ταχύτητα περιστροφής κατά τη διάρκεια του χρόνου. Αν ξεπεράσει τις 270 μοίρες, την θέτουμε ίση με -360 για να συνεχίζει να περιστρέφεται. Στη μεταβλητή **maxUpAndDown** ορίζουμε την μέγιστη απόσταση με την οποία μπορεί να ανεβοκατεβαίνει στον αέρα, σε μέτρα της Unity. Τον αριθμό γωνιών της angle το πολλαπλασιάζουμε με την toDegrees για να πάρουμε τα ακτίνια, τα οποία τα θέτουμε ως είσοδο στην **Mathf.Sin** για να μας δώσει το ημίτονο της γωνίας αυτής σε ακτίνια. Το αποτέλεσμα το πολλαπλασιάζουμε με την maxUpAndDown, το προσθέτουμε στον εαυτό της και το διαιρούμε με το 2 για να πάρουμε μια αρμονική κίνηση σύμφωνα με την περιστροφή του και το ύψος στο οποίο μπορεί να κινηθεί.

Την αυξομείωση του ύψους του αντικειμένου το ρυθμίζουμε με την **transform.localPosition.y**, καθώς μας ενδιαφέρει μόνο ο άξονας των y, στην οποία

προσθέτουμε στο αρχικό ύψος startHeight το παραπάνω αποτέλεσμα της παλινδρομικής κίνησης.

```
var maxUpAndDown : float = 1;          // amount of meters going up and down
var speed : float = 50;                        // up and down speed
protected var angle : float = -90;          // angle to determin the height by using the sinus
protected var toDegrees : float = Mathf.PI/180;   // radians to degrees
protected var startHeight : float;          // height of the object when the script starts
function FixedUpdate()
{
        angle += speed * Time.deltaTime;
        if (angle > 270) angle -= 360;
        transform.localPosition.y = startHeight + maxUpAndDown *
                (1 + Mathf.Sin(angle * toDegrees)) / 2;
        //Mathf.Sin : Returns the sine of angle f in radians.
}
```

5.5.8 Υλοποίηση του θυσαυρού-τερματισμού



Στο θησαυρό έχουμε προσαρτήσει το συστατικό Animator στο οποίο έχουμε ένα απλό animation για να ανοίγει το σεντούκι και να φαίνονται τα χρυσά νομίσματα όταν τερματίζει ο ήρωας. Σε ευκολότερες πίστες για να ανοίξει το σεντούκι και να ολοκληρωθεί η πίστα, αρκεί απλά να το ακουμπήσει ο ήρωας και ελέγχεται μέσω του **Treasure.js**. Αρχικά φροντίζουμε να είναι κλειστό το σεντούκι θέτοντας ψευδή την παράμετρο που οδηγεί στη κατάσταση του animation ανοίγματος. Επίσης ελέγχουμε μέσω της συνάρτησης OnTriggerEnter() πότε ο collider του θυσαυρού έρχεται σε επαφή με τον CharacterController του ήρωα, όπου ενεργοποιείται το animation ανοίγματος του σεντουκιού. Με την συνάρτηση OnGUI() εμφανίζουμε στο κέντρο της οθόνης ένα μήνυμα “You Win!” που αναβοσβήνει για μερικά δευτερόλεπτα και μετά παραμένει σταθερό μέχρι να εμφανιστεί η καρτέλα της βαθμολογίας, και σταματάμε το χρόνο του θυσαυρού, όπως ακριβώς κάνουμε στην HealthControl.js όταν πεθαίνει ο ήρωας.

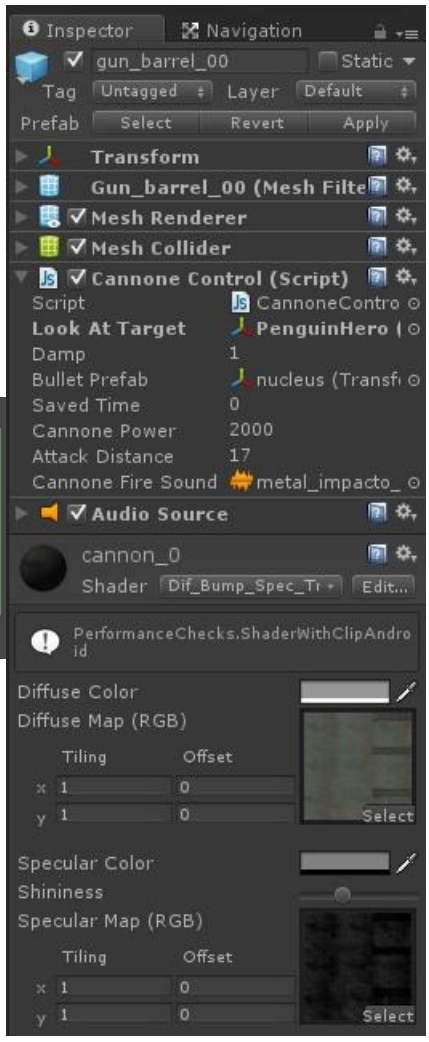
```
function Update()
{
        if(AndroidMoveAround.CANNONES == 0 && AndroidMoveAround.VIKINGS == 0)
        {
                animTreasure.SetBool("TreasureOpen", true);
        }       //SetBool : Sets the value of a bool parameter
}
```

Σε δυσκολότερες πίστες για να ανοίξει το σεντούκι πρέπει πρώτα να σκοτώσει όλους τους εχθρούς, το οποίο ελέγχουμε στην Update() μέσω του **TreasureControlMobile.js**. Αν πάει

σε αυτόν, το οποίο το ελέγχουμε με την OnTriggerEnter(), ενώ υπάρχουν ακόμα εχθροί στην πίστα εμφανίζεται ένα μήνυμα στην οθόνη "Destroy your enemies first!" που τον ειδοποιεί ότι πρέπει να σκοτώσει όλους τους εχθρούς του πρώτα. Μόλις απομακρυνθεί από το θησαυρό, το οποίο το ξέρουμε μέσω της OnTriggerExit(), το μήνυμα εξαφανίζεται.

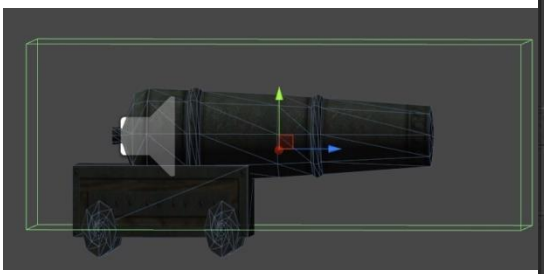
Υπάρχει τοποθετημένη μια δεύτερη κάμερα μπροστά από τον θησαυρό την οποία ελέγχουμε με το **CamTreasureSwitcherMobile.js**. Μόλις σκοτώσει τον τελευταίο εχθρό, η κάμερα αλλάζει, απενεργοποιώντας την κύρια κάμερα και ενεργοποιώντας αυτήν ώστε να φαίνεται στην οθόνη για λίγα δευτερόλεπτα μόνο το σεντούκι την ώρα που ανοίγει αποκαλύπτοντας το θησαυρό, για να ενημερώσει τον χρήστη ότι τώρα μπορεί να τερματίσει. Έπειτα την απενεργοποιούμε και ενεργοποιούμε πάλι την κύρια κάμερα για να μπορεί να παίξει ο χρήστης.

```
function Update ()
{	//all the enemies are dead and the treasure chest has opened
	if(AndroidMoveAround.CANNONES == 0 && AndroidMoveAround.VIKINGS == 0
	&& counter < 400)
	{
		treasureCam.camera.enabled = true;
		mainCam.camera.enabled = false;
		counter = counter + 1;
	}
	else
	{
		treasureCam.camera.enabled = false;
		mainCam.camera.enabled = true;
	}
}
```



5.5.9 Υλοποίηση εχθρών

Υλοποίηση κανονιού



Όταν ο πιγκουίνος πλησιάσει τα κανόνια σε μια συγκεκριμένη απόσταση που έχουμε ορίσει, τον εντοπίζουν και τον σημαδεύουν. Αν ξεπεράσει αυτή την απόσταση δεν απειλείται άλλο από αυτά. Τα κανόνια τα ελέγχουμε με τον κώδικα **CannoneControl.js**. Την απόσταση μεταξύ της θέσης του πιγκουίνου και του κανονιού την βρίσκουμε μέσω της **Vector3.Distance.** Αν ο πιγκουίνος βρεθεί μέσα στα όρια εντοπισμού, τότε το κανόνι περιστρέφεται προς το μέρος του. Στη μεταβλητή rotate ορίζουμε την

Quaternion.LookRotation, που δημιουργεί μια περιστροφή με τις καθορισμένες κατευθύνσεις προς τα εμπρός και προς τα πάνω, στην οποία εισάγουμε τη διαφορά της θέσης του στόχου και θέσης του κανονιού. Για την περιστροφή χρησιμοποιούμε την **Quaternion.Slerp** στην οποία εισάγουμε την τωρινή περιστροφή του κανονιού, την διαφορά περιστροφής μεταξύ του στόχου και του κανονιού και τον χρόνο με τον οποίο θέλουμε να περιστραφεί πολλαπλασιασμένο με μια μεταβλητή καθυστέρησης.

Το κανόνι μπορεί να πυροβολίσει μόνο μία φορά κάθε δευτερόλεπτο, για να είναι πιο εύκολο στο χρήστη να το αποφύγει. Αποθηκεύουμε στη μεταβλητή **seconds** το χρόνο, ο οποίος είναι float, αλλά επειδή θέλουμε μόνο τα δευτερόλεπτα την θέτουμε ως integer. Θέτουμε την άρτια η τιμή της στη μεταβλητή **oddeven** χρησιμοποιώντας το module 2, και καλούμε τη συνάρτηση Shoot() με την οποία το κανόνι πυροβολάει τον πιγκουίνο. Ως είσοδό της ορίζουμε τα δευτερόλεπτα που υπολογίζουμε.

```
function Update ()
{       //we need to make the cannon to look at the target
        //the cannones will point at the hero only when he comes too close
        var distance = Vector3.Distance(gameObject.FindWithTag("Player").transform.position,
                                gameObject.transform.position);
                                //to calculate the distance between the  penguin and the cannone
        if( distance <= attackDistance && distance>5) )
        {       //we wil use a little bit slower rotation that Transform.LookAt
                if(LookAtTarget)        //safety net
                {
                    var rotate = Quaternion.LookRotation(LookAtTarget.position -
                                                        transform.position);
 //for inputs we give 2 positions, 2 Vector3 variables(the position of the target and the position of the
//cannone) and from their difference it will know the rotation in 3d space to make the rotation
                        transform.rotation = Quaternion.Slerp(transform.rotation, rotate,
                                                        Time.deltaTime * damp  );
//it takes 3 variables: (1)our current rotation, (2)the difference between two points in space, (3)the
time //which is needed (it looks the rotation we have and the rotation it is needed and then it will
rotate it in //the time we have specified * a damping var to be slower)
//deltaTime is the time difference between one frame and the next one

                        //only call it once a second to make it easier to shoot
                        var seconds : int = Time.time;          //it's a float var and we only want the
                                                                // seconds, so we make it integer
                        //if we divided it by 2 it's either even or uneven
                        var oddeven = (seconds % 2);    //this will be true or false
                        if(oddeven)                     //if it's true we can shoot
                        {
                                Shoot(seconds);//we send our seconds variable
```

```
                    }//and if it's not true we don't shoot
                }
        }
}
```

Όταν καλείται η συνάρτηση **Shoot**, πρέπει να θυμόμαστε το χρόνο με τον οποίο κλήθηκε, και αν ο χρόνος αυτός έχει χρησιμοποιηθεί για να πυροβολήσει το κανόνι, δε θα τον χρησιμοποιήσουμε ξανά. Ελέγχουμε αν τα δευτερόλεπτα αυτά δεν έχουν χρησιμοποιηθεί για πυροβολισμό, μέσω της μεταβλητής savedTime στην οποία αποθηκεύουμε το χρόνο που χρησιμοποιήθηκε. Στη συνέχεια δημιουργούμε με την Instantiate ένα βλήμα από το σημείο που έχουμε ορίσει μπροστά από το κανόνι τοποθετώντας ένα κενό GameObject **spawnPoint**, και χρησιμοποιούμε την περιστροφή του κανονιού μέσω της Quaternion.identity. Στο κάθε βλήμα που δημιουργείται, θέτουμε την ετικέτα "**enemyProjectile**" για να μπορέσουμε να ελέγξουμε πότε χτυπάει τον ήρωα. Με την **Rigidbody.AddForce** προσθέτουμε μια δύναμη στο rigidbody του βλήματος για να αρχίσει να κινείται με μια συγκεκριμένη δύναμη κατευθυνόμενο μπροστά από το κανόνι κατά τον άξονα των z transform.forward, ενεργοποιώντας ταυτόχρονα και έναν ήχο πυροβολισμού. Στο τέλος αποθηκεύουμε στη **savedTime** τα δευτερόλεπτα με τα οποία κλήθηκε η συνάρτηση για να μπορέσουμε να ελέγξουμε το χρόνο στο επόμενο κάλεσμα.

```
function Shoot(seconds)         //it goes to the Shoot functions a million times a second
{       //as soon as the shoot function is called we need to remember the time that it send us, and if
        //that time is used to shoot a fireball we are not going to use it again
        if(seconds != savedTime)        //if our seconds are already used then we can shoot
        {//we need to be sure that the fireball is created from our spawnpoint and actually moves
                var bullet = Instantiate(bulletPrefab, ransform.Find("spawnPoint").transform.position,
                                                Quaternion.identity);
        //3 var: (1)the prefab we want to shoot, (2)from where we want to shoot(the spawnpoint, it
        //needs to look at our transform to find the "spawnpoint" because we have another
        //spawnpoint in the game), (3)the rotation the same as the current cannone(rotation
        //identity))
                //when it shoots it is tagged as enemyProjectile and we should spinning around
                bullet.gameObject.tag = "enemyProjectile";
                //we need to make this bullet move
                bullet.rigidbody.AddForce(transform.forward * CannonePower);
        //we make it move forward along the z axis. The hero is a little faster from the bullet because
        //we have multiply its movement by 2000, so the game can be easier to beat
                audio.PlayOneShot(CannoneFireSound);                    //shoot sound
                savedTime = seconds;//after shooting we save our time and set it to our seconds
        }
}
```

Στην κεφαλή του κανονιού έχουμε προσαρτήσει ξεχωριστά τον κώδικα **CannoneCollision.js**, με τον οποίο ελέγχουμε πότε ο πιγκουίνος το πετυχαίνει με μια

χιονόμπαλα. Αν το σημαδεύσει καταστρέφουμε τη χιονόμπαλα, δημιουργούμε μια έκρηξη στη θέση του κανονιού για κάποια δευτερόλεπτα, και κάνουμε αόρατη την κεφαλή του κανονιού, μειώνοντας κατά ένα τον συνολικό αριθμό των κανονιών της πίστας. Δεν την καταστρέφουμε γιατί όταν ο ήρωας χάσει μια ζωή και επανέλθει στο αρχικό του σημείο θέλουμε να επανεμφανίσουμε το κανόνι. Αν το καταστρέφαμε δεν θα γινόταν να ξανά ενεργοποιηθεί, καθώς μια καταστροφή αντικειμένου είναι μόνιμη. Έτσι το απενεργοποιούμε κάνοντάς το αόρατο, και το ξανά κάνουμε ορατό όποτε χρειαστεί.

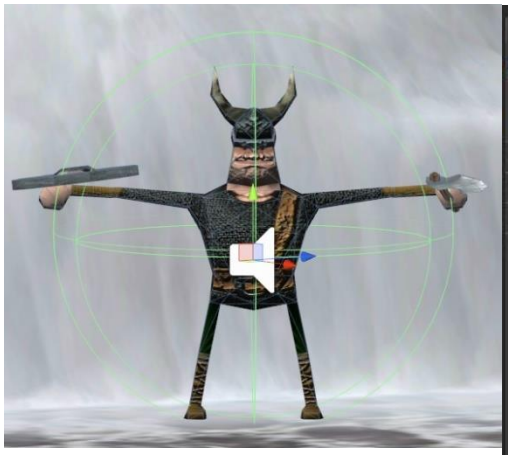
```
function OnTriggerEnter( hit : Collider )
{	//this function is called when something hits the cannones
	if(hit.gameObject.tag == "penguinProjectile")   //we check if the cannon is hit by the penguin
	{
		Destroy(hit.gameObject);		//we destroy the the object that hit the cannone
							//("hit" refers to the projectile)
		var exp = Instantiate(explosion, transform.Find("boomPoint").transform.position,
				Quaternion.identity ); //(1)prefab, (2)the position of this current
				//cannone, (3)the rotation which is the same rotation as the cannone
		gameObject.active = false;//we dont destroy the child because we want to use it again
		AndroidMoveAround.CANNONES -= 1;//one less cannone because we destroyed it
	}
}
```

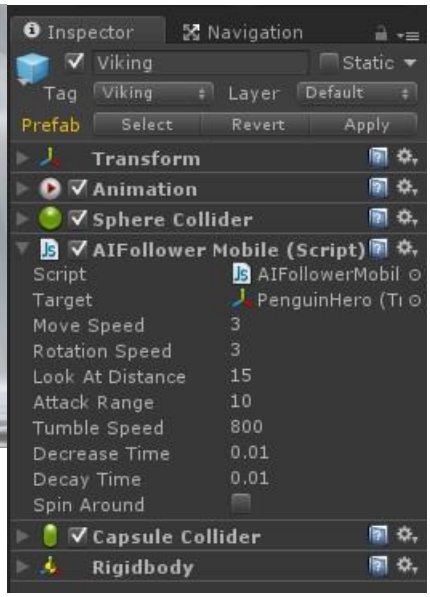
Η επαναφορά ενός απενεργοποιημένου αντικειμένου δεν μπορεί να πραγματοποιηθεί από το ίδιο το αντικείμενο. Έτσι δημιουργούμε ένα κενό GameObject, το οποίο είναι γονέας του κανονιού και χρησιμοποιείται για την επαναφορά του μέσω του κώδικα **ReactivateCannone.js**. Στη συνάρτηση LateUpdate() αποκτάμε πρόσβαση σε όλα τα παιδιά του αντικειμένου, τα οποία τα ενεργοποιούμε με την επαναφορά του ήρωα στο αρχικό του σημείο μετά το χάσιμο μιας ζωής. Επίσης, ο ήχος της έκρηξης κατά την εξολόθρευση του κανονιού, δημιουργείται από αυτό το αντικείμενο, καθώς με την απενεργοποίηση της κεφαλής θα σταματούσε και η αναπαραγωγή του.

```
function LateUpdate()
{
	for (var child : Transform in transform) {// do whatever we want with child transform here
		if(AndroidMoveAround.dead)
		{
			child.active = true;
		}
	}
}
```

Με τον ίδιο τρόπο ενεργοποιούμε τους βίκινγκς με το **ReactivateVikingMobile.js**, τη φάλαινα με το **ReactivateWhale.js** και τα μαγικά σμαράγδια με το **ReactivatePower.js** που βρίσκονται στην πίστα κατά την επανάληψή της.

Υλοποίηση βίκινγκ





Ο βίκινγκ αρχικά στέκεται ακίνητος σε ένα σημείο και ρυθμίζουμε τη συμπεριφορά του μέσα από το script **AIFollowerMobile.js**. Ορίζουμε δύο μεταβλητές, **lookAtDistance** και **attackRange**, στις οποίες θέτουμε μια συγκεκριμένη απόσταση για να καθορίσουμε μια διαφορετική συμπεριφορά του βίκινγκ. Την απόσταση μεταξύ του ήρωα και του βίκινγκ την υπολογίζουμε με την Vector3.Distance. Οι τιμές των μεταβλητών αυτών είναι προσβάσιμες μέσα από τον Inspector της Unity, ώστε να τις αλλάζουμε αναλόγως τη δυσκολία της πίστας. Στην συνάρτηση Start() τον ενεργοποιούμε, σε περίπτωση που είχε μείνει ανενεργός από κάποια προηγούμενη πίστα.

```
function Start()
{
	gameObject.SetActive(true);
}
```

Μόλις ο ήρωας πλησιάσει τον βίκινγκ μέχρι μια συγκεκριμένη απόσταση **lookAtDistance**, καλείται η συνάρτηση **lookAt()** με την οποία τον αντιλαμβάνεται, τον κοιτάει και γυρίζει προς το μέρος του, και ετοιμάζεται να πάει προς αυτόν αλλά δεν προχωράει από τη θέση του. Για την υλοποίησή της χρησιμοποιούμε την **Transform.LookAt** η οποία περιστρέφει το αντικείμενο ώστε να κοιτάει το στόχο που θέτουμε. Ενδιαφερόμαστε να στρέφεται προς τον στόχο μόνο στον άξονα των x και των z, στους οποίους εισάγουμε τη θέση του στόχου, και εισάγουμε τη θέση του βίκινγκ ως είσοδο στον άξονα των y καθώς δεν θέλουμε να αλλάξει ύψος.

Αν ο πιγκουίνος τον πλησιάσει ακόμα περισσότερο, σε μια απόσταση **attackRange**, τότε καλείται η συνάρτηση **attack()** με την οποία ο βίκινγκ τρέχει κατά πάνω του και τον κηνυγάει στο χώρο. Η υλοποίηση γίνεται με την **Transform.Translate** η οποία μετακινεί το αντικείμενο προς την κατεύθυνση και απόσταση που ορίζουμε. Εμείς θέλουμε να μετακινηθεί προς τα εμπρός χρησιμοποιώντας την Vector3.forward με μια ταχύτητα **moveSpeed** ομαλά με την πάροδο του χρόνου.

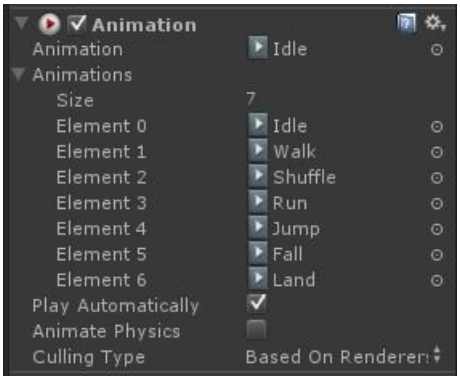
```
function Update ()
{
	if(!gothit)
	{	//the distance between the viking and the player
```

```
                var distance = Vector3.Distance(target.position, myTransform.position);
                animation.CrossFade("Idle");
                if( distance < lookAtDistance )
                {
                        lookAt ();
                        animation.CrossFade("Shuffle");
                }
                if( distance < attackRange )
                {
                        attack ();
                        animation.CrossFade("Run");
                }
        }
        if(AndroidMoveAround.isHit)
        {
                animation.CrossFade("Idle");
        }
        if(Treasure.winGUI1)
        {
                animation.Stop();       //if the penguin won the viking stops chasing him
        }
}

function lookAt ()
{       // Rotates the transform so the forward vector points at target's current position.
        myTransform.LookAt(Vector3(target.position.x, myTransform.position.y, target.position.z));
}

function attack ()
{
        if(!AndroidMoveAround.isHit)
        {       //Moves the transform in the forward direction
                myTransform.Translate(Vector3.forward * moveSpeed * Time.deltaTime);
        }
}
```

Ο βίκινγκ σταματάει να τον κυνηγάει μόνο αν τον πιάσει και τον σκοτώσει, ή αν ο ήρωας απομακρυνθεί αρκετά από αυτόν, ή αν σκοτωθεί από μια χιονόμπαλα, ή αν (σε ευκολότερες πίστες) φτάσει ο ήρωας στο θησαυρό. Με τη συνάρτηση OnTriggerEnter() ελέγχουμε πότε ήρθε σε επαφή με τον ήρωα ή με κάποια χιονόμπαλα. Αν τον πετύχει με χιονόμπαλα την καταστρέφουμε και καλείται η συνάρτηση **SpinAround()** με την οποία ενεργοποιείται η περιστροφή του βίκινγκ γύρω από τον εαυτό του μέσω της LateUpdate() με τον ίδιο τρόπο που στριφογυρίζουμε τον πιγκουίνο όταν τον χτυπάει ένα βλήμα. Όσο

περιστρέφεται, φωνάζει και εξαφανίζεται, και τον κάνουμε τον αόρατο, δεν τον καταστρέφουμε.

Για τα animation της κίνησής του, έχουμε προσαρτήσει το συστατικό Animation στο οποίο θέσαμε εφτά διαφορετικά animation με αντιπροσωπευτικά ονόματα. Τα animation αυτά τα καλούμε με το όνομά τους μέσω κώδικα, χωρίς να χρειάζεται να θέσουμε τιμές παραμέτρων για να εισαχθούμε σε αντίστοιχες καταστάσεις όπως κάναμε με τον πιγκουίνο. Η **Animation.CrossFade** εξασθενίζει το animation με το όνομα που εισάγουμε πάνω από ένα χρονικό διάστημα δευτερολέπτων για να εισαχθούμε σε άλλο animation. Η **Animation.Stop** διακόπτει το animation που παίζει επαναφέροντάς το στην αρχή, η **Animation.Play** παίζει το animation χωρίς καμία ανάμειξη, η **Animation.wrapMode** καθορίζει πως πρέπει να χρησιμοποιηθεί το χρόνικό διάστημα πέρα από το εύρος αναπαραγωγής του κλιπ, και η **WrapMode.Once** διακόπτει το animation όταν φτάσει στο τέλος του χρόνου του.




```
function OnTriggerEnter( hit : Collider )
{
	if(hit.gameObject.tag == "Player")
	{
		animation["Land"].wrapMode = WrapMode.Once;
		animation.Play("Land");
	}
	if(hit.gameObject.tag == "penguinProjectile")
	{
		if(!vikingDown)
		{
			SpinAround();
			Destroy(hit.gameObject);	//we destroy the the object that hit the viking
							//("hit" refers to the projectile)
			vikingDown = true;
		}
	}
}
function SpinAround()
{
	gothit = true;			//cannot attack or walk anymore because he got hit
	spinAround = true;		//start spinning because he got hit
}
function LateUpdate()
{
```

```
        if(spinAround)          //we check if we got hit
        {
        animation.Stop();
        if(tumbleSpeed < 1)     //if he stops spinning around we need to set everything back to default
                                // (safety net)
                {
                        //he is not hit anymore
                        tumbleSpeed = backup[0];
                        decreaseTime = backup[1];
                        decayTime = backup[2];
                        gameObject.active = false;      //we dont destroy the child because we want
                                                        //to use it again
                        spinAround = false;             //has stopped spinning around
                }
                else    //if he is  still spinning around then we need to rotate the character
                {
                        //he's hit! Spin the character around
                        transform.Rotate(0,tumbleSpeed * Time.deltaTime,0, Space.World);
                        //rotate in the world space using only the y axis
                        tumbleSpeed = tumbleSpeed - decreaseTime;//slowly stops rotating over time
                        decreaseTime += decreaseTime;           //it makes the decreaseTime bigger, so
                        // each time it goes faster down to zero and that gives an illusion of animation
                }
        }
}
```

Ο βίκινγκ είναι παιδί ενός κενού GameObject με το οποίο τον επαναφέρουμε στη θέση του κατά την επανάληψη της πίστας με το χάσιμο μιας ζωής, μέσω του κώδικα **ReactivateVikingMobile.js**. Επειδή όμως κινείται στο χώρο, αποθηκεύουμε τη θέση του την ώρα που ξεκινάει η πίστα σε μια μεταβλητή **vikingPos** για να μπορέσουμε να τον επαναφέρουμε σε αυτή. Ο ήχος που αναπαράγεται όταν πεθαίνει βγάζοντας μια κραυγή, ρυθμίζεται από εδώ καθώς με την απενεργοποίηση του βίκινγκ κατά το θάνατό του θα διακοπτόταν ο ήχος. Μετά την περιστροφή του βίκινγκ και την εξασφάλιση ότι απενεργοποιήθηκε από την πίστα, ο αριθμός των βίκινγκ στην πίστα μειώνεται κατά ένα. Η αφαίρεση του συνολικού αριθμού των βίκινγκς γίνεται εδώ, γιατί αν τον αφαιρέσουμε κατά ένα, όταν χτυπηθεί από μια χιονόμπαλα, όσο χρόνο περιστρέφεται πριν εξαφανιστεί μπορεί να δεχθεί και άλλες χιονόμπαλες και να αφαιρεθεί ξανά ο αριθμός των βίκινγκς. Έτσι φροντίζουμε να αφαιρεθεί μία μόνο φορά κατά ένα εφόσον εξαφανιστεί, και αυτό μπορεί να γίνει μόνο από εδώ. Κατά την επαναφορά του στην πίστα ενεργοποιείται πάλι και αρχικοποιούμε όλες τις τιμές των μεταβλητών.

```
#pragma strict
```

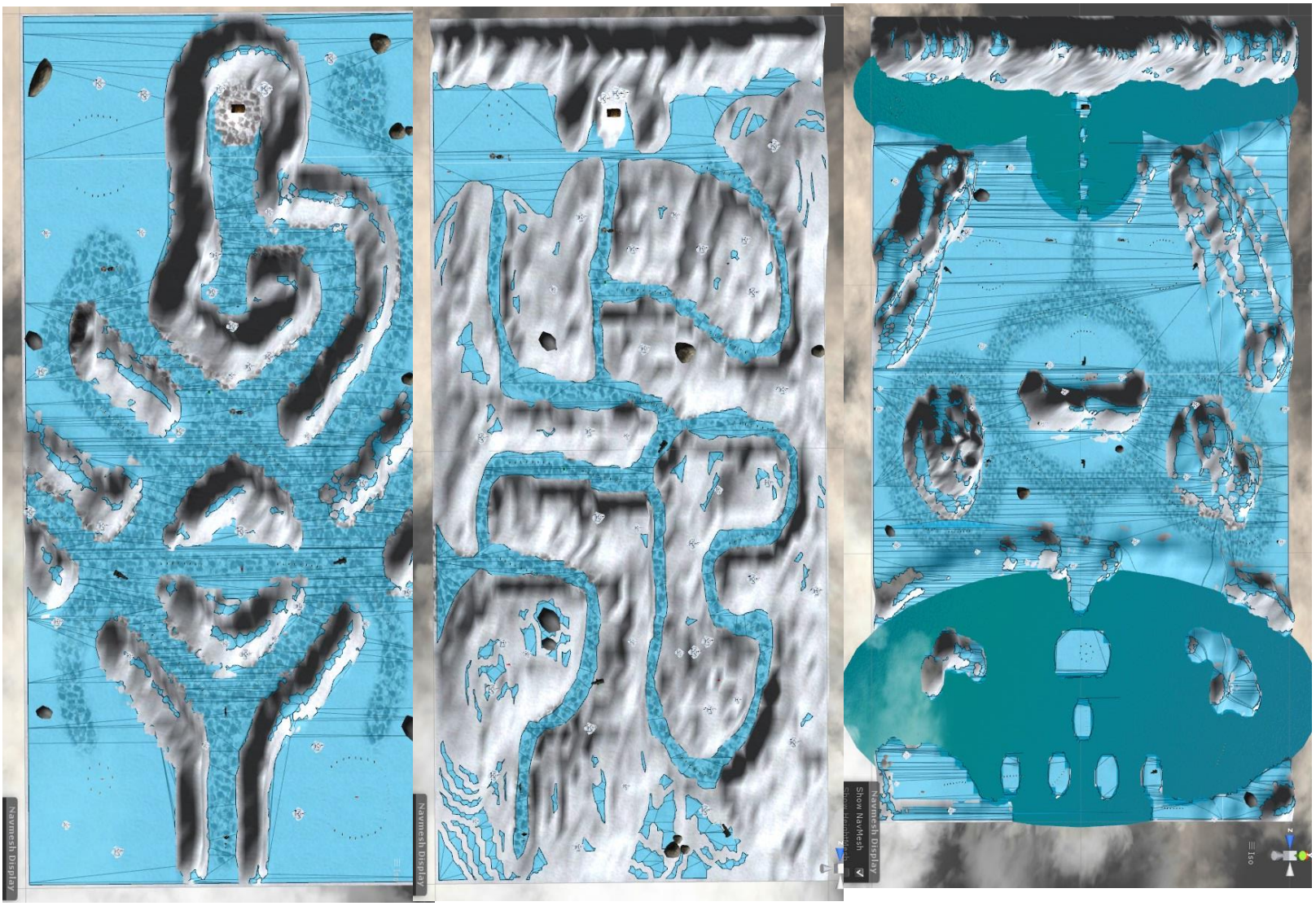
```
private var vikingPos : Vector3;                    //the position of the viking
private var oneLess : int;                          //an integer to erase from the number of the vikings
function Start ()
{
        vikingPos = transform.position;
        oneLess = 1;
}

var VikingDieSound : AudioClip;                     //sound when the viking dies
var tempVikings : int;              //to save the number of the vikings that are still alive in the game

function Update () {
        for (var child : Transform in transform) {  // do whatever we want with child transform here
                if(AIFollowerMobile.vikingDown)      //if the child viking is hit by one snowball
                {
                        audio.PlayOneShot(VikingDieSound);  //scream of the viking when he dies
                        //decrease the total number of the vikings by one
                        AndroidMoveAround.VIKINGS -= oneLess;
                        oneLess = 0;                        //so we cannot decrease the vikings anymore
                        //save the temporary nymber of the existing vikings
                        tempVikings = AndroidMoveAround.VIKINGS;
                        AIFollowerMobile.vikingDown = false;
                                        //then make again the viking vulnerable
                        CheckNumber();
                }
        }
}

function CheckNumber()        //to check if we have alive vikings in the game to enable, because
when //we killed one of them, no one else can attack without enable them again
{
        if(tempVikings > 0)                         //if there are more vikings to be killed
        {
                yield WaitForSeconds(3);            //wait for 3 seconds
                AIFollowerMobile.gothit = false;                //enable them
                oneLess = 1;                        //now we can decrease the number of the
                                    //vikings again by one(only one viking can be killed at a time)
        }
}

function LateUpdate()
{
for (var child : Transform in transform) {        // do whatever we want with child transform here
```

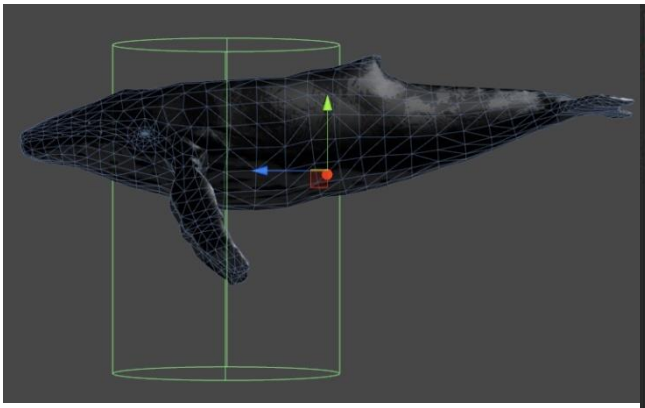
```
        if(AndroidMoveAround.dead)          //when the player dies, everything must be reboot
        {
                AIFollowerMobile.vikingDown = false;//double check that the viking can be hit again
                child.position = vikingPos;             //replacce the viking at his initial position
                child.active = true;                    //make him active again (visible)
                oneLess = 1;                    //the number of the vikings can again be decreased
                AIFollowerMobile.gothit = false;
                                //enable the viking's functions because he is not hit yet
        }
}
}
```

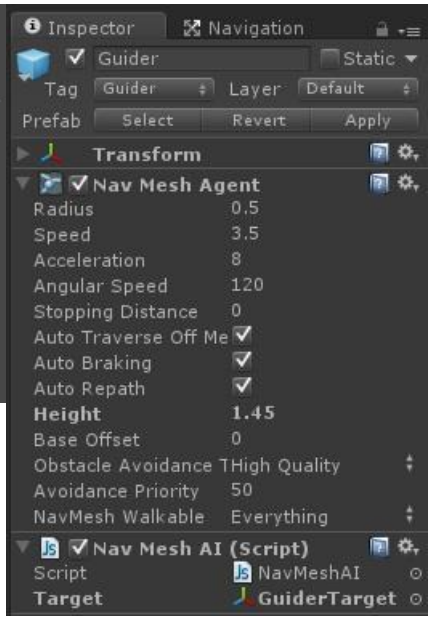
5.5.10 Υλοποίηση φάλαινας με τεχνητή νοημοσύνη

Η φάλαινα οδηγεί τον ήρωα στο θησαυρό από τη βέλτιστη επιτρεπτή διαδρομή της πίστας, δηλαδή τη συντομότερη, χρησιμοποιώντας την τεχνητή νοημοσύνη που προσφέρεται από τη Unity. Δεν συναντάει εμπόδια στο δρόμο της εκτός από τον ήρωα που ίσως βρεθεί μπροστά της. Δεν τον αποφεύγει όμως γιατί μπορεί να πηγαίνει συνεχώς μπροστά της και να μην μπορεί να φτάσει ποτέ στο στόχο της. Για να μην εμποδίζεται από αυτόν, δεν τις έχει προστεθεί collider, έτσι ώστε να περνάει μέσα από τον collider του πιγκουίνου και να μην έρχεται σε σύγκρουση μαζί του.

Για την υλοποίηση της πλοήγησης-Navigation διαβάζουμε το έδαφος της κάθε πίστας ξεχωριστά, δημιουργώντας πλέγματα πλοήγησης, τα navigation meshes ή αλλιώς navmeshes, με μια διαδικασία baking, στην οποία εισάγουμε το έδαφος που μας ενδιαφέρει και ορίζουμε τις παραμέτρους που θέλουμε να ληφθούν υπόψιν κατά την πλοήγηση του χαρακτήρα. Το πλέγμα φαίνεται σαν μια γαλάζια διαφάνεια πάνω από το έδαφος.







Για να μπορέσουν τα δεδομένα αυτά να διαβαστούν από τον χαρακτήρα, και να μπορέσει να πλοηγηθεί μέσα στην πίστα ακολουθώντας το στόχο του και αποφεύγοντας τα εμπόδια, πρέπει να του εφαρμόσουμε το συστατικό **NavMesh Agent**. Ο NavMesh Agent είναι ένας κύλινδρος γύρω από τον χαρακτήρα, σαν collider, ο οποίος πρέπει να ακουμπάει το έδαφος για να μπορεί να κινηθεί. Η φάλαινά μας κολυμπάει, και στον αέρα και στο νερό, και ανεβοκατεβαίνει στον αέρα

όσο προχωράει, όπως και τα αντικείμενα συλλογής. Αν προσαρτήσουμε στη φάλαινα τον NavMeshAgent, όσο ανεβαίνει προς τα πάνω θα χάνει επαφή από το έδαφος. Έτσι θέσαμε τη φάλαινα παιδί ενός κενού GameObject στο οποίο συνδέσαμε τον NavMesh Agent σε τέτοιο ύψος, ώστε να περιλαμβάνει τη φάλαινα που βρίσκεται στον αέρα και να ακουμπάει το έδαφος, οργανώνοντας την συμπεριφορά του μέσω του script **NavMeshAI.js**.

Για να ξεκινήσει η φάλαινα να κατευθύνεται προς το στόχο πρέπει ο πιγκουίνος να βρίσκεται κοντά της, μέσα σε μια συγκεκριμένη απόσταση. Αν η απόσταση μεταξύ τους ξεπεράσει τα 20 μέτρα της Unity, σταματάει και τον περιμένει να την πλησιάσει για να συνεχίσει να κινείται. Την απόσταση μεταξύ της φάλαινας και του πικουίνου την υπολογίζουμε με την **Vector3.Distance.**

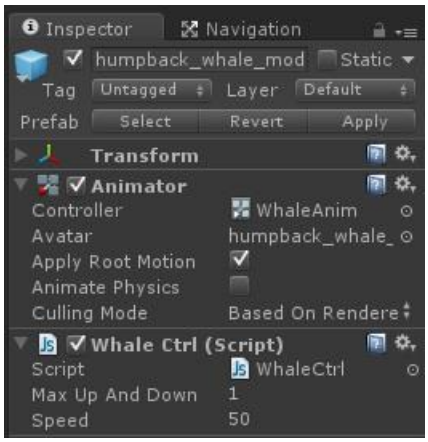
Για να αποκτήσουμε πρόσβαση στο συστατικό NavMeshAgent χρησιμοποιούμε την **GetComponent()**. Με την **NavMeshAgent.destination** ορίζουμε το σημείο προορισμού για τον πράκτορα στον οποίο θα κατευθυνθεί. Θα σχεδιαστεί αυτόματα το μονοπάτι προς τον προορισμό και ο πράκτορας θα αρχίσει να κινείται προς την κατεύθυνση αυτή. Το σημείο προορισμού μπορεί επίσης να αλλάξει και να διαβαστεί ενώ ο πράκτορας κινείται.

Όσο ο πιγκουίνος βρίσκεται κοντά στη φάλαινα, ο στόχος της είναι το σεντούκι του θυσαυρού, οπότε και κατευθύνεται προς αυτό. Αν απομακρυνθεί από αυτήν, θέλουμε η φάλαινα να παραμείνει στο σημείο που την άφησε μέχρι να ξανά γυρίσει και να συνεχίσει πάλι να κινείται προς το στόχο. Όσο η φάλαινα μένει σταθερή, θέτουμε ως στόχο τον εαυτό της, και όταν θέλουμε πάλι να συνεχίσει να κινείται επαναφέρουμε τον αρχικό στόχο.

```
#pragma strict
/*
	Script that uses the NavMesh and the NavMeshAgent to make the guider of the player
	find the shortest path for the treasure with AI following
*/
var target : Transform;

function Update ()
{	//to calculate the distance between the penguin and the whale
	var distance = Vector3.Distance(gameObject.Find("PenguinHero").transform.position,
                                                gameObject.transform.position);
	if( distance <= 20)
	{	///set the destination of the character to the player's position
		GetComponent(NavMeshAgent).destination = target.position;
		//Destination point for the agent to navigate towards
	}
	else
	{
```

```
                GetComponent(NavMeshAgent).destination = transform.position;
        }
}
```

Για να κάνουμε τη φάλαινα να ανεβοκατεβαίνει από το σημείο που βρίσκεται, είτε στον αέρα είτε στο νερό, και κολυμπάει καθόλη τη διάρκεια του παιχνδιού. Η ρύθμιση της κίνησής της γίνεται μέσω του **WhaleCtrl.js** το οποίο το συνδέουμε μόνο στο αντικείμενο της φάλαινας, και όχι στο GameObject με το συστατικό του πράκτορα. Για την απόδοση της κίνησης τη συνδέουμε το συστατικό Animator με ένα μόνο animation για το κολύμπι, όπου και βρίσκεται διαρκώς σε αυτήν την κατάσταση. Η κίνηση πάνω και κάτω από το σημείο που βρίσκεται γίνεται με τον ίδιο τρόπο που ρυθμίσαμε την ίδια κίνηση για τα αντικείμενα συλλογής.




```
function FixedUpdate()
{
   angle += speed * Time.deltaTime;
   if (angle > 270) angle -= 360;
   transform.localPosition.y = startHeight + maxUpAndDown * (1 + Mathf.Sin(angle * toDegrees)) / 2;
        //Mathf.Sin : Returns the sine of angle f in radians.
}
```

Κάθε φορά που ο ήρωας χάνει μια ζωή και επιστρέφει στο αρχικό του σημείο, επιστρέφει και η φάλαινα στο αρχικό της σημείο για να ξαναδείξει το δρόμο. Για αυτό το λόγο κατά την ενεργοποίηση του κώδικα αποθηκεύουμε τη θέση της σε μια μεταβλητή. Για να επαναφέρουμε στη θέση αυτή και τη φάλαινα αλλά και το κενό GameObject με το NavMesh Agent για να μπορέσει να κινηθεί πάλι, τα θέτουμε και τα δύο ως παιδιά ενός άλλου κενού GameObject στο οποίο συνδέσαμε το script **ReactivateWhale.js**. Με τον κώδικα αυτόν τοποθετούμε και τα δύο του παιδιά στην αρχική του θέση όπως κάναμε και στην περίπτωση του βίκινγκ και του κανονιού.

```
#pragma strict
private var whalePos : Vector3;                    //the position of the whale guider
function Start ()
{
        whalePos = transform.position;
}

function LateUpdate()
{
        for (var child : Transform in transform) {    // do whatever we want with child transform here
                if(AndroidMoveAround.dead)  //when the player dies, everything must be reboot
                {
```

```
                    child.position = whalePos;      //replacce the viking at his initial position
                }
            }
}
```

5.5.11 Υλοποίηση αντικειμένων Prefab

Κατά τη διάρκεια του παιχνιδιού δημιουργούνται υπό συνθήκες νέα κλωνοποιημένα αντικείμενα, όπως οι χιονόμπαλες που πετάει ο ήρωας, τα βλήματα που εκτοξεύονται από τα κανόνια, οι εκρήξεις που δημιουργούνται στη θέση που πεθαίνουν τα πιγκουινάκια ή ο ήρωας, και στη θέση του κανονιού. Έχουμε δημιουργήσει το κάθε αντικείμενο από αυτά μία φορά, και το κλωνοποιούμε με την Instantiate όσες φορές χρειαστεί.

Πρέπει όμως μετά από κάποιο συγκεκριμένο χρόνο να καταστρέψουμε το κλωνοποιημένο αντικείμενο για να μην επιβαρύνεται το παιχνίδι μας και να μην γεμίσει η μνήμη με έναν τεράστιο αριθμό αντικειμένων. Για αυτό τα συνδέουμε με τον κώδικα **KillMeOverTime.js**, ο οποίος τους δίνει διάρκεια ζωής ενός δευτερολέπτου από τη στιγμή της δημιουργίας τους.

```
var lifeTime = 1.0;             //1 second
function Awake ()               // the function is trigged when a prefab or object is created
{
    Destroy (gameObject, lifeTime); //it destroys the gameObject the script is attached to
    //(it reffers where is the script attached to, the time in seconds that the snowball has to live)
}
```

Για την υλοποίηση των εκρήξεων δημιουργήσαμε δύο διαφορετικά GameObjects με το κατάλληλο υλικό και υφή αναλόγως την έκρηξη, αλλά και στα δύο για την προσωμοίωση της έκρηξης μέσω εφέ, προσαρτήσαμε τα συστατικά **EllipsoideParticle Emitter**, **Particle Animator** και **Particle Renderer**.

5.5.12 Υλοποίηση χειριστηρίου

Για τον χειρισμό της κάθετης κίνησης του πιγκουίνου μπροστά και πίσω στον άξονα z και της περιστροφής του γύρω από τον άξονα y, χρησιμοποιούμε στο κάτω αριστερό μέρος της οθόνης ένα χειριστήριο αφής. Αναλόγως την κίνηση του δαχτύλου του χρήστη πάνω στο χειριστήριο, μετακινείται ελάχιστα προς την αντίστοιχη κατεύθυνση ώστε να δωθεί η ψευδαίσθηση ενός πραγματικού χειριστηρίου και να υποδείξει την κατεύθυνση κίνησης και του ήρωα. Με την κάθετη κίνηση του δαχτύλου πάνω στο χειριστήριο ο ήρωας κινείται αντίστοιχα προς τα μπροστά ή προς τα πίσω, ενώ με την οριζόντια κίνηση στρίβει γύρω από τον εαυτό του προς την αντίστοιχη κατεύθυνση. Σε ενδιάμεσες θέσεις κίνησης του δαχτύλου, συνδυάζονται οι κινήσεις, δηλαδή προχωράει και στρίβει ταυτόχρονα προς την κατεύθυνση που κινούμε το δάχτυλό μας.

Αποτελείται από ένα GUITexture που περιλαμβάνει την εικόνα του χειριστηρίου, και το ελέγχουμε από το script **Joystick.js**. Το κινητό αυτό χειριστήριο μπορεί να χειριστεί εισόδους αφής, αριθμούς και καταστάσεις αφής. Οι νεκρές ζώνες μπορούν να ελέγξουν πότε πρέπει να ομαλοποιηθούν οι εισόδοι του χειριστηρίου. Ένα **TouchPad** επιτρέπει στο δάχτυλο να ακουμπήσει την οθόνη σε οποιοδήποτε σημείο και να παρακολουθεί την σχετική κίνηση χωρίς να κινείται το γραφικό στοιχείο.

Για ένα χειριστήριο είναι υποχρεωτική η ύπαρξη του GUITexture, για αυτό δηλώνουμε με τη **RequireComponent** την αυτόματη προσθήκη του συστατικού αυτού για να αποφεύγονται σφάλματα. Στην κλάση **Boundery** δηλώνουμε σε μεταβλητές **min** και **max**, τα μέγιστα και τα ελάχιστα όρια στα οποία μπορεί να κινηθεί το χειριστήριο.

Αρχικά ορίζουμε μια Boolean μεταβλητή **touchPad** για να ελέγχουμε αν το χειριστήριο θα λειτουργεί ως TouchPad δηλαδή θα είναι ημιδιάφανο όσο δεν το αγγίζουμε και θα έχει έντονο χρώμα όσο το ακουμπάμε και θα παραμένει σταθερό στο σημείο που έχουμε ορίσει, ή αν θα έχει έντονο χρώμα καθόλη τη διάρκεια και θα ακολουθεί την κίνηση του δαχτύλου πάνω στα όρια αφής του χειριστηρίου. Επίσης ορίζουμε μια **Rect** μεταβλητή **touchZone** με την οποία θα καθορίσουμε την περιοχή αφής του χειριστηρίου, μια μεταβλητή **deadZone** για να καθορίσουμε την περιοχή εκτός εμβέλειας της αφής του χειριστηρίου, και μια δισδιάστατη μεταβλητή **position** για να καθορίσουμε τη θέση στους άξονες x και y.

Επιπλέον ορίζουμε και μια μεταβλητή **lastFingerId** που περιέχει το δάχτυλο που χρησιμοποιήθηκε τελευταίο για το χειριστήριο και είναι αρχικά ίση με -1 όταν δεν υπάρχει δάχτυλο, μια δισδιάστατη μεταβλητή **fingerDownPos** για τον υπολογισμό της θέσης του δαχτύλου στην οθόνη, μια μεταβλητή **fingerDownTime** για τον υπολογισμό του χρόνου που βρίσκεται το δάχτυλο στο χειριστήριο, και μια μεταβλητή **firstDeltaTime** για ομαλοποίηση του χρόνου.

Τέλος, ορίζουμε μια GUITexture μεταβλητή **gui** για να θέσουμε την εικόνα του χειριστηρίου, μια Rect μεταβλητή **defaultRect** ως ορθογώνιο για να θέσουμε τη προεπιλεγμένη θέση της εικόνας του χειριστηρίου, τη **guiBoundary** που περιλαμβάνει τα όρια της εικόνας του χειριστηρίου από την έξοδο της κλάσης Boundary, τη δισδιάστατη **guiTouchOffset** για την είσοδο αφής στην εικόνα του χειριστηρίου, και τη δισδιάστατη **guiCenter** για το κέντρο της εικόνας του του χειριστηρίου.

Στη συνάρτηση Start() αποκτάμε πρόσβαση στο GUITexture με την **GetComponent**, ώστε να μπορέσουμε να το κινήσουμε, και το αποθηκεύουμε στη μεταβλητή gui. Στη μεταβλητή defaultRect αποθηκεύουμε το προεπιλεγμένο ορθογώνιο-rect για το gui, ώστε να μπορούμε να επανερχόμαστε σε αυτό. Έτσι αποθηκεύουμε την αρχική θέση του χειριστηρίου στην οθόνη ως προς το ύψος στον άξονα των y και το πλάτος στον άξονα των x, και το μέγεθος

και τη θέση των pixel gui.pixelInset. Αφού τα αποθηκεύσουμε, μηδενίζουμε τη θέση του στον άξονα των x και y.

Μπορούμε να επιλέξουμε αν το χειριστήριο θα φαίνεται στην οθόνη και θα κινείται ανάλογα με το δάχτυλο, θέτοντας τη μεταβλητή **touchPad** στον Inspector της Unity ψευδή, ή αν θα είναι ημιδιάφανο και θα γίνεται έντονο όταν το πατάμε στην οθόνη αλλά δε θα κουνιέται, θέτοντάς την αληθή. Αν είναι αληθής και έχει οριστεί εικόνα, αποθηκεύουμε το ορθογώνιο των δεδομένων ύψους και πλάτους defaultRect στην touchZone για να ορίσουμε την περιοχή αφής του χειριστηρίου. Αν η touchPad είναι ψευδής θα κουνιέται το χειριστήριο, και ορίζουμε την περιοχή αφής guiTouchOffset καθορίζοντας το πλάτος, το ύψος, το κέντρο guiCenter αφού δεν αλλάζει, και τα μέγιστα και ελάχιστα όρια guiBoundary για να περιορίσουμε την κίνηση του χειριστηρίου. Όταν τερματίζει το παιχνίδι απενεργοποιούμε το χειριστήριο.

```
#pragma strict
/*
	Joystick creates a movable joystick (via GUITexture) that  handles touch input, taps, and
	phases. Dead zones can control  where the joystick input gets picked up and can be
	normalized. We can enable the touchPad property from the editor to treat this Joystick as a
	TouchPad. A TouchPad allows the finger to touch down at any point and it tracks the
	movement relatively without moving the graphic
*/
@script RequireComponent( GUITexture )

// A simple class for bounding how far the GUITexture will move
class Boundary
{
	var min : Vector2 = Vector2.zero;
	var max : Vector2 = Vector2.zero;
}

var touchPad : boolean;					// Is this a TouchPad?
var touchZone : Rect;
var deadZone : Vector2 = Vector2.zero;		// Control when position is output
var position : Vector2;					// [-1, 1] in x,y
private var lastFingerId = -1;				// Finger last used for this joystick
private var fingerDownPos : Vector2;
private var fingerDownTime : float;
private var firstDeltaTime : float = 0.5;
private var gui : GUITexture;				// Joystick graphic
private var defaultRect : Rect;			// Default position / extents of the joystick graphic
private var guiBoundary : Boundary = Boundary();	// Boundary for joystick graphic
private var guiTouchOffset : Vector2;			// Offset to apply to touch input
```

```
private var guiCenter : Vector2;                    // Center of joystick

function Start()
{
        // Cache this component at startup instead of looking up every frame
        gui = GetComponent( GUITexture );

        // Store the default rect for the gui, so we can snap back to it
        defaultRect = gui.pixelInset;

         defaultRect.x += transform.position.x * Screen.width;
        defaultRect.y += transform.position.y * Screen.height;

        transform.position.x = 0.0;
        transform.position.y = 0.0;

        if ( touchPad )
        {
                // If a texture has been assigned, then use the rect ferom the gui as our touchZone
                if ( gui.texture )
                        touchZone = defaultRect;
        }
        else
        {
                // This is an offset for touch input to match with the top left corner of the GUI
                guiTouchOffset.x = defaultRect.width * 0.5;
                guiTouchOffset.y = defaultRect.height * 0.5;

                // Cache the center of the GUI, since it doesn't change
                guiCenter.x = defaultRect.x + guiTouchOffset.x;
                guiCenter.y = defaultRect.y + guiTouchOffset.y;

                // Let's build the GUI boundary, so we can clamp joystick movement
                guiBoundary.min.x = defaultRect.x - guiTouchOffset.x;
                guiBoundary.max.x = defaultRect.x + guiTouchOffset.x;
                guiBoundary.min.y = defaultRect.y - guiTouchOffset.y;
                guiBoundary.max.y = defaultRect.y + guiTouchOffset.y;
        }
}

function Disable()
{
        gameObject.active = false;
}
```

Όταν ο χρήστης σηκώσει το δάχτυλο και αφήσει το χειριστήριο, αυτό γυρίζει στην αρχική του θέση μέσω της συνάρτησης **ResetJoystick()**, θέτοντας την εικόνα gui.pixelInset στη θέση της στα πλαίσια του ορθογωνίου αφής defaultRect, και μηδενίζοντας τη θέση της position στον άξονα x και y. Επίσης αρχικοποιούμε και τη μεταβλητή lastFingerId στο -1 καθώς δεν υπάρχει πλέον δάχτυλο στο χειριστήριο, και τη fingerDownPos μηδενίζοντας τη θέση του δαχτύλου στην οθόνη. Αν έχουμε επιλέξει να λειτουργεί το χειριστήριο ως touchpad κάνουμε αχνό το χρώμα του όταν δεν το αγγίζουμε μέσω της **Color.a** θέτοντας μια συγκεκριμένη τιμή αχνότητας.

```
function ResetJoystick()
{
	// Release the finger control and set the joystick back to the default position
	gui.pixelInset = defaultRect;
	lastFingerId = -1;
	position = Vector2.zero;
	fingerDownPos = Vector2.zero;

	if ( touchPad )
		gui.color.a = 0.025;
}
```

Στην συνάρτηση Update() θέτουμε στη μεταβλητή **count** τον αριθμό των αγγιγμάτων αφής. Αν είναι ίση με μηδέν, δηλαδή δεν υπάρχει κανένα άγγιγμα στο χειριστήριο, το επαναφέρουμε στην αρχική του θέση καλώντας τη συνάρτηση ResetJoystick(). Αλλιώς για κάθε άγγιγμα, θέτουμε στη Touch μεταβλητή **touch** την **Input.GetTouch** που επιστρέφει την κατάσταση της συγκεκριμένης επαφής που εισάγουμε, στη δισδιάστατη μεταβλητή **guiTouchPos** θέτουμε τη διαφορά θέσης του αγγίγματος από την αρχική θέση αγγίματος του χειριστηρίου guiTouchOffset, και ορίζουμε μια Boolean μεταβλητή **shouldLatchFinger** για να γνωρίζουμε πότε σύρεται το δάχτυλο.

Αν έχουμε επιλέξει να λειτουργεί ως touchPad και αν το άγγιγμα στην οθόνη touch.position βρίσκεται εντός των ορίως αφής του χειριστηρίου touchZone, θέτουμε την shouldLatchFinger αληθή. Αν δεν λειτουργεί ως touchpad αλλά το άγγιγμα στην οθόνη βρίσκεται πάνω στην περιοχή της εικόνας του χειριστηρίου, το οποίο το γνωρίζουμε μέσω της GUIElement.HitTest, θέτουμε την shouldLatchFinger αληθή.

Αν σέρνεται το δάχτυλο στην οθόνη και αυτό είναι ένα καινούργιο άγγιγμα, τότε αν έχουμε επιλέξει να λειτουργεί το χειριστήριο ως touchPad, κάνουμε έντονο το χρώμα του κατά την επιλογή του δίνοντας μια συγκεκριμένη τιμή στο χρώμα του, θέτουμε στην lastFingerId την εύρεση της αφής με την fingerId του συγκεκριμένου αγγίγματος, θέτουμε στην fingerDownPos τη θέση του αγγίγματος, και στην fingerDownTime το χρόνο που διαρκεί το άγγιγμα.

Αν δεν είναι καινούργιο άγγιγμα, αλλά το έχουμε εντοπίσει από πριν με την **touch.fingerId**, αν έχουμε επιλέξει να λειτουργεί ως touchpad ορίζουμε την τωρινή θέση που βασίζεται άμεσα με την απόσταση από την αρχική θέση στην οθόνη touchdown. Για τον άξονα x, υπολογίσαμε τη διαφορά της τωρινής θέσης του δαχτύλου touch.position.x από την αρχική στον άξονα των x fingerDownPos.x, διαιρώντας το με το μισό πλάτος της περιοχής αφής του χειριστηρίου touchZone.width. Για τον άξονα y, υπολογίσαμε τη διαφορά της τωρινής θέσης του δαχτύλου από την αρχική στον άξονα των y, διαιρώντας το με το μισό ύψος της περιοχής αφής του χειριστηρίου touchZone. Για τον υπολογισμό της θέση στον κάθε άξονα position.x και position.y, εισάγουμε το αντίστοιχο αποτέλεσμα στην **Mathf.Clamp** για να πάρουμε μια τιμή μεταξύ του -1 και του 1.

Αν δεν λειτουργεί ως touchpad, αλλάζουμε τη θέση της εικόνας του χειριστηρίου για να ταιριάζει με το άγγιγμα. Στην θέση και το μέγεθος της εικόνας στον άξονα των x gui.pixelInset.x, θέτουμε την θέση του δαχτύλου στην εικόνα του χειριστηρίου στον άξονα των x guiTouchPos.x, δίνοντας μια τιμή μεταξύ του ελάχιστου και του μέγιστου ορίου σε αυτόν τον άξονα μέσω της Mathf.Clamp. Αντίστοιχα στη θέση και το μέγεθος της εικόνας στον άξονα των y gui.pixelInset.y, θέτουμε την θέση του δαχτύλου στην εικόνα του χειριστηρίου στον άξονα των y guiTouchPos.y, δίνοντας μια τιμή μεταξύ του ελάχιστου και του μέγιστου ορίου σε αυτόν τον άξονα.

Αν το άγγιγμα τελείωσε TouchPhase.Ended ή ακυρώθηκε TouchPhase.Canceled, καλείται η συνάρτηση ResetJoystick() για να αρχικοποιήσουμε το χειριστήριο.

Αν δεν έχει επιλεχθεί να λειτουργεί το χειριστήριο ως touchpad, για τον υπολογισμό της θέσης παίρνουμε μια τιμή μεταξύ του -1 και του 1 με βάση την τοποθεσία του γραφικού στοιχείου του χειριστηρίου για τον κάθε άξονα. Στη θέση και το μέγεθος της εικόνας του χειριστηρίου gui.pixelInset.x προσθέτουμε τη θέση αφής πάνω στο χειριστήριο guiTouchOffset.x αφαιρώντας το κέντρο του χειριστηρίου guiCenter.x. Το αποτέλεσμα το διαιρούμε με τη θέση αφής πάνω στο χειριστήριο guiTouchOffset.x και το θέτουμε στη θέση που ορίζουμε για το χειριστήριο στον άξονα x position.x. Αντίστοιχα για τον άξονα y, στη θέση και το μέγεθος της εικόνας του χειριστηρίου gui.pixelInset.y προσθέτουμε τη θέση αφής πάνω στο χειριστήριο guiTouchOffset.y αφαιρώντας το κέντρο του χειριστηρίου guiCenter.y. Το αποτέλεσμα το διαιρούμε με τη θέση αφής πάνω στο χειριστήριο guiTouchOffset.y και το θέτουμε στη θέση που ορίζουμε για το χειριστήριο στον άξονα y position.y.

Παίρνουμε τις απόλυτες τιμές των θέσεων αυτών μέσω της Mathf.Abs και τις θέτουμε στις μεταβλητές **absoluteX** και **absoluteY** για να ορίσουμε τη νεκρή ζώνη στην οποία δεν θα ανταποκρίνεται το χειριστήριο. Για τον κάθε άξονα ελέγχουμε αν η τιμή αυτή είναι μικρότερη από την τιμή της νεκρής ζώνης σε αυτόν τον άξονα. Για τον άξονα x, αν η absoluteX είναι μικρότερη από την deadZone.x τότε το χειριστήριο είναι εντός της νεκρής ζώνης και δεν

δεχόμαστε εισόδους τιμών θέτοντας τη θέση στον άξονα x μηδέν. Αντίστοιχα για τον άξονα y, αν η absoluteY είναι μικρότερη από την deadZone.y τότε το χειριστήριο είναι εντός της νεκρής ζώνης και δεν δεχόμαστε εισόδους τιμών θέτοντας τη θέση στον άξονα y μηδέν.

```
function Update()
{
	var count = Input.touchCount;

	if ( count == 0 )
		ResetJoystick();
	else
	{
		for(var i : int = 0;i < count; i++)
		{
			var touch : Touch = Input.GetTouch(i);
			var guiTouchPos : Vector2 = touch.position - guiTouchOffset;

			var shouldLatchFinger = false;
			if ( touchPad )
			{
				if ( touchZone.Contains( touch.position ) )
					shouldLatchFinger = true;
			}
			else if ( gui.HitTest( touch.position ) )
			{
				shouldLatchFinger = true;
			}

			// Latch the finger if this is a new touch
			if (shouldLatchFinger&& (lastFingerId == -1 || lastFingerId != touch.fingerId ))
			{
				if ( touchPad )
				{
					gui.color.a = 0.15;
					lastFingerId = touch.fingerId;
					fingerDownPos = touch.position;
					fingerDownTime = Time.time;
				}

				lastFingerId = touch.fingerId;
			}
			if ( lastFingerId == touch.fingerId )
			{
				if ( touchPad )
```

```
                                {       // For a touchpad, let's just set the position directly based on
                                        // distance from initial touchdown
                                        position.x = Mathf.Clamp( ( touch.position.x -
                                                fingerDownPos.x ) / ( touchZone.width / 2 ), -1, 1 );
                                        position.y = Mathf.Clamp( ( touch.position.y -
                                                fingerDownPos.y ) / ( touchZone.height / 2 ), -1, 1 );
                                }
                                else
                                {
                        // Change the location of the joystick graphic to match where the touch is
                                        gui.pixelInset.x =  Mathf.Clamp( guiTouchPos.x,
                                                        guiBoundary.min.x, guiBoundary.max.x );
                                        gui.pixelInset.y =  Mathf.Clamp( guiTouchPos.y,
                                                        guiBoundary.min.y, guiBoundary.max.y );
                                }
                                if ( touch.phase == TouchPhase.Ended ||
                                                        touch.phase == TouchPhase.Canceled )
                                        ResetJoystick();
                        }
                }
        }
        if ( !touchPad )
        {
                // Get a value between -1 and 1 based on the joystick graphic location
                position.x = ( gui.pixelInset.x + guiTouchOffset.x - guiCenter.x ) / guiTouchOffset.x;
                position.y = ( gui.pixelInset.y + guiTouchOffset.y - guiCenter.y ) / guiTouchOffset.y;
        }
        // Adjust for dead zone
        var absoluteX = Mathf.Abs( position.x );
        var absoluteY = Mathf.Abs( position.y );

        if ( absoluteX < deadZone.x )
        {
                // Report the joystick as being at the center if it is within the dead zone
                position.x = 0;
        }
        if ( absoluteY < deadZone.y )
        {
                // Report the joystick as being at the center if it is within the dead zone
                position.y = 0;
        }
}
```

5.6 Υλοποίηση της JumpLand

5.6.1 Υλοποίηση περιστροφής

Η JumpLand είναι κατάλληλα σχεδιασμένη για κινητές συσκευές σε όρθια θέση, με το κεντρικό κουμπί προς τα κάτω. Μόλις ξεκινάει μια πίστα αυτού του κόσμου, ελέγχεται πρώτα η θέση της συσκευής μέσω του κώδικα **TiltControlJump.js,** τον οποίο έχουμε προσαρτήσει σε ένα GUITexture. Το παιχνίδι θα ξεκινήσει μόνο αν η συσκευή είναι σε όρθια θέση και το κουμπί βρίσκεται προς τα κάτω. Αν βρίσκεται σε οποιαδήποτε άλλη όρθια θέση, δηλαδή σε το κεντρικό κουμπί προς τα πάνω, ή προς τα δεξιά, ή προς τα αριστερά, τότε δεν θα μπορεί να παίξει ο χρήστης.

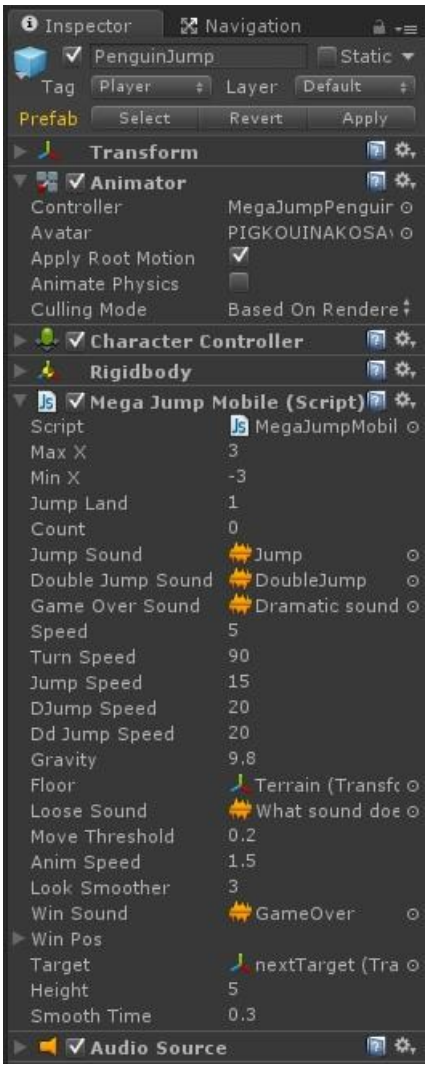
Αν ο χρήστης αφήσει τη συσκευή παράλληλα προς το έδαφος με την οθόνη προς τα πάνω ή προς τα κάτω, το παιχνίδι σταματάει και εμφανίζεται μια εικόνα που ενημερώνει τον χρήστη να σηκώσει τη συσκευή προς τα πάνω για να συνεχιστεί το παιχνίδι. Για να εμφανιστεί αυτή η εικόνα ορίζουμε μια Texture2D μεταβλητή FaceUpTexture. Στη συνάρτηση Start() θέτουμε την οθόνη **Screen.orientation** στη θέση Portrait με την **ScreenOrientation.Portrait** ώστε να καταλαβαίνει ο χρήστης για τη θέση που πρέπει να κρατήσει τη συσκευή, και απενεργοποιούμε το GUITexture, ώστε να μην εμφανίζεται κάποια εικόνα μηνύματος. Ορίζουμε μια στατική Boolean μεταβλητή **canPlay** στην οποία έχουν πρόσβαση όλες οι κλάσεις του παιχνιδιού, και γίνεται αληθής μόνο όταν η συσκευή βρίσκεται στην θέση Portrait για να παίξουμε. Σε οποιαδήποτε άλλη θέση είναι ψευδής και το παιχνίδι βρίσκεται σε αναμονή.

Στη συνάρτηση Update() ελέγχουμε κάθε καρέ στην θέση της συσκευής μέσω της **Input.deviceOrientation** διαβάζοντας τις καταστάσεις της **DeviceOrientation**. Όσο η συσκευή είναι σε θέση Portrait, απενεργοποιούμε την εικόνα, ενεργοποιούμε το χρόνο του παιχνιδιού αν ο χρήστης δεν έχει επιλέξει παύση, είτε με το κουμπί παύσης είτε τοποθετώντας ταυτόχρονα δύο δάχτυλα πάνω στην οθόνη, και θέτουμε την canPlay αληθή για να μπορούμε να παίξουμε. Όσο η συσκευή βρίσκεται σε θέση PortraitUpsideDown, ή LandscapeLeft, ή LandscapreRight διατηρούμε απενεργοποιήμενη την εικόνα μηνύματος, και την οθόνη περιστραμένη σε θέση Portrait ώστε να φαίνεται πια είναι η σωστή θέση με την οποία πρέπει να κρατηθεί η συσκευή, θέτουμε την canPlay ψευδή και σταματάμε το χρόνο για να μην μπορεί να συνεχιστεί το παιχνίδι. Όσο βρίσκεται σε θέση FaceUp ή FaceDown ορίζουμε την εικόνα που θα εμφανιστεί σε όλη την έκταση της οθόνης με ένα μήνυμα που ενημερώνει τον χρήστη να σηκώσει τη συσκευή προς τα πάνω. Οι διαστάσεις της εικόνας υπολογίζονται αναλόγως την οθόνη της κάθε συσκευής, ως προς τη θέση της και το μέγεθός της στους άξονες x και y. Επίσης διακόπτουμε το παιχνίδι θέτοντας την canPlay ψευδή και σταματώντας το χρόνο. Μόλις ο χρήστης σηκώσει τη συσκευή σε σωστή θέση το παιχνίδι θα συνεχιστεί από το σημείο που σταμάτησε ενεργοποιώντας πάλι το χρόνο και κάνοντας αληθή την canPlay.

```
#pragma strict
/*
	DeviceOrientation : Describes physical orientation of the device as determined by the OS.
	If device is physically situated between discrete positions, as when (for example) rotated
	diagonally, system will report Unknown orientation.
*/
var FaceUpTexture : Texture2D;

function Start()
{
	Screen.orientation = ScreenOrientation.Portrait;		//only then the game starts
	guiTexture.enabled = false;
}

static var canPlay : boolean = false;

function Update ()
{
	//Portrait: The device is in portrait mode, with the device held upright and the home button
	//at the bottom.
	if (Input.deviceOrientation == DeviceOrientation.Portrait)
	{
		//Pause the game
		guiTexture.enabled = false;				//don't show anything
		if(PauseTouchJump.pauseNow == false && MegaJumpMobile.PauseFingers == false)
		{
			Time.timeScale = 1;
		}
		canPlay = true;
	}
	//PortraitUpsideDown: The device is in portrait mode but upside down, with the device held
	//upright and the home button at the top.
	//LandscapeLeft: The device is in landscape mode, with the device held upright and the home
	//button on the right side.
	//LandscapeRight: The device is in landscape mode, with the device held upright and the
	//home button on the left side.
	else if (Input.deviceOrientation == DeviceOrientation.PortraitUpsideDown ||
		Input.deviceOrientation == DeviceOrientation.LandscapeLeft ||
			Input.deviceOrientation == DeviceOrientation.LandscapeRight)
	{
		//Pause the game
		guiTexture.enabled = false;	//it doesnt start until we rotate it on the portrait side
		Screen.orientation = ScreenOrientation.Portrait;
		canPlay = false;
```

```
                Time.timeScale = 0;
        }
        //FaceUp: The device is held parallel to the ground with the screen facing upwards.
        //FaceDown: The device is held parallel to the ground with the screen facing downwards.
        else if (Input.deviceOrientation == DeviceOrientation.FaceUp ||
                Input.deviceOrientation == DeviceOrientation.FaceDown)
        {
                //Pause the game
                guiTexture.pixelInset.x = -Screen.width/2;
                guiTexture.pixelInset.y = -Screen.height/2;
                guiTexture.pixelInset.width = Screen.width;
                guiTexture.pixelInset.height = Screen.height;
                guiTexture.enabled = true;
                guiTexture.texture = FaceUpTexture;
                canPlay = false;
                Time.timeScale = 0;
        }
}
```

5.6.2 Υλοποίηση του πιγκουίνου

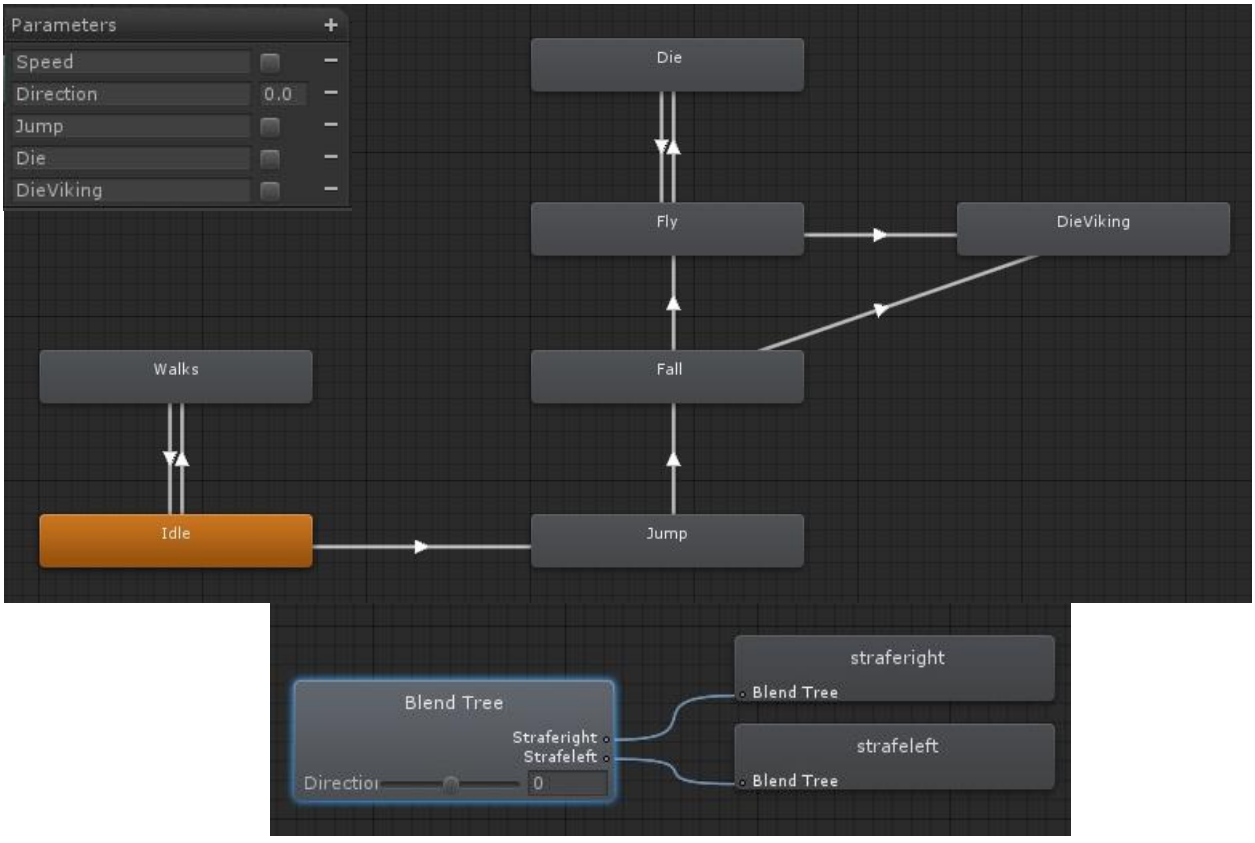


Ο πιγκουίνος στην JumpLand είναι πιο απλός σε σχέση με αυτόν της SnowLand και της WaterLand. Αποτελείται από το συστατικό Transform για το καθορισμό της θέσης, της περιστροφής και του μεγέθους του στο χώρο, από το συστατικό Animator για την υλοποίηση των animations της συμπερφοράς του, από τον Character Controller για να ελέγξουμε την κίνησή του, από το Rigidbody για να προσωμοιωθεί η φυσική στις κινήσεις του, από την Audio Source για την αναπαραγωγή ήχων, και τον κώδικα **MegaJumpMobile.js** με τον οποίο ελέχουμε όλη τη συμπεριφορά του.

Οι κινήσεις του ήρωα είναι περιορισμένες. Κοιτάει προς την κάμερα και βρίσκεται στο κέντρο της οθόνης. Μπορεί να προχωρήσει μόνο δεξιά και αριστερά εντός στα όρια της οθόνης, καθώς η κάμερα δεν τον ακολουθεί στον οριζόντιο άξονα. Η μετακίνησή του στον άξονα των x γίνεται με την κλίση της συσκευής δεξιά και αριστερά αντίστοιχα. Για να πετάξει αρκεί να πατήσει ο χρήστης την οθόνη μια φορά με το δάχτυλό του σε οποιοδήποτε σημείο, όπου ασκείται στον πιγκουίνο μια μεγάλη δύναμη για να πηδήξει αρκετά ψηλά. Επειδή όμως διαθέτει Rigidbody, η βαρύτητα θα τον τραβήξει πάλι στο έδαφος. Για να συνεχίσει να ανεβαίνει

προς τα πάνω, πρέπει να συλλέξει τα διάφορα αντικείμενα που αιωρούνται κατά το ύψος της πίστας, όπως τα νομίσματα, τα ψάρια μπόνους και τα μαγικά σμαράγδια, τα οποία του δίνουν μια παραπάνω δύναμη για να τον ωθήσουν ακόμα πιο ψηλά. Όταν σταματήσει να τα συλλέγει η βαρύτητα τον τραβάει προς τα κάτω και πεθαίνει. Για να κερδίσει πρέπει να φτάσει στα σύννεφα τερματισμού αποφεύγοντας τους βίκινκς μετακινώντας τον με την κλίση της συσκευής.

Ο Animator ελέγχεται από το **MegaJumpPenguin** animator controller, ο οποίος είναι απλός, περιλαμβάνοντας λιγότερες καταστάσεις animations και παραμέτρους σε σχέση με τους προηγούμενους δύο κόσμους. Όσο βρίσκεται στο έδαφος, περιμένει μέχρι ο χρήστης να κάνει κάτι σε κατάσταση **Idle**. Αν δώσει κλίση στη συσκευή, προχωράει δεξιά ή αριστερά αντίστοιχα κοιτώντας πάντα προς την κάμερα. Η κίνηση αυτή ελέγχεται από το Blend Tree **Walks** στο οποίο εισάγεται όταν ο πιγκουίνος έχει κάποια ταχύτητα **Speed** και δεν είναι ακίνητος. Περιλαμβάνει δύο καταστάσεις, αυτήν που περπατάει προς τα αριστερά **strafeleft** και αυτή που περπατάει προς τα δεξιά **straferight**, στις οποίες εισάγεται αναλόγως την τιμή της παραμέτρου **Direction** που καθορίζεται από την κατεύθυνση της κίνησης. Μόλις πηδήξει ενεργοποιείται το animation του άλματος **Jump** και ετοιμάζεται να πετάξει **Fall**, και αν συλλέξει κάποιο αντικείμενο θα πετάει με το animation πτήσης **Fly**. Αν πέσει πάνω σε κάποιον βίκινγκ θα ενεργοποιηθεί το animation με το οποίο πεθαίνει στριφογυρνώντας στον αέρα κατεβαίνοντας προς τα κάτω **DieViking**. Αν απλά σταματήσει να συλλέγει αντικείμενα ενεργοποιείται άλλο animation **Die** με το οποίο πεθαίνει πέφτοντας αργά στο κενό.



Ορίζουμε μια μεταβλητή **speed** για την ταχύτητα με την οποία κινείται πάνω στο έδαφος, την **jumpSpeed** με την οποία πηδάει, την **dJumpSpeed** με την οποία πηδάει κάθε φορά πιο ψηλά συλλέγοντας νομίσματα ή ψάρια, την **ddJumpSpeed** για να του δωθεί μεγάλη ώθηση με το μαγικό σμαράγδι, τη **gravity** με την οποία τον τραβάει η βαρύτητα προς τα κάτω, την **floor** στην οποία θέτουμε το έδαφος για να μπορούμε να το χειριστούμε, το **maxX** και **minX** για να ορίσουμε τα όρια στα οποία μπορεί να κινηθεί δεξιά και αριστερά μέσα στην οθόνη, το **moveThreshold** για να θέσουμε το όριο το οποίο αν ξεπεράσει η είσοδος του επιταχυνσιόμετρου θα αρχίσει ο ήρωας να κινείται, την **movex** για να ορίσουμε την κίνησή του, και την **iPx** για να ορίσουμε τις εισόδους του επιταχυνσιόμετρου για να ελέγξουμε την κίνηση. Στη συνάρτηση Start() αρχικοποιούμε όλες τις μεταβλητές και καταστάσεις του πιγκουίνου, ελέγχουμε αν επιτρέπεται η αναπαραγωγή του ήχου αναλόγως την επιλογή του χρήστη στο μενού, και αποκτάμε πρόσβαση στα συστατικά που τον αποτελούν.

```
private var controller : CharacterController;
private var anim : Animator;                    // a reference to the animator on the character
function Start ()
{
        controller = GetComponent(CharacterController);
        anim = GetComponent(Animator);
}
```

Στη συνάρτηση Update() ελέχουμε αν η συσκευή βρίσκεται στη σωστή θέση, και μόνο αν η canPlay είναι θετική μπορούμε να παίξουμε. Περιστρέφουμε τον ήρωα 180° με την **Quaternion.Euler** ώστε να κοιτάει προς την κάμερα και να μην έχει πλάτη προς αυτήν.

Ελέγχουμε την κίνηση του ήρωα στον άξονα x με τις εισόδους του επιταχυνσιόμετρου. Θέτουμε την movex ίση με μηδέν και την iPx ίση με την κίνηση του επιταχυνσιόμετρου της συσκευής στον άξονα x μέσω της **Input.acceleration** η οποία μας δίνει την τελευταία μετρούμενη γραμμική επιτάχυνση της συσκευής στον τρισδιάστατο χώρο. Αν η απόλυτη τιμή που παίρνουμε από το επιταχυνσιόμετρο iPx μέσω της Mathf.Abs είναι μεγαλύτερη από την τιμή του ορίου moveThreshold που έχουμε ορίσει για να ξεκινήσει να κινείται ο ήρωας, θέτουμε τη **movex** μέσω της **Mathf.Sign** ίση με 1 αν η τιμή της εισόδου του επιταχυνσιομέτρου iPx είναι θετική ή μηδέν, και ίση με -1 αν η iPx είναι αρνητική.

Σε μια τρισδιάστατη μεταβλητή **vel** καθορίζουμε την κίνηση μόνο δεξιά και αριστερά στον άξονα x μέσω της **Transform.right** πολλαπλασιασμένο με την movex και την ταχύτητα της κίνησης.

Αν ο ήρωας βρίσκεται στο έδαφος και δεν έχει πηδήξει ακόμα, και αν η movex είναι αρνητική ή θετική σημαίνει ότι ο ήρωας περπατάει, οπότε ενεργοποιούμε το animation περπατήματος θέτοντας αληθή την παράμετρο Speed, ενώ αν είναι ίση με μηδέν σημαίνει ότι δεν περπατάει και θέτουμε την Speed ψευδή για να σταματήσει το animation περπατήματος

και να γυρίσει στο Idle. Αν δεν έχει πατηθεί το κουμπί παύσης μπορεί ο ήρωας να πηδήξει. Θέτουμε την μεταβλητή της ταχύτητας του άλματος vSpeed ίση με μηδέν και ελέγχουμε σε κάθε άγγιγμα τη φάση στην οποία βρίσκεται μέσω της **Input.touches**. Για κάθε άγγιγμα που δεν έχει τελειώσει **TouchPhase.Ended** και δεν έχει ακυρωθεί **TouchPhase.Canceled** αυξάνουμε κατά ένα τη μεταβλητή **jumpcount**. Αν η μεταβλητή αυτή είναι μεγαλύτερη του μηδενός, σημαίνει ότι ο χρήστης έχει αγγίξει την οθόνη, οπότε ο πιγκουίνος πηδάει θέτοντας στην vSpeed την ταχύτητα άλματος jumpSpeed, σταματάμε το περπάτημα θέτοντας την Speed ψευδή, ενεργοποιούμε το animation άλματος θέτοντας την παράμετρο Jump αληθή, ενεργοποιούμε τον ήχος άλματος, και ενημερώνουμε ότι ο πιγκουίνος έχει πηδήξει θέτοντας την **hasJumped** αληθή.

Αν έχει πηδήξει, δηλαδή αν η hasJumped είναι αληθής, ο χρήστης μπορεί να διακόψει το παιχνίδι τοποθετώντας ταυτόχρονα δύο δάχτυλα πάνω στην οθόνη. Ορίζουμε μια μεταβλητή **fingerCount** για να υπολογίσουμε τον αριθμό των δαχτύλων που βρίσκονται στην οθόνη, και την θέτουμε αρχικά ίση με μηδέν. Για κάθε άγγιγμα ελέγχουμε τη φάση στην οποία βρίσκεται. Αν το άγγιγμα δεν έχει τελειώσει ή ακυρωθεί αυξάνουμε την fingerCount κατά ένα. Αν η fingerCount είναι ίση με 2 σημαίνει ότι υπάρχουν δύο δάχτυλα πάνω στην οθόνη, οπότε διακόπτουμε το παιχνίδι σταματώντας το χρόνο και θέτοντας αληθή τη μεταβλητή **PauseFingers** η οποία μας εισάγει στο μενού της παύσης.

Η μεταβλητή **jumpHigher** είναι αληθής όποτε ο ήρωας συλλέξει κάποιο αντικείμενο, είτε νομίσματα είτε ψάρι μπόνους. Αν ο πιγκουίνος έχει συλλέξει κάποιο από αυτά και η jumpHigher είναι αληθής, και δεν έχει χτυπηθεί από βίκινγκ προστίθεται μια δύναμη dJumpSpeed στην ταχύτητα άλματος vSpeed ώστε να ανέβει πιο ψηλά.

Η μεταβλητή **dJumpHigher** είναι αληθής όποτε ο ήρωας συλλέξει κάποιο μαγικό σμαράγδι. Αν η dJumpHigher είναι αληθής και δεν έχει χτυπηθεί από κάποιον βίκινγκ onViking, προστίθεται μια μεγάλη δύναμη ddJumpSpeed στην vSpeed ώστε να ανέβει προς τα πάνω με μεγάλη ταχύτητα και για μεγάλη απόσταση. Αν κατά τη διάρκεια που έχει ταχύτητα ddJumpSpeed πάρει και άλλα νομίσματα, προστίθεται κάθε φορά πάλι η ταχύτητα dJumpSpeed στην ταχύτητά του vSpeed. Παίρνοντας το μαγικό σμαράγδι καλείται η συνάρτηση **DoubleSize()** με την οποία διπλασιάζεται το μέγεθός του.

Για να μην αιωρείται για πάντα και να μην έχει μόνιμη επιτάχυνση προς τα πάνω με τα αντικείμενα που συλλέγει και την ταχύτητα που του προστίθεται, εφαρμόζουμε βαρύτητα στον πιγκουίνο. Από την ταχύτητά του vSpeed αφαιρούμε κάθε καρέ την τιμή της βαρύτητας που έχουμε ορίσει με τη μεταβλητη gravity ανά δευτερόλεπτο που χρειάστηκε να ολοκληρωθεί το τελευταίο καρέ. Το αποτέλεσμα το θέτουμε στον άξονα y της μεταβλητής vel που ορίζει την κίνηση του πιγκουίνου. Την κίνηση την αποδίδουμε μέσα από τον

CharacterController με την **CharacterController.Move** εισάγοντάς της την μεταβλητή vel πολλαπλασιασμένη με το χρόνο ανά δευτερόλεπτο.

Επίσης χρειάζεται να περιορίσουμε την κίνηση του ήρωα στον άξονα x μεταξύ δύο ορισμένων σημείων. Στη μεταβλητή **pos** θέτουμε τη θέση του πιγκουίνου με την **transform.position**. Στον άξονα x της μεταβλητής αυτής, pos.x, θέτουμε τον άξονα x μεταξύ ενός ελάχιστου σημείο minX και ενός μέγιστου maxX μέσω της Mathf.Clamp. Έπειτα θέτουμε τη μεταβλητή αυτή στη θέση του ήρωα.

Με την **Vector3.Distance** υπολογίζουμε την απόσταση του πιγκουίνου από το έδαφος. Αν ο ήρωας έχει πηδήξει από το έδαφος, και δεν έχει κερδίσει και δεν έχει πέσει πάνω σε κάποιον βίκινγκ, αν η ταχύτητα άλματος vel.y είναι μικρότερη του -0,5, δηλαδή έχει αρχίσει και πέφτει προς το έδαφος, ενεργοποιείται το Die animation με το οποίο ειδοποιούμε το χρήστη ότι θα πεθάνει ο ήρωάς του. Αν εξακολουθεί και πέφτει και η vel.y είναι μικρότερη του -11, τότε χάνεται από την οθόνη καθώς έχει πέσει κάτω και πεθαίνει, ενεργοποιείται ο ήχος ήττας του παιχνιδιού, σταματάμε το χρόνο, και δονείται το κινητό με την Handheld.Vibrate() κατά τη μετάβαση του animation για να καταλάβει ο χρήστης ότι έχασε. Αν καταφέρει να μαζέψει κάποιο αντικείμενο καθώς πέφτει πριν όμως πέσει αρκετά, τότε ξανά παίρνει ώθηση και συνεχίζει το παιχνίδι και απενεργοποιούμε το Die animation.

Αν φτάσει στα μεγάλα σύννεφα, ο πιγκουίνος σταματάει να πετάει και τον τοποθετούμε ακίνητο μία μονάδα πιο πάνω από τη θέση που είχε όταν τα άγγιξε στον άξονα y ώστε να τον κρύψουμε μέσα στα σύννεφα και να μην αιωρείται στον αέρα.

Αν ακουμπήσει κάποιον βίκινγκ ενώ δεν έχει πάρει το μαγικό σμαράγδι θα πεθάνει και δεν θα μπορεί να συλλέξει κανένα άλλο αντικείμενο καθώς πέφτει. Ενεργοποιείται ένας ήχος κραυγής του πιγκουίνου καθώς πέφτει, κάνουμε αρνητική την ταχύτητά του ώστε να πέσει γρήγορα προς το έδαφος, ενεργοποιούμε το animation DieViking με το οποίο στριφογυρίζει στον αέρα θέτοντας την παράμετρό του αληθή, υπολογίζουμε το χρόνο για να εμφανιστεί σε λίγα δευτερόλεπτα η καρτέλα τέλος παιχνιδιού, και σταματάμε το χρόνο του παιχνιδιού.

```
function Update ()
{
if(TiltControlJump.canPlay)
{
        //ROTATE
        transform.rotation = Quaternion.Euler(0,180,0);          //so he can't rotate while moving
        // MOVE left / right
        movex = 0;
        iPx = iPhoneInput.acceleration.x;     //get the movement from the accelerometer of the mobile
         if (Mathf.Abs(iPx) > moveThreshold)  //put a threshold to move the character if we pass it off
        {//Return value is 1 when iPx is positive or zero, -1 when f is negative. just like forward.right
```

```
            movex = Mathf.Sign(iPx);
        }
        var vel: Vector3 = -transform.right * movex* speed;      //move only the side direction
        if (!hasJumped)//(controller.isGrounded)                 // We are grounded, so recalculate
        {
            if(movex <0 || movex  >0)                            //move left or right
            {
                anim.SetBool("Speed", true);
            }
            if(movex  == 0)
            {
                anim.SetBool("Speed", false);                    //idle state
            }
            if(PauseTouchJump.pauseNow == false)
            {       //if we haven't press the pause button we can jump
                //JUMP
                vSpeed = 0;       // grounded character has vSpeed = 0
                for (var touch : Touch in Input.touches) {
              //we can jump when the player touches the screen, it is enough if he touches it once
                    if (touch.phase != TouchPhase.Ended &&
                        touch.phase != TouchPhase.Canceled)
                        jumpcount++;
                    //Each entry represents a status of a finger touching the screen.
                }
                if (jumpcount > 0)                  //User has touched the screen, so Jump
                {
                    vSpeed = jumpSpeed;
                    anim.SetBool("Speed", false);
                    anim.SetBool("Jump", true);
                    if(PlayerPrefs.GetInt("Sound")==1)//sound is on from the main menu
                    {
                        audio.PlayOneShot(JumpSound);
                    }
                    playJumpSound = true;
                    hasJumped = true;
                }
            }
        }
        if(hasJumped)//if we have already jumped we can pause the game by put two finger on screen
        {
            //PAUSE IF TWO FINGER ARE ON THE SCREEN
            var fingerCount = 0;
            for (var touch : Touch in Input.touches) {
```

```
                if (touch.phase != TouchPhase.Ended && touch.phase !=
                                                        TouchPhase.Canceled)
                        fingerCount++;
                //Each entry represents a status of a finger touching the screen.
        }
        if (fingerCount == 2)           //User has 2 fingers touching the screen
        {
                Time.timeScale = 0;
                PauseFingers = true;    //Pause
        }
}
if(jumpHigher && onViking == false)                 //power jump for each coin
{
        vSpeed = dJumpSpeed;
        jumpHigher = false;
}
if(dJumpHigher && onViking == false)                //power up with the pick up power
{
        vSpeed = ddJumpSpeed;
        if(jumpHigher )                                     //power jump for each coin
        {
                vSpeed = dJumpSpeed;
                jumpHigher = false;
        }
        DoubleSize();
        dJumpHigher = false;
}
// apply gravity acceleration to vertical speed:
vSpeed -= gravity * Time.deltaTime;
vel.y = vSpeed; // include vertical speed in vel
// convert vel to displacement and Move the character:
controller.Move(vel * Time.deltaTime );
var pos = transform.position;
// clamp new position:
pos.x = Mathf.Clamp(pos.x, minX, maxX);
// and assign it to the object:
transform.position = pos;
//the distance between the player and the floor
var distance = Vector3.Distance(floor.position, transform.position);

if(hasJumped && (stopFlying == false) && (onViking == false))
{
        if(vel.y < -0.5)
        {
```

```
                        anim.SetBool("Die", true);                  //when he starts falling
                        if(vel.y < -11)   //if he starts falling for a while and no coin is been collected
                        {
                                hasLost = true;
                                if(PlayerPrefs.GetInt("Sound")==1)//sound is on from the main menu
                                {
                                        audio.PlayOneShot(GameOverSound);
                                }
                                anim.SetBool("Die", false);
                                Time.timeScale = 0.0;               //stop time
                        }
                }
                else
                {
                        anim.SetBool("Die", false);
                }
        }
        if(stopFlying)                                      //if we reached the clouds
        {
                transform.position.y = winPos.y + 1;
        }
        if(onViking && (cannotDie == false))   //we hit on a viking and if we do not have any power
                                // up we die, and cannot collect any other coins or powers or fishes
        {
                if(PlayerPrefs.GetInt("Sound")==1)//sound is on from the main menu
                {
                        audio.PlayOneShot(LooseSound);
                }
                vSpeed = -4;
                anim.SetBool("DieViking", true);
                count = count + 1;
                if(count >= 80)             //if he starts falling for a while and no coin is been collected
                {
                        hasLost = true;
                        count =0;
                        Time.timeScale = 0.0;               //stop time
                }
        }
}
}
```

Σε μεταβλητές θέτουμε τις καταστάσεις των animations με τα ονόματά τους, για να μπορούμε να τις ελέγξουμε. Στη συνάρτηση FixedUpdate() αν η συσκευή είναι στη σωστή θέση, αρχικοποιούμε τις παραμέτρους που μας εισάγουν στις καταστάσεις των animations, όταν βρισκόμαστε σε αυτά και δεν είναι στη φάση της μετάβασης **Animator.IsInTransition**,

ώστε να μπορέσουμε να εισαχθούμε ξανά σε αυτά. Στη μεταβλητή currentBaseState θέτουμε την παρούσα κατάσταση του animation στην οποία βρισκόμαστε, και στη μεταβλητή **h** θέτουμε την movex ώστε να ξέρουμε αναλόγως την κατεύθυνση της κίνησης Direction ποιό animation περπατήματος να ενεργοποιούμε.

Όταν πεθαίνει χάνοντας ύψος όπου ενεργοποιείται το animation Die, θέλουμε να δονηθεί το κινητό για να καταλάβει ο χρήστης ότι έχασε. Ενεργοποιούμε τη δόνηση μόνο στο στάδιο της μετάβασης, για να μην έχουμε συνεχόμενη δόνηση αλλά για μία φορά μόνο.

```
//ANIMATION
// a reference to the current state of the animator, used for base layer
private var currentBaseState : AnimatorStateInfo;
// these integers are references to our animator's states  and are used to check state for various actions
//to occur
static var dieJState : int = Animator.StringToHash("Base Layer.Die");
//.............More code here……..//
function FixedUpdate ()
{
if(TiltControlJump.canPlay)
{
	var h : float = movex ;
	// set our animator's float parameter 'Direction ' equal to the vertical input axis
	anim.SetFloat("Direction", h);
	// set our currentState variable to the current state of the Base Layer (0) of animation
	currentBaseState = anim.GetCurrentAnimatorStateInfo(0);
	if (currentBaseState.nameHash == dieJState)
	{	//  ..and not still in transition..
		if(!anim.IsInTransition(0))		//Animation Transitions define what happens
				//when you switch from one Animation State to another. There can be
				// only one transition active at any given time.
		{
			anim.SetBool("Die", false);
		}
		else	//in transition
		{
			Handheld.Vibrate ();				// vibrate the device
		}
	}	//.............More code here……..//
}
```

Στη συνάρτηση OnTriggerEnter() αν η συσκευή είναι στη σωστή θέση ελέγχουμε με ποιά αντικείμενα έρχεται σε επαφή ο πιγκουίνος για να ενεργοποιοήσουμε τις αντίστοιχες ενέργειες. Αν ο πιγκουίνος πάρει το μαγικό σμαράγδι γίνεται αθάνατος, και θέτουμε αληθή τη μεταβλητή **cannotDie** με την οποία ελέγχουμε αν μπορεί να πεθάνει από τους βίκινγκς ή όχι.

Αν πάρει ένα νόμισμα αυξάνουμε κατά ένα τη μεταβλητή **coinsJ** για να βρούμε το συνολικό αριθμό των νομισμάτων που συνέλεξε στο τέλος του παιχνιδιού για την απόδοση της βαθμολογίας. Για κάθε ψάρι μπόνους που συλλέγει γίνεται θετική η αντίστοιχη μεταβλητή, **FishJ1**, **FishJ2**, ή **FishJ3**, και προστίθενται 100 βαθμοί για το πρώτο ψάρι, 200 για το δεύτερο, και 300 για το τρίτο, στους πόντους των ψαριών για την απόδοση της βαθμολογίας.

Αν φτάσει στα σύννεφα τερματισμού, αποθηκεύουμε στη μεταβλητή **winPos** τη θέση με την οποία τερμάτισε για να μπορέσουμε να τον τοποθετήσουμε λίγο πιο ψηλά και να τον κρύψουμε μέσα στα σύννεφα ώστε να μην αιωρείται στον αέρα, σταματάει να πετάει θέτοντας την **stopFlying** ψευδή, ενεργοποιείται η μουσική νίκης, ειδοποιούμε για τη νίκη θέτοντας την **winGUIJ** αληθή ώστε να εμφανιστεί ένα μήνυμα στην οθόνη, και ενημερώνουμε ότι κέρδισε θέτοντας την **hasWon** αληθή ώστε να ξεκλειδώσει η επόμενη πίστα και να αποθηκευτούν τα δεδομένα στο μενού πιστών. Αν είχε διπλάσιο μέγεθος όταν τερμάτισε, τον επαναφέρουμε στο κανονικό του μέγεθος.

Αν ακουμπήσει κάποιον βίκινγκ και δεν είχε πάρει μαγικό σμαράγδι, θέτουμε την **onViking** αληθή και πεθαίνει, και ενεργοποιείται η δόνηση του κινητού μία φορά, για να καταλάβει ο χρήστης ότι έχασε.

```
function OnTriggerEnter( hit : Collider )
{
if(TiltControlJump.canPlay)
{
	//COIN
	if (hit.tag == "Coin")	//if the penguin collects coins we make a sound
	{
		coinsJ++;
	}
	//FISH BONUS
	if(hit.gameObject.tag == "Fish1")			//if the penguin has touched the fish1
	{
		FishJ1 = true;	//we have collect the fish1
		FishJBonus = FishJBonus + 100;
	}
	//WIN
	if(hit.tag == "End")
	{
		winPos = transform.position;
		stopFlying = true;
		if(PlayerPrefs.GetInt("Sound")==1)		//sound is on from the main menu
		{
			audio.PlayOneShot(WinSound);
		}
```

```
                EndCtrlJump.winGUIJ = true;             //show win gui message
                if(cannotDie)
                {
                        gameObject.transform.localScale = Vector3(10,10,10);
                        cannotDie = false;
                        gravity = 9.8;
                }
                hasWon = true;
        }
        //LOOSE
        if(hit.tag == "Viking")
        {
                if(cannotDie == false)
                {
                        onViking = true;
                        Handheld.Vibrate ();                            // vibrate the device
                }
        }
}
}
```

Η συνάρτη DoubleSize() καλείται όταν ο πιγκουίνος πάρει το μαγικό σμαράγδι, στην οποία τοποθετείται λίγο πιο ψηλά από την αρχική του θέση ώστε να φαίνεται στην ίδια θέση κατά την αλλαγή του κέντρου του, και διπλασιάζεται το μέγεθός του για οχτώ δευτερόλεπτα. Έπειτα επανέρχεται στο αρχικό του μέγεθος και θέτουμε πάλι την cannotDie ψευδή ώστε να μπορεί να σκοτωθεί από τους βίκινγκς.

```
function DoubleSize()
{
        gameObject.transform.position.y = gameObject.transform.position.y +2;
        gameObject.transform.localScale = Vector3(20,20,20); //he will have double size
        gravity = 7;
        yield WaitForSeconds(8);
        gameObject.transform.localScale = Vector3(10,10,10);
        cannotDie = false;                                          //he is vulnerable again
        gravity = 9.8;

}
```

Στη συνάρτηση LateUpdate() αν η συσκευή είναι στη σωστή θέση μετακινούμε τον ήρωα προς τα πάνω για να μπορεί να πετάει. Όποτε η jumpHigher είναι αληθής, δηλαδή έχει συλλέξει ένα νόμισμα ή ψάρι, εκτός από τη δύναμη ταχύτητας που του προσθέτουμε, τον κατευθύνουμε προς τα πάνω αλλάζοντας τη θέση του. Έχουμε δημιουργήσει ένα κενό GameObject ως παιδί του ήρωα, πάνω από το κεφάλι του ώστε να τον ακολουθεί συνεχώς, το οποίο το χρησιμοποιούμε για να τον κατευθύνουμε προς τη θέση που βρίσκεται αυτό, μέσω

της **Mathf.SmoothDamp**, η οποία αλλάζει σταδιακά την τιμή της τωρινής θέσης του ήρωας προς τον επιθυμητό στόχο με την πάροδο του χρόνου **smoothTime**. Η τιμή της θέσης του εξομαλύνεται από την τιμή **yVelocity** που ορίζουμε, την οποία ποτέ δεν θα υπερβεί. Οι τιμές αυτές αποθηκεύονται στη μεταβλητή **newPosition** την οποία θέτουμε στον άξονα y της θέσης του ήρωα transform.position, αφήνοντας ίδια τη θέση του στους άλλους άξονες.

Μετά την ολοκλήρωση της κάθε πίστας, μόλις η hasWon γίνει θετική, αποθηκεύουμε τα δεδομένα (αριθμός νομισμάτων και ψάρια μπόνους) με την **PlayerPrefs.Save()** για την απόδοση της βαθμολογίας, και για την εμφάνιση των δεδομένων στο μενού των πιστών. Η βαθμολογία **ScoreCounter** σε αυτόν τον κόσμο δημιουργείται από το άθροισμα του αριθμού των νομισμάτων coinsJ και των ψαριών FishJBonus. Με την **PlayerPrefs.SetInt** τα θέτουμε σε συκεκριμένες μεταβλητές τις οποίες μπορούμε να καλέσουμε με την **PlayerPrefs.GetInt**.

Κάθε φορά που ο χρήστης κάνει νέο ρεκόρ βαθμολογίας, εμφανίζεται η ένδειξη “High Score” με γαλάζια γράμματα. Αν δεν κάνει νέο ρεκόρ, απλά εμφανίζεται η βαθμολογία με κίτρινα γράμματα αναγράφοντας “Score”. Για το λόγο αυτό πρέπει να αποθηκεύουμε πάντα τη μεγαλύτερη βαθμολογία ώστε να μπορούμε να συγκρίνουμε τις τιμές.

```
function LateUpdate()
{
if(TiltControlJump.canPlay)
{
        if(jumpHigher && cannotDie == false)
        {
                var newPosition : float = Mathf.SmoothDamp(transform.position.y, target.position.y+
                                                height, yVelocity, smoothTime);
                transform.position = Vector3(transform.position.x, newPosition,
                                                transform.position.z);
        }
}
        //WIN
        if(hasWon)              //we save the data for each level
        {
                //SNOWLAND
                if(jumpLand == 1)
                {
                        PlayerPrefs.SetInt("WinJump1", 1);
                        //it will automatically check if this value is more then the previous and it will
                        //save automatically on save points.
                 if (PlayerPrefs.GetInt("jCoin1")<coinsJ)
                {
                        PlayerPrefs.SetInt("jCoin1", coinsJ);
                }
```

```
            if(PlayerPrefs.GetInt("jFishBonus1")<FishCtrlJump.fishCounter)//save the fishes
            {
                PlayerPrefs.SetInt("jFishBonus1", FishCtrlJump.fishCounter);
            }
            //to find which fishes we have collected
            if(FishJ1)
            {
                PlayerPrefs.SetInt("j1GotFish1",1);
            }
            else
            {
                PlayerPrefs.SetInt("j1GotFish1",0);
            }
            if(FishJ2)
            {
                PlayerPrefs.SetInt("j1GotFish2",1);
            }
            else
            {
                PlayerPrefs.SetInt("j1GotFish2",0);
            }
            if(FishJ3)
            {
                PlayerPrefs.SetInt("j1GotFish3",1);
            }
            else
            {
                PlayerPrefs.SetInt("j1GotFish3",0);
            }
            //SCORE
            ScoreCounter = coinsJ + FishJBonus ;                    // each coin is 1 point
            if (PlayerPrefs.GetInt("jHighScore1")<ScoreCounter)
            {
                PlayerPrefs.SetInt("jHighScore1", ScoreCounter);
            }
            PlayerPrefs.Save();
        }
}
}
```

5.6.3 Υλοποίηση της κάμερας

Η κάμερα κοιτάει τον ήρωα από μπροστά και τον ακολουθεί μόνο στον κάθετο άξονα y, με μια ομαλή κίνηση που ελέγχεται από το **CameraFollow.js**. Χρησιμοποιούμε πάλι την

Mathf.SmoothDamp με την οποία σταδιακά αλλάζουμε την τιμή της θέσης της στον άξονα y, transform.position.y, προς το ύψος του ήρωα στην πάροδο του χρόνου smoothTime, και θέτουμε την τιμή αυτή στη μεταβλητή newPosition. Στη συνέχεια τη χρησιμοποιούμε για να ορίσουμε τη θέση της κάμερας στον άξονα y.

```
/*
        This camera smoothes out rotation around the y-axis and height. Horizontal Distance to the
        target is always fixed. There are many different ways to smooth the rotation but doing it this
        way gives you a lot of control over how the camera behaves.
        For every of those smoothed values we calculate the wanted value and the current value.
        Then we smooth it using the Lerp function.  Then we apply the smoothed values to the
        transform's position.
*/
var target : Transform;  // The target we are following
var distance = 10.0;      // The distance in the x-z plane to the target
var height = 5.0;         // the height we want the camera to be above the target
// Place the script in the Camera-Control group in the component menu
@script AddComponentMenu("Camera-Control/Smooth Follow")
var smoothTime = 0.3;
private var yVelocity = 0.0;

function LateUpdate () {
        // Early out if we don't have a target
        if (!target)
                return;
        // Set the height of the camera
        var newPosition : float = Mathf.SmoothDamp(transform.position.y, target.position.y+ height,
                                        yVelocity, smoothTime);
        transform.position = Vector3(transform.position.x, newPosition, transform.position.z);
}
```

5.6.4 Υλοποίηση αντικειμένων συλλογής

Τα αντικείμενα που μπορεί να συλλέξει σε αυτό τον κόσμο είναι τα νομίσματα, τα ψάρια μπόνους, και τα μαγικά σμαράδια, που όλα αιωρούνται στον αέρα και περιστρέφονται γύρω από τον εαυτό τους και μαγνητίζονται από τον ήρωα όταν τα πλησιάσει αρκετά.

Η περιστροφή γύρω από τον εαυτό τους και η έλξη από το μαγνήτη γίνεται με τον ίδιο τρόπο που υλοποιήθηκε από τη SnowLand και τη WaterLand, χρησιμοποιώντας την **transform.Rotate** για την περιστροφή τους ανά το χρόνο, και την **Vector3.MoveTowards** για τη μετακίνησή τους προς τον ήρωα αλλάζοντας τη θέση τους με την transform.position, και θέτοντας ως στόχο τον ήρωα.

Όταν η OnTriggerEnter() ενημερώσει για την επαφή τους με τον ήρωα, θέτουμε αληθή την υπεύθυνη μεταβλητή αναλόγως με το αντικείμενο, με την οποία πηδάει πιο ψηλά προσθέτοντας μια δύναμη στην ταχύτητα άλματός του και μετακινώντας τον πιο ψηλά ακολουθώντας το στόχο πάνω από το κεφάλι του, και καταστρέφεται το ίδιο το αντικείμενο. Στο **CoinJump.js** που ελέγχεται η συμπεριφορά των νομισμάτων, θέτουμε αληθή την jumpHigher, στο **PowerupJump.js** που ελέχεται το μαγικό σμαράγδι θέτουμε αληθή την dJumpHigher για να αποκτήσει πολύ μεγάλη ταχύτητα, και στο **FishCtrlJump.js** που ελέγχει τα ψαράκια μπόνους θέτουμε αληθή επίσης jumpHigher και αυξάνουμε τον αριθμό των ψαριών κατά ένα.

Επίσης, τα ψαράκια εκτός από την περιστροφή γύρω από τον εαυτό τους, ανεβοκατεβαίνουν στον αέρα μέσα σε συκεκριμένα όρια, ώστε να είναι ακόμα πιο δύσκολη η συλλογή τους. Η μετακίνησή τους στον κάθετο άξονα γίνεται με τον τρόπο που υλοποιήσαμε στους προηγούμενους κόσμους, ορίζοντας μια ταχύτητα στη γωνία περιστροφής, επαναλαμβάνοντας τις τιμές των γωνιών, και ταλατεύοντας τη θέση τους στον άξονα y **transform.localPosition.y** μεταξύ ορίων αναλόγως τις μοίρες περιστροφής ξεκινώντας από την αρχική τους θέση.

5.6.5 Υλοποίηση κουμπιών

Το μόνο κουμπί αυτού του κόσμου κατά τη διάρκεια του παιχνιδιού, είναι το κουμπί παύσης, το οποίο υλοποιήθηκε με τον ίδιο τρόπο με τους προηγούμενους κόσμους. Είναι ένα GUITexture που ελέγχεται από τον κώδικα **PauseTouchJump.js**, όπου απενεργοποιούμε την εικόνα του κουμπιού αν ο χρήστης κέρδισε ή έχασε, ή επέλεξε παύση είτε με τα δύο δάχτυλα στην οθόνη είτε πατώντας το κουμπί παύσης, ή αν η συσκευή βρίσκεται σε λάθος θέση. Αν καμία από αυτές τις περιπτώσεις δεν ισχύει, τότε ενεργοποιούμε την εικόνα του κουμπιού και αν υπάρχει κάποιο άγγιγμα **Input.touchCount**, ελέγχουμε για το καθένα **Input.touches** ξεχωριστά τη θέση στην οποία πραγματοποιήθηκε **touch.position**. Αν το δάχτυλο ήταν στη θέση που βρίσκεται η εικόνα του κουμπιού **GUIElement.HitTest** τότε σταματάμε το χρόνο του παιχνιδιού και θέτουμε την pauseNow αληθή για να εμφανιστεί το μενού παύσης.

5.6.6 Υλοποίηση GUI στοιχείων

Υλοποίηση καρτέλας παύσης

Η καρτέλα παύσης είναι ένα GUITexture που ελέγχεται από τον κώδικα **PauseGuiJumpMobile.js** και εμφανίζεται είτε επιλέγοντας το κουμπί παύσης, είτε τοποθετώντας ταυτόχρονα στην οθόνη, εφόσον η συσκευή βρίσκεται στη σωστή θέση. Υλοποιείται με τον ίδιο τρόπο με τους προηγούμενους κόσμους, αποτελείται από τέσσερα κουμπιά που δημιουργήθηκαν στη συνάρτηση OnGUI() με την GUI.Button(). Δύο κουμπιά αφής για την επιστροφή στο παιχνίδι από το σημείο που το αφήσαμε, ένα για την επανάληψη

της πίστας, και ένα για την επιστροφή στο μενού των πιστών αυτού του κόσμου. Με την επιλογή του κάθε κουμπιού, αναπαράγεται ένας ήχος κουμπιού, ξεκινάει πάλι ο χρόνος και εκτελείται η αντίστοιχη λειτουργία. Στο κάτω μέρος εμφανίζεται με την GUI.Label ένα μήνυμα που παροτρύνει τον χρήστη να συνεχίσει το παιχνίδι.

Υλοποίηση ετικετών

Στην οθόνη κατά τη διάρκεια του παιχνιδιού, εμφανίζονται πληροφορίες σχετικά με τις επιδόσεις του ήρωα. Αν ο χρήστης δεν έχει χάσει ή κερδίσει, και αν δεν έχει επιλέξει παύση και αν κρατάει τη συσκευή στη σωστή θέση, αναγράφεται ο αριθμός των νομισμάτων που συλλέγει δίπλα σε μια ετικέτα ενός νομίσματος, και για κάθε ψαράκι μπόνους που συλλέγει δίνει στην αντίστοιχη αχνή ετικέτα του έντονο χρώμα. Αυτά ελέγχονται από ένα GUIText μέσω του κώδικα **ScoreJumpMobile.js**, το οποίο έχουμε συνδέσει στην κύρια κάμερα ώστε να την ακολουθεί παντού και να αναπαράγει τη μουσική υπόκρουση του παιχνιδιού. Αρχικά ελέγχεται στη συνάρτηση Start() αν ο χρήστης είχε επιλέξει στο μενού να είναι ενεργοποιημένος ο ήχος ή όχι, για να ξέρουμε αν θα τον αναπαράγουμε. Στη συνάρτηση Update() χαμηλώνουμε την ένταση της μουσικής με την **AudioListener.volume** αν ο χρήστης χάσει ή κερδίσει, ή επιλέξει παύση ή κρατήσει λάθος τη συσκευή για να δείξουμε πως το παιχνίδι δεν συνεχίζεται, και την επαναφέρουμε στην κανονική της ένταση όταν δεν ισχύουν αυτές οι περιπτώσεις.

Υλοποίηση βαθμολογίας

Όταν ο ήρωας κερδίσει εμφανίζεται η καρτέλα στη συνάρτηση OnGUI(), με τη βαθμολογία του που αποτελείται από τον αριθμό των νομισμάτων και των ψαριών, το οποίο υλοποιείται από ένα GUITexture και τον κώδικα **WinGuiJumpMobile.js**. Για κάθε πίστα ελέγχουμε την μεγαλύτερη αποθηκευμένη βαθμολογία από την **PlayerPrefs** που είχε αποκτηθεί κάποια προηγούμενη φορά και τη συκρίνουμε με την τωρινή. Κάθε φορά που πραγματοποιείται νέο ρεκόρ βαθμολογίας, αναγράφεται με γαλάζια γράμματα “High Score” και το αποτέλεσμα, ενώ αν δεν είναι νέο ρεκόρ, αναγράφεται απλά με κίτρινα γράμαμτα “Score” και το αποτέλεσμα.

Από κάτω αναγράφονται αναλυτικά ο αριθμός των νομισμάτων και εμφανίζονται με έντονο χρώμα τα εικονίδια των ψαριών που συνέλεξε. Από κάτω υπάρχουν τρία κουμπία αφής που δημιουργήθηκαν με την GUI.Button, ένα για την επιστροφή στο μενού των πιστών, ένα για την επανάληψη της πίστας, και ένα για την εισαγωγή στην επόμενη πίστα. Στο κάτω μέρος της οθόνης εμφανίζεται ένα μήνυμα στο χρήστη μέσω της GUI.Label.

```
function OnGUI ()
{
        guiTexture.enabled = false;              //we need to make the 2d texture invisible
```

```
        if( EndCtrlJump.WinGameJ && MegaJumpMobile.hasLost == false)     //if we won tha game
        {
                guiTexture.enabled = true;                  //we need to make the 2d texture visible
                if(jumpLand == 1)
                {
                        if (PlayerPrefs.GetInt("jHighScore1")==MegaJumpMobile.ScoreCounter)
                        {
                                // Show the HIGH SCORE
                                GUI.Label (Rect ( widthButHighScore , bonusHeight, totalWidth,
                                                totalHeight ), "High Score:", myHighScore);
                                GUI.Label (Rect ( highscoreWidth, highscoreHeight, totalWidth,
                                     totalHeight )," " +MegaJumpMobile.ScoreCounter, myHighScore);
                        }
                        if (PlayerPrefs.GetInt("jHighScore1")>MegaJumpMobile.ScoreCounter)
                        {
                                // Show the SCORE
                                GUI.Label (Rect ( widthBut , bonusHeight, totalWidth, totalHeight ),
                                        "Score:", myBonusStyle);
                                GUI.Label (Rect ( highscoreWidth, highscoreHeight, totalWidth,
                                     totalHeight ), " "+MegaJumpMobile.ScoreCounter, myBonusStyle);
                        }
                }
        }
}
```

Υλοποίηση καρτέλα τέλος παιχνιδιού

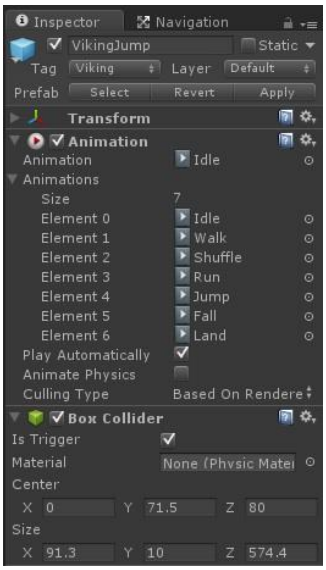
Όταν χάσει ο ήρωας, εμφανίζεται μια καρτέλα τέλους από ένα GUITexture που ελέχεται από το **GameOverGuiJumpMobile.js**. Περιλαμβάνει τρία κουμπιά στο κέντρο της οθόνης μέσω της GUI.Button, ένα για την επιστροφή στο μενού των πιστών, ένα για την επανάληψη της πίστας, και ένα για την εισαγωγή στην επόμενη πίστα μόνο εφόσον η τωρινή είχε τερματιστεί επιτυχώς κάποια προηγούμεη φορά. Στο κάτω μέρος εμφανίζεται ένα μήνυμα προς το χρήστη μέσω της GUI.Label.

5.6.7 Υλοποίηση σημείου τερματισμού

Όταν ο ήρωας φτάσει στα σύννεφα, εμφανίζεται ένα μήνυμα νίκης στο κέντρο της οθόνης “You Win!” με γαλάζια γράμματα με την **GUI.Label**, το οποίο αναβοσβήνει για λίγα δευτερόλεπτα και μετά παραμένει σταθερό μέχρι να εμφανιστεί η καρτέλα της βαθμολογίας, όπου σταματάμε το χρόνο του παιχνιδιού. Τα σύννεφα αυτά αποτελούνται από ένα Plane με την εικόνα των σύννεφων, ένα Box Collider για να εντοπιστεί η επαφή με τον ήρωα, και τον κώδικα **EndCtrlJump.js**.

5.6.8 Υλοποίηση εχθρού

Οι βίκινγκ βρίσκονται κρυμμένοι στα σύννεφα και στα δέντρα και στέκονται περιμένοντας τον πιγκουίνο να πέσει πάνω τους για να τον σκοτώσουν. Αποτελούνται από έναν Box Collider για να εντοπιστεί η επαφή αυτή και το Animation component το οποίο αναπαράγει συνεχώς το animation clip Idle.



5.7 Υλοποίηση της RunLand

5.7.1 Υλοποίηση της περιστροφής

Οι πίστες αυτού του κόσμου είναι ειδικά σχεδιασμένες για να λειτουργούν σε όρθια θέση του κινητού με το κεντρικό κουμπί προς τα κάτω. Όσο η συσκευή **Input.deviceOrientation** είναι σε θέση **DeviceOrientation.Portrait** ο χρόνος συνεχίζεται κανονικά εφόσον ο χρήστης δεν έχει επιλέξει παύση, είτε τοποθετώντας δύο δάχτυλα στην οθόνη είτε επιλέγοντας το κουμπί παύσης. Για οποιαδήποτε άλλη όρθια θέση της συσκευής, το παιχνίδι δεν συνεχίζεται και σταματάμε το χρόνο μέχρι να περιστραφεί στη σωστή θέση. Ανεξαρτήτως περιστροφής, η οθόνη **Screen.orientation** είναι στραμμένη σε θέση Portrait για να καταλαβαίνει ο χρήστης σε ποια θέση πρέπει να την κρατήσει. Αν η συσκευή είναι σε θέση παράλληλη προς το έδαφος με την οθόνη προς τα πάνω ή προς τα κάτω, το παιχνίδι σταματάει και εμφανίζεται μια εικόνα σε όλη την έκταση της οθόνης με ένα μήνυμα που ενημερώνει το χρήστη να σηκώσει το κινητό για να παίξει.

Η μεταβλητή **canShow** χρησιμοποιείται για να εμφανιστεί ένα μενού επιλογής χειρισμού της κίνησης του ήρωα στον άξονα x, στο κέντρο της οθόνης πριν την εκκίνηση του παιχνιδιού. Η μεταβλητή αυτή είναι ψευδής όταν η συσκευή είναι παράλληλη προς το έδαφος ώστε να μην εμφανίζεται το μενού των επιλογών αυτών, εκτός από την εικόνα με το μήνυμα περιστροφής, και είναι αληθής σε όλες τις άλλες θέσεις της συσκευής όπου εμφανίζεται και αντίστοιχα το μενού.

Οι έλεγχοι αυτού πραγματοποιούνται από ένα GUITexture και τον κώδικα **TiltControlRun.js** που είναι παρόμοιος με αυτόν που χρησιμοποιήθηκε στη JumpLand για την περιστροφή.

5.7.2 Υλοποίηση του μενού διαχείρισης της κίνησης

Στο μενού αυτό επιλέγει ο χρήστης αν θα κινεί τον ήρωα με την κλίση της συσκευής ή με τα κουμπιά. Δημιουργούμε δύο στατικές μεταβλητές για να μπορούμε να έχουμε πρόσβαση σε αυτές από την κλάση ελέγχου κίνησης του πιγκουίνου, οι οποίες σχετίζονται με την επιλογή του τρόπου κίνησης του ήρωα, μέσω επιταχυνσιόμετρου με την **tiltChoice**, ή μέσω κουμπιών

αφής **touchChoice**. Το παιχνίδι θα ξεκινήσει όταν ο χρήστης επιλέξει κάποιο από τα δύο κουμπιά του μενού, θέτοντας την μεταβλητή **canPlay** αληθή.

Πριν ξεκινήσει το παιχνίδι, και εφόσον η συσκευή δεν είναι παράλληλη προς το έδαφος (οπότε δεν υπάρχει καρτέλα μηνύματος και μπορεί να εμφανιστεί το μενού με την canShow αληθή), και δεν έχει επιλεχθεί κάποιο κουμπί από αυτό το μενού (με την canPlay ψευδή), εμφανίζεται στο κέντρο της οθόνης ένα μενού για τη διαχείριση της κίνησης του ήρωα στον οριζόντιο άξονα από ένα GUITexture που ελέγχεται από το **Controls.js**. Ο ήρωας κατά τη διάρκεια του παιχνιδιού, ανεξάρτητα από την στροφή 90 μοιρών στο χώρο, μπορεί να κινηθεί δεξιά και αριστερά πλαγίως είτε πιέζοντας τα αντίστοιχα κουμπιά αφής που βρίσκονται δεξιά και αριστερά στην οθόνη, είτε δίνοντας κλίση στη συσκευή δεξιά και αριστερά αντίστοιχα.

Ο χρήστης μπορεί να διαλέξει τον τρόπο που θα χειριστεί αυτήν την κίνηση, επιλέγοντας το αντίστοιχο κουμπί πάνω στην καρτέλα που εμφανίζεται. Μέχρι να επιλέξει ο χρήστης ένα από τα δύο κουμπιά, το παιχνίδι δεν ξεκινάει και ο χρόνος είναι σταματημένος. Για να τονίσουμε μόνο την καρτέλα του μενού και να καταλάβει ο χρήστης ότι το παιχνίδι βρίσκεται σε παύση, η πίστα που φαίνεται στο πίσω μέρος της καρτέλας είναι με πολύ χαμηλό φωτισμό και φαίνεται σκούρο το περιβάλλον. Μόλις επιλέξει κάποιο κουμπί, η καρτέλα εξαφανίζεται, το φως γίνεται έντονο και ο χρόνος του παιχνιδιού ξεκινάει. Την ένταση του φωτισμού του Directional Light της πίστας, την ελέγχουμε μέσα από την **Light.intensity**

```
#pragma strict
static var canPlay : boolean = false;
static var tiltChoice : boolean = false;
static var touchChoice : boolean = false;
function Start ()
{
	canPlay = false;
if(canPlay == false && TiltCtrlRun.canShow == true && PauseTouchRun.pauseNow == false)
{
	guiTexture.enabled = true;
}
	gameObject.Find("Directional light").transform.GetComponent(Light).intensity = 0.1;
	tiltChoice = false;
	touchChoice = false;
	Time.timeScale = 0;
}
```

Πάνω από την καρτέλα υπάρχει το κουμπί παύσης το οποίο είναι ενεργό. Στην Update() συνάρτηση ελέγχουμε αν επιλέχθηκε το κουμπί παύσης ή αν η συσκευή τοποθετήθηκε παράλληλα προς το έδαφος και απενεργοποιούμε την καρτέλα του μενού. Αν δεν ισχύει καμία από αυτές τις περιπτώσεις και εφόσον δεν έχει πατηθεί κανένα κουμπί ακόμα από το μενού, η καρτέλα θα παραμένει στο κέντρο της οθόνης.

```
function Update ()
{
	if(PauseTouchRun.pauseNow == true || TiltCtrlRun.canShow == false)
	{
		guiTexture.enabled = false;
	}
	else if(canPlay == false && PauseTouchRun.pauseNow == false &&
                                                        TiltCtrlRun.canShow == true)
	{
		guiTexture.enabled = true;
	}
}
```

Στη συνάρτηση OnGUI() δημιουργούμε τα δύο κουμπιά αφής, για τα οποία θέτουμε στις Texture2D μεταβλητές **tiltControl** και **touchControl** τις εικόνες που επιθυμούμε με την GUI.Button, και τα εμφανίζουμε μόνο όσο δεν έχει επιλεχθεί κάποιο κουμπί, δεν έχει επιλεχθεί η παύση και η συσκευή δεν είναι παράλληλη προς το έδαφος. Αν επιλεχθεί κάποιο κουμπί, ενεργοποιείται ο χρόνος του παιχνιδιού, η ένταση του φωτισμού επανέρχεται στη φυσιολογική της τιμή, και απενεργοποιείται η καρτέλα του μενού και τα κουμπιά, θέτοντας την canPlay θετική. Για να ξέρουμε με ποιο τρόπο θα πραγματοποιηθεί η κίνηση αναλόγως το κουμπί που επιλέγεται, θέτουμε την **tiltChoice** θετική αν η επιλεχθεί ο χειρισμός μέσω της κλίσης του κινητού **Tilt**, αλλιώς θέτουμε θετική την **touchChoice** αν επιλεχθεί ο χειρισμός μέσω κουμπιών αφής **Touch**. Το μέγεθος της καρτέλας και των κουμπιών προσαρμόζεται στις διαστάσεις της οθόνης της κάθε συσκευής ώστε να πιάνει την ίδια έκταση.

```
//BUTTONS
var tiltControl : Texture2D;
var touchControl : Texture2D;

function OnGUI()
{
	guiTexture.pixelInset.x = -Screen.width*0.45;
	guiTexture.pixelInset.y = -Screen.height/5;
	guiTexture.pixelInset.width = Screen.width * 0.92;
	guiTexture.pixelInset.height = Screen.height/2;
if(canPlay == false && TiltCtrlRun.canShow == true && PauseTouchRun.pauseNow == false)
{
	var width = Screen.width * 0.19;
	var height = Screen.height *0.18;
	var widthStart = Screen.width/2 - width*1.35;
	var heightStart = Screen.height/2;

	var widthStart2 = Screen.width/2 + width/2;
```

```
        if (GUI.Button(Rect(widthStart,heightStart,width,height),tiltControl,""))
        {
                canPlay = true;
                tiltChoice = true;
                touchChoice = false;
                guiTexture.enabled = false;
                gameObject.Find("Directional light").transform.GetComponent(Light).intensity = 0.5;
                if(PauseTouchRun.pauseNow == false && RunMoveMobile.PauseFingers == false)
                {
                        Time.timeScale = 1;
                }
        }
        if (GUI.Button(Rect(widthStart2,heightStart,width,height),touchControl,""))
        {
                canPlay = true;
                tiltChoice = false;
                touchChoice = true;
                guiTexture.enabled = false;
                gameObject.Find("Directional
light").transform.GetComponent(Light).intensity = 0.5;
                if(PauseTouchRun.pauseNow == false &&
RunMoveMobile.PauseFingers == false)
                {
                        Time.timeScale = 1;
                }
        }
}
}
```

5.7.3 Υλοποίηση του πιγκουίνου

Ο ήρωας αποτελείται από το συστατικό Transform για τον έλεγχο της θέσης του, τον Animator το οποίο περιλαμβάνει το **penguinAnimCtrlRun** animator controller για τον έλεγχο των animations, τον Character Controller για τον έλεγχο της κίνησής του, το Rigidbody για την προσθήκη φυσικής στη συμπεριφορά του, το Audio Source για την αναπαραγωγή του ήχου, το script **RunMoveMobile.js** για τον έλεγχο της συμπεριφοράς του, και το script **SwipeCtrlRun.js** για τον έλεγχο της κατεύθυνσης του δαχτύλου του χρήστη στην οθόνη για την ενεργοποίηση αντίστοιχων συμπεριφορών.

Στη συνάρτηση Start() αρχικοποιούμε όλες τις μεταβλητές και καταστάσεις, και αποκτάμε πρόσβαση στα συστατικά του χαρακτήρα για

να μπορέσουμε να τα ελέγξουμε. Επίσης ελέγχουμε αν θα πραγματοποιηθεί αναπαραγωγή του ήχου αναλόγως την επιλογή του χρήστη στα αρχικά μενού.

Ορίζουμε μια μεταβλητή **showButtons** η οποία είναι θετική αν ο χρήστης φτάσει στο σημείο που ξεκινάει η δράση και παίρνει ο χρήστης τον έλεγχο της κίνησης, για να γνωρίζουμε πότε να ενεργοποιήσουμε τα κουμπιά κίνησης εφόσον έχουν επιλεχθεί. Αν ο χρήστης επιλέξει να χειριστεί την κίνηση στον οριζόντιο άξονα μέσω των κουμπιών αφής και τεθούν αληθείς οι canPlay και touchChoice, και όσο χρήστης δεν έχει χάσει ή κερδίσει, ή δεν έχει επιλέξει παύση του παιχνιδιού είτε με το αντίστοιχο κουμπί είτε με δύο δάχτυλα στην οθόνη, και η συσκευή βρίσκεται σε σωστή θέση, και ο ήρωας έχει φτάσει στο σημείο στο οποίο ξεκινάει το παιχνίδι με την showButtons αληθή, δημιουργούμε τα δύο κουμπιά αφής στη συνάρτηση OnGUI(), με τα οποία θα μετακινείται δεξιά και αριστερά στον οριζόντιο άξονα. Όσο ο χρήστης πιέζει το κάθε κουμπί, τόσο ο πιγκουίνος προχωράει προς αυτή τη κατεύθυνση. Χρειάζεται δηλαδή να ξέρουμε πόση διάρκεια επιλέγει το κάθε κουμπί. Έτσι για την υλοποίηση των κουμπιών χρησιμοποιούμε την **GUI.RepeatButton** διατηρεί ενεργό το κουμπί όσο ο χρήστης το πιέζει.

Επίσης ορίζουμε μια μεταβλητή **movementX** με την οποία καθορίζουμε τις εισόδους που έχουμε από τα κουμπιά για να μπορέσουμε να κινήσουμε τον ήρωα. Όσο είναι επιλεγμένο το αριστερό κουμπί θέτουμε τη μεταβλητή ίση με -1 για να κινηθεί ο ήρωας προς τα αριστερά, όσο είναι επιλεγμένο το δεξί κουμπί τη θέτουμε ίση με 1 για να κινηθεί δεξιά, και όσο δεν είναι επιλεγμένο κανένα κουμπί είναι ίση με μηδέν ώστε να μην κινείται.

```
function OnGUI ()
{
if(Controls.canPlay && Controls.touchChoice)
{
if( HealthRun.GameOver == false && HealthRun.WinGame == false &&  showButtons == true &&
PauseTouchRun.pauseNow == false && PauseFingers == false && TiltCtrlRun.canShow == true)
{
        //MOVEMENT BUTTONS
        var widthBut = Screen.width * 0.2;
        var heightBut = Screen.height * 0.2;
        var width1 = Screen.width*0.04;
        var height = Screen.height - Screen.height*0.34;
        var width2 = Screen.width - widthBut;
        //GO LEFT
        if(GUI.RepeatButton(Rect(width1, height, widthBut, heightBut), leftButton,""))
        //Make a button that is active as long as the user holds it down.
        {
                movementX = -1;
        }
        //GO RIGHT
```

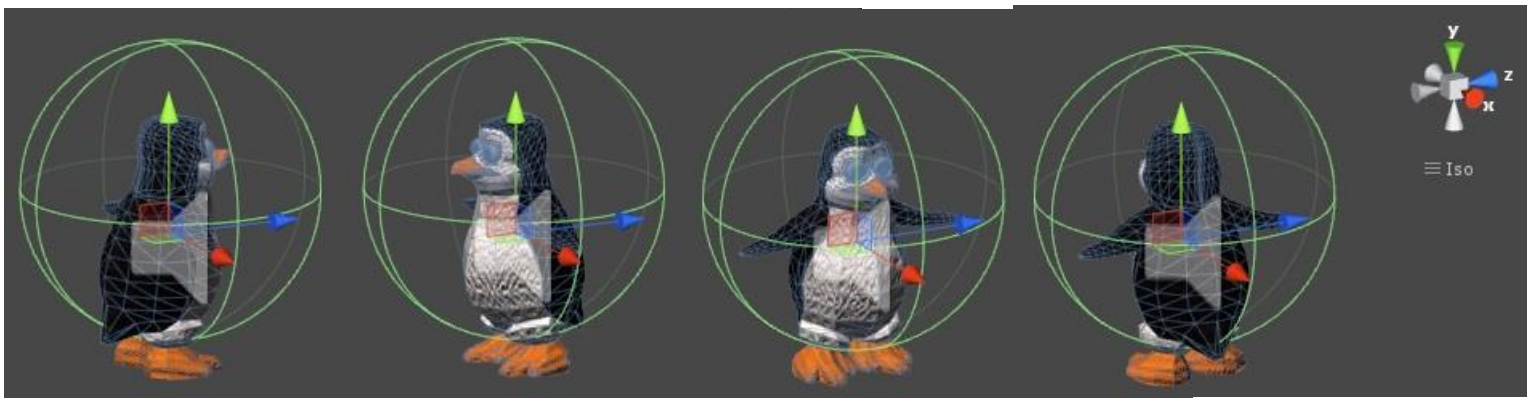
```
        else if(GUI.RepeatButton(Rect(width2, height, widthBut, heightBut), rightButton,""))
        {
                movementX = 1;
        }
        else
        {
                movementX = 0;
        }
}
}
}
```

Στη συνάρτηση Update() ελέγχουμε αν έχει επιλεχθεί κάποιο κουμπί για τον χειρισμό της κίνησης για να μπορέσει να ξεκινήσει το παιχνίδι, δηλαδή αν η canPlay είναι αληθής. Αν ο ήρωας έχει φτάσει στο σημείο που αλλάζει η κάμερα και ενεργοποιείται ο έλεγχος της κίνησης (showButtons είναι αληθής), υπολογίζεται η απόσταση μεταξύ του ήρωα και του στόχου-θυσαυρού μέσω της **Vector3.Distance** εισάγοντας τη θέση τους ως εισόδους, ώστε να εμφανιστεί στο πάνω μέρος της οθόνης η απόσταση του ήρωα από το θησαυρό για να ξέρει αν πλησιάζει ή αν έχει χάσει το δρόμο του.

Ορίζουμε μια μεταβλητή **canMove** την οποία θέτουμε αληθή όταν ο ήρωας μπορεί να κινηθεί, δηλαδή αν δεν έχει χτυπηθεί ή πεθάνει. Αν η μεταβλητή αυτή είναι αληθής, καθορίζουμε την κίνηση του ήρωα ελέγχοντας την επιλογή του χρήστη ως προς το χειρισμό της οριζόντιας κίνησης. Επίσης ορίζουμε και τέσσερις μεταβλητές για να δηλώνουμε προς τα ποια κατεύθυνση κινεί ο χρήστης τον ήρωα για να καθορίσουμε τη συμπεριφορά του. Αν η **amStraight** είναι αληθής ο πιγκουίνος κατευθύνεται ευθεία προς τα μπροστά, αν η **amRight** είναι αληθής ο ήρωας στρίβει 90° δεξιά, αν είναι αληθής η **amLeft** στρίβει 90° αριστερά, και αν είναι αληθής η **amBackward** έχει κάνει μεταβολή 180° και κατευθύνεται προς τα πίσω. Στη μεταβλητή **horizontal** θέτουμε τις τιμές κίνησης στον οριζόντιο άξονα, και στην **vertical** τιμές για την κίνηση στον κάθετο άξονα. Τη μεταβλητή **Froze** τη χρησιμοποιούμε όταν θέλουμε να παγώσουμε την κίνηση του ήρωα θέτοντας μηδέν τις εισόδους των μεταβλητών horizontal και vertical όσο η Froze είναι αληθής.

Αν η touchChoice είναι αληθής, δηλαδή έχει επιλέξει να μετακινεί τον ήρωα με τα κουμπιά που βρίσκονται δεξιά και αριστερά, και αν δεν έχει χάσει όλες του τις ζωές (ώστε να μην μπορεί να κινείται άλλο αφού πεθάνει), ελέγχουμε προς τα ποια κατεύθυνση πρέπει να πάει για να τον στρέψουμε προς τα εκεί.

Αν η amStraight είναι αληθής πρέπει να πάει προς τα μπροστά. Ο πιγκουίνος με μηδέν μοίρες περιστροφής κοιτάει την κάμερα, όμως πρέπει να έχει στραμμένη την πλάτη του προς την κάμερα και να προχωράει. Για να προχωρήσει προς τα μπροστά προς τη μεριά που κοιτάει η κάμερα, τον περιστρέφουμε 180° στον άξονα y μέσω της **transform.eulerAngles.y**. Όσο είναι στραμμένος προς αυτήν την κατεύθυνση, ο άξονα z είναι θετικός προς το πίσω μέρος της πλάτης του, ο άξονας x είναι θετικός προς το αριστερό του χέρι, ενώ ο άξονας y παραμένει θετικός προς τα πάνω. Ενώ όσο είχε μηδέν μοίρες περιστροφής ο άξονα z ήταν θετικός προς την κατεύθυνση που κοιτούσε, και ο άξονας x προς το δεξί του χέρι. Στις εικόνες που ακολουθούν φαίνονται οι άξονες ως προς τον ήρωα με μηδέν περιστροφή στον άξονα y, με περιστροφή 180°, με 90° και με 270° αντίστοιχα.



Η κίνησή του στον οριζόντιο άξονα καθορίζεται από την movementX, η οποία είναι ορισμένη να κινεί τον χαρακτήρα αριστερά και δεξιά με μηδέν περιστροφή. Επειδή ο πιγκουίνος είναι στραμμένος 180°, οι τιμές που καθορίζουν την κατεύθυνση πρέπει να αντιστραφούν. Στη μεταβλητή horizontal θέτουμε την αρνητική τιμή της movementX, ώστε να είναι αντίθετη για κάθε περίπτωση. Στη μεταβλητή vertical θέτουμε την τιμή -1 καθώς θέλουμε ο ήρωας να έχει μια σταθερή ταχύτητα όλη τη διάρκεια του παιχνιδιού ως προς την κάθετη κίνηση, και να μην είναι ποτέ σταθερός. Η τιμή αυτή είναι αρνητική επειδή ο ήρωας είναι στραμμένος προς την αντίθετη κατεύθυνση, διαφορετικά θα ήταν ίση με 1.

Αν η amRight είναι αληθής πρέπει να στρίψει άμεσα 90° δεξιά όπως κοιτάει ο πιγκουίνος στο παιχνίδι, που σημαίνει στον αρνητικό άξονα των x που βρίσκεται στο δεξί του χέρι. Προσθέτοντας 90° για να στρίψει ενώ βρίσκεται στις 180°, καταλήγει να έχει 270° περιστροφή στον άξονα των y. Η οριζόντια με την κάθετη κίνηση θα αντιστραφούν, οπότε η horizontal θα γίνει ίση με -1 για να συνεχίσει να τρέχει ο πιγκουίνος προς τα μπροστά στην κατεύθυνση που κοιτάει αφού έστριψε, και η vertical θα γίνει η πλαϊνή κίνηση εισάγοντας τις τιμές της movementX χωρίς αντιστροφή καθώς τώρα ο άξονας z είναι θετικός από τη δεξιά μεριά του πιγκουίνου.

Αν η amLeft είναι αληθής πρέπει να στρίψει άμεσα 90° αριστερά όπως κοιτάει ο πιγκουίνος στο παιχνίδι, δηλαδή να στρίψει προς τον θετικό άξονα των x που βρίσκεται στο αριστερό του χέρι. Αφαιρούμε 90° για να στρίψει ενώ βρίσκεται στις 180°, καταλήγοντας να έχει 90° περιστροφή στον άξονα των y. Η οριζόντια και η κάθετη κίνηση έχουν ίδιες αλλά αντίθετες τιμές από αυτές που έχουν όταν ο ήρωας στρίβει προς τα δεξιά, οπότε η horizontal θα γίνει ίση με 1 για να συνεχίσει να τρέχει ο πιγκουίνος προς τα μπροστά στην κατεύθυνση που κοιτάει αφού έστριψε, και η vertical θα γίνει η πλαϊνή κίνηση εισάγοντας τις αντίθετες τιμές της movementX καθώς τώρα ο άξονας z είναι αρνητικός από τη δεξιά μεριά του πιγκουίνου.

Αν η amBackward είναι αληθής, πρέπει να στρίψει άμεσα 180°, δηλαδή να κοιτάει προς τον θετικό άξονα των z, αποκτώντας μηδέν μοίρες περιστροφής στον άξονα των y. Η οριζόντια με την κάθετη κίνηση παίρνουν τις κανονικές του τιμές, η horizontal θα πάρει τις εισόδους της movementX για να αποδοθεί η οριζόντια κίνηση, και η vertical θα γίνει ίση με 1 για να κινείται συνεχώς κάθετα ο ήρωας.

Αν όμως ο χρήστης είχε επιλέξει να χειριστεί την πλάγια κίνηση μέσω του επιταχυνσιομέτρου, έχοντας την tiltChoice αληθή, και αν δεν έχει χάσει όλες τις ζωές του, εισάγουμε άλλες τιμές στις μεταβλητές αυτές. Στη μεταβλητή **moveThreshold** θέτουμε το όριο από το οποίο θα αρχίσουμε να κινούμε τον ήρωα αναλόγως τις εισόδους του επιταχυνσιομέτρου, στην **movex** εισάγουμε τις τιμές που καθορίζουν την οριζόντια κίνηση, και στην **iPx** εισάγουμε τις τιμές που επιστρέφει το επιταχυνσιόμετρο. Αρχικά θέτουμε την movex ίση με μηδέν και την iPx εισάγουμε τις τιμές που επιστρέφει το επιταχυνσιόμετρο από την κίνηση της συσκευής για τον άξονα x μέσω του Input.acceleration.x. Αν η απόλυτη τιμή της τιμής του επιταχυνσιομέτρου είναι μεγαλύτερη από το όριο moveThreshold που έχουμε θέσει, τότε ο ήρωας πρέπει να κινηθεί και θέτουμε στην movex την τιμή 1 για να κινηθεί δεξιά αν η iPx είναι θετική ή μηδέν, και ίση με -1 για να κινηθεί αριστερά αν είναι αρνητική.

Στη συνέχεια ελέγχουμε τις μεταβλητές που δηλώνουν την κατεύθυνση του ήρωα για να θέσουμε τις μεταβλητές τις οριζόντιας και της κάθετης κίνησης με τον ίδιο τρόπο με προηγουμένως, αλλά με την διαφορά ότι όπου εισάγαμε την movementX με τις εισόδους των κουμπιών, τώρα εισάγουμε την movex με τις εισόδους του επιταχυσιομέτρου.

Αν η amStraight είναι αληθής προχωράει ευθεία και θέτουμε τις μοίρες στον άξονα y 180°, τη μεταβλητή horizontal που καθορίζει την οριζόντια κίνηση ίση με τις αντίθετες τιμές της movex, και την vertical που καθορίζει την κάθετη κίνηση ίση με -1, ώστε να αντιδράσει σωστά ο ήρωας σύμφωνα με την κατεύθυνση που βρίσκεται. Αν η amRight είναι αληθής στρίβει δεξιά και θέτουμε τις μοίρες 270°, τη μεταβλητή horizontal που τώρα καθορίζει την κάθετη κίνηση ίση με -1, και την vertical που καθορίζει την οριζόντια κίνηση ίση με movex. Αν η amLeft είναι αληθής στρίβει αριστερά και θέτουμε τις μοίρες 90°, τη μεταβλητή horizontal

ίση με 1, και την vertical ίση με τις αντίθετες τιμές της movex. Αν η amBackward είναι αληθής στρίβει προς τα πίσω και θέτουμε τις μοίρες 0°, τη μεταβλητή horizontal ίση με movex, και την vertical ίση με 1.

Αν κατά τη διάρκεια οποιασδήποτε κίνησης η Froze γίνει θετική, οι εισόδοι των μεταβλητών horizontal και vertical γίνονται μηδέν για κάθε περίπτωση ξεχωριστά, και ο πιγκουίνος μένει ακίνητος στο σημείο που βρισκόταν μόλις αυτή ενεργοποιήθηκε.

Για να στρίψει ο ήρωας άμεσα 90°, πρέπει ο χρήστης να σύρει το δάχτυλο του πάνω στην οθόνη προς την αντίστοιχη κατεύθυνση, η οποία αναγνωρίζεται από το script **SwipeCtrlRun.js**. Αν εντοπίσει ότι το δάχτυλο του χρήστη σύρθηκε προς τα αριστερά θέτει την **canRotateRight** αληθή με την οποία καταλαβαίνουμε ότι πρέπει να στρίψουμε αριστερά και αν δεν έχει χάσει τις ζωές του, θέτουμε την μεταβλητή **turnLeft** αληθή που ενεργοποιεί αυτήν την ενέργεια. Αντίστοιχα, αν εντοπίσει κίνηση δαχτύλου προς τα δεξιά θέτει αληθή την **canRotateLeft** την οποία διαβάζουμε για να θέσουμε την **turnRight** αληθή ώστε να ενεργοποιηθεί η αντίστοιχη στροφή αν δεν έχει χάσει όλες τις ζωές του ώστε να μην μπορεί να στρίβει αφού πεθάνει.

Αν ο ήρωας χάσει και τις τρεις ζωές του, η Froze γίνεται αληθής ώστε να σταματήσει να κινείται, ενεργοποιούμε το animation με το οποίο πεθαίνει θέτοντας την παράμετρο Die αληθή και αναπαράγοντας τον ήχο ήττας παιχνιδιού.

Αν ο χρήστη τοποθετήσει δύο δάχτυλα ταυτόχρονα στην οθόνη σταματάει ο χρόνος του παιχνιδιού και εμφανίζεται η καρτέλα με το μενού παύσης. Τη μεταβλητή **fingerCount** τη χρησιμοποιούμε για να μετρήσουμε τον αριθμό των δαχτύλων που βρίσκονται κάθε καρέ στην οθόνη, και αρχικά τη θέτουμε ίση με μηδέν. Για κάθε άγγιγμα που εντοπίζεται με τη **Input.touches** το οποίο το αποθηκεύουμε σε μια μεταβλητή **touch**, ελέγχουμε τη φάση στην οποία βρίσκεται. Αν το άγγιγμα δεν τελείωσε TouchPhase.Ended ή δεν ακυρώθηκε TouchPhase.Canceled αυξάνουμε τον αριθμό της fingerCount κατά ένα. Αν γίνει ίση με 2, σημαίνει ότι υπάρχουν δύο δάχτυλα στην οθόνη, οπότε σταματάμε το χρόνο και θέτουμε την **PauseFingers** αληθή ώστε να μεταβούμε στην καρτέλα παύσης.

Στην τρισδιάσταση μεταβλητή **vel**, εισάγουμε στους άξονες x και z τις τιμές που έχουμε υπολογίσει για την κίνηση του ήρωα, και την πολλαπλασιάζουμε με την σταθερή ταχύτητα speed την οποία θέλουμε να έχει καθόλη τη διάρκεια του παιχνιδιού.

Αν ο CharacterController ακουμπάει το έδαφος μπορεί να πηδήξει, εφόσον δεν έχει χάσει όλες του τις ζωές. Θέτουμε αρχικά την ταχύτητα άλματος **vSpeed** ίση με μηδέν και ελέγχουμε αν ο χρήστης έχει σύρει το δάχτυλο του προς τα πάνω, που δηλώνεται από την **canJump** του script που ελέγχει την κατεύθυνση του δαχτύλου. Αν είναι αληθής πρέπει να πηδήξει και θέτουμε την ταχύτητα άλματος που θέλουμε, ενεργοποιούμε το animation

άλματος θέτοντας την παράμετρο **Jump** αληθή, αναπαράγουμε τον ήχο άλματος και θέτουμε την **canDJump** αληθή η οποία μας επιτρέπει να εκτελέσουμε διπλό άλμα.

Αν, ενώ βρισκόμαστε στο έδαφος, η **canSlide** είναι αληθής, δηλαδή ο χρήστης σύρει το δάχτυλο προς τα κάτω, θέτουμε την παράμετρο **isSlide** αληθή ώστε να ενεργοποιηθεί το animation με το οποίο κυλιέται στο έδαφος κάνοντας τούμπα, και ενεργοποιώντας τον αντίστοιχο ήχο.

Μόνο εφόσον δεν βρίσκεται στον αέρα και η canDJump είναι αληθής μπορεί να πραματοποιηθεί διπλό άλμα. Αν σύρει το δάχτυλό του προς τα πάνω για να πηδήξει υπό αυτές τις συνθήκες, θέτουμε την ταχύτητα διπλού άλματος στην vSpeed, κάνουμε αληθή την παράμετρο **DoubleJump** για να ενεργοποιηθεί το animation του διπλού άλματος και αναπαράγουμε τον αντίστοιχο ήχο. Αν έχει ξεκινήσει και πλησιάζει προς το έδαφος και δεν έχει επιλέξει διπλό άλμα ο χρήστης, τότε ενεργοποιείται το animation πτώσης από την παράμετρο Fall και αν καθώς πέφτει πηδήξει ενεργοποιείται το διπλό άλμα.

Από τη vSpeed αφαιρούμε κάθε καρέ τη τιμή **gravity** που έχουμε δηλώσει ως βαρύτητα με τη πάροδο του χρόνου των δευτερολέπτων, και εισάγουμε το αποτέλεσμά της στον άξονα y της vel που καθορίζει την κίνηση. Η κίνηση πραγματοποιείται από την **CharacterController.Move** με είσοδο την vel με την πάροδο του χρόνου.

Αν ο ήρωας δεν έχει φτάσει ακόμα στο σημείο που αναλαμβάνει ο χρήστης δράση και δεν έχει ενεργοποιηθεί ακόμα η κύρια κάμερα (showButtons είναι ψευδή), ο ήρωας τρέχει αυτόματα προς τα μπροστά με 180° στον άξονα των y. Δεν μπορεί ο χρήστης να τον μετακινήσει δεξιά και αριστερά οπότε η horizontal είναι ίση με μηδέν, και η vertical είναι ίση με -1. Στη μεταβλητή vel θέτουμε τις τωρινές τιμές της οριζόντιας και κάθετης κίνησης και την πολλαπλασιάζουμε με την ταχύτητα που θέλουμε και με τον χρόνο ανά δευτερόλεπτα εισάγοντάς την στην Move() για να πραγματοποιηθεί η κίνηση.

```
function Update ()
{
if(Controls.canPlay)
{
if(showButtons)
{
        distance = Vector3.Distance(gameObject.FindWithTag("Treasure").transform.position,
                                    transform.position);//the distance between the player and the treasure
if(canMove)                                         //only then we can control the movement
{       //if we chose to move by touching the screen buttons
        if(Controls.touchChoice && HealthRun.LIVES != 0)
        {
                if(amStraight)                          //STRAIGHT
```

```
		{
			transform.eulerAngles.y = 180;
			horizontal = -movementX;	//TURN left/right
			vertical = -1;
			if(Froze){
				horizontal = 0;
				vertical = 0;
			}
		}
		if(amRight)				//RIGHT
		{
			transform.eulerAngles.y = 270;
			horizontal = -1;			//TURN left/right
			vertical = movementX;
			if(Froze){
				horizontal = 0;
				vertical = 0;
			}
		}
		if(amLeft)				//LEFT
		{
			transform.eulerAngles.y = 90;
			horizontal = 1;			//TURN left/right
			vertical = -movementX;
			if(Froze){
				horizontal = 0;
				vertical = 0;
			}
		}
		if(amBackward)					//BACKWARD
		{
			transform.eulerAngles.y = 0;
			horizontal = movementX;		//TURN left/right
			vertical = 1;
			if(Froze){
				horizontal = 0;
				vertical = 0;
			}
		}
	}
	else if(Controls.tiltChoice && HealthRun.LIVES != 0)
	{//if we chose to move by tilting the device
		movex = 0;			// MOVE left / right
		//get the movement from the accelerometer of the mobile
```

```
iPx = iPhoneInput.acceleration.x;
if (Mathf.Abs(iPx) > moveThreshold)
{ // put a threshold to move the character if we pass it off
        movex = Mathf.Sign(iPx);      //Return value is 1 when iPx is positive or zero,
                                      //-1 when f is negative. just like forward.right
}
if(amStraight)                        //STRAIGHT
{
        transform.eulerAngles.y = 180;
        horizontal = -movex;          //TURN left/right
        vertical = -1;
        if(Froze){
                horizontal = 0;
                vertical = 0;
        }
}
if(amRight)                           //RIGHT
{
        transform.eulerAngles.y = 270;
        horizontal = -1;              //TURN left/right
        vertical = movex;
        if(Froze){
                horizontal = 0;
                vertical = 0;
        }
}
if(amLeft)                            //LEFT
{
        transform.eulerAngles.y = 90;
        horizontal = 1;               //TURN left/right
        vertical = -movex;
        if(Froze){
                horizontal = 0;
                vertical = 0;
        }
}
if(amBackward)                                //BACKWARD
{
        transform.eulerAngles.y = 0;
        horizontal = movex;           //TURN left/right
        vertical = 1;
        if(Froze){
                horizontal = 0;
                vertical = 0;
```

```
                }
            }
        }
        //TURN LEFT
        if(SwipeCtrlRun.canRotateRight  &&  HealthRun.LIVES != 0)
        {
            turnLeft = true;
        }
        //TURN RIGHT
        if(SwipeCtrlRun.canRotateLeft  &&  HealthRun.LIVES != 0)
        {
            turnRight = true;
        }
        if(HealthRun.LIVES == 0)                    //we have died and the viking reaches us
        {
            Froze = true;
            anim.SetBool("Die", true);
            playLooseSound = true;
        }
        if(Froze){
            horizontal = 0;
            vertical = 0;
        }

        //PAUSE IF TWO FINGER ARE ON THE SCREEN
        var fingerCount = 0;
            for (var touch : Touch in Input.touches) {
                if (touch.phase != TouchPhase.Ended &&
                                                    touch.phase != TouchPhase.Canceled)
                    fingerCount++;
            }       //Each entry represents a status of a finger touching the screen.
            if (fingerCount == 2)                   //User has 2 fingers touching the screen
            {
                Time.timeScale = 0;
                PauseFingers = true;                    //Pause
            }
        // MOVE, only the forward direction because we are interested to moving forward
        var vel: Vector3 = Vector3(horizontal, 0, vertical);
        vel *= speed;

        //JUMP
        if (controller.isGrounded  && HealthRun.LIVES != 0)  // We are grounded, so recalculate
        {
            vSpeed = 0; // grounded character has vSpeed = 0...
```

```
            if (SwipeCtrlRun.canJump)      //JUMP
            {           // unless it jumps:
                        vSpeed = jumpSpeed;
                        anim.SetBool("Jump", true);     //SetBool : Sets the value of a bool parameter.
                        playJumpSound = true;
                        canDJump = true;
                        SwipeCtrlRun.canJump = false;
            }

            //LEAN DOWN
            if(SwipeCtrlRun.canSlide)       //he can slide only if he is grounded
            {
                        anim.SetBool("isSlide", true);
                        playSlideSound = true;
                        SwipeCtrlRun.canSlide = false;
            }
        }
        //Double JUMP
        if (!controller.isGrounded && canDJump && SwipeCtrlRun.canJump )
        {           //User has double touched the screen, so double Jump
            vSpeed = dJumpSpeed;
            anim.SetBool("DoubleJump", true);      //DOUBLE JUMP
            playDoubleJumpSound = true;
            anim.SetBool("Fall", false);
            canDJump = false;
            SwipeCtrlRun.canJump = false;
        }
        else if (!controller.isGrounded && canDJump && !SwipeCtrlRun.canJump)
        {           //he jumps higher if we press the jump button after a while since the first one
            anim.SetBool("Fall", true);                 //FALLING
            if (!controller.isGrounded && canDJump && SwipeCtrlRun.canJump)
            {
                        vSpeed = dJumpSpeed;
                        anim.SetBool("DoubleJump", true);      //DOUBLE JUMP after FALLING
                        playDoubleJumpSound = true;
                        canDJump = false;
                        SwipeCtrlRun.canJump = false;
            }
         }
        // apply gravity acceleration to vertical speed:
        vSpeed -= gravity * Time.deltaTime;
        vel.y = vSpeed;                     // include vertical speed in vel
        // convert vel to displacement and Move the character:
        controller.Move(vel * Time.deltaTime );
```

```
}
}
else                                    //he can only run by itself
{
        transform.eulerAngles.y = 180;
        horizontal = 0;
        vertical = -1;
        // MOVE
        vel = Vector3(horizontal, 0, vertical);   //only running towards
        vel *= speed;
        controller.Move(vel * Time.deltaTime );
}
}
}
```

Στη συνάρτηση LateUpdate() πραγματοποιείται η στροφή 90° δεξιά και αριστερά. Αν η turnLeft είναι αληθής, στρίβουμε απότομα προς τα αριστερά θέτοντας τον y άξονα ίσο με 90° μέσω της **transform.Rotate** και θέτοντας πάλι ψευδείς τις μεταβλητές που μας οδήγησαν εδώ, turnLeft και canRotateRight, ώστε να μπορέσουμε να ξαναστρίψουμε. Αν η turnRight είναι αληθής, στρίβουμε απότομα προς τα δεξιά θέτοντας τον άξονα y ίσο με -90° αρχικοποιώντας τις turnRight και canRotateLeft.

Για να γνωρίζουμε πότε ο πιγκουίνος προχωράει προς τα μπροστά, ή προς τα πίσω, ή προς τα δεξιά, ή προς τα αριστερά συγκριτικά με τον κόσμο και όχι με αυτόν, ορίζουμε τις κατευθύνσεις αναλόγως τις μοίρες που έχει κάθε φορά στον άξονα y. Έτσι μπορούμε να γνωρίζουμε προς τα που θα πρέπει να στρίψει κάθε φορά όσες φορές και αν στρίψει. Οι τιμές που έχει ο κάθε άξονας κατά τη διάρκεια της κίνησής του δεν είναι ένας ακέραιος ή συγκεκριμένος αριθμός, καθώς μεταβάλλονται αναλόγως την κίνηση και του animation γύρω από μια σταθερή τιμή προσεγγίζοντάς την. Αν ο άξονας y έχει περίπου γωνία 180 μοιρών, δηλαδή έχει τιμή Transform.eulerAngles μεγαλύτερη του 170 και μικρότερη του 190, τότε κατευθύνεται προς τα μπροστά θέτοντας την amStraight αληθή και τις άλλες μεταβλητές κατευθύνσεων ψευδείς. Αν έχει γωνία περίπου ίση με 270 μοίρες, δηλαδή με τιμή μεγαλύτερη του 260 και μικρότερη του 280, κατευθύνεται προς τα δεξιά και θέτουμε την amRight αληθή και τις άλλες ψευδείς. Αν έχει γωνία περίπου ίση με 90 μοίρες, μεγαλύτερη του 80 και μικρότερη του 100, κατευθύνεται προς τα αριστερά και θέτουμε την amLeft αληθή και τις άλλες ψευδείς. Προς τα πίσω μπορεί να κατευθυνθεί ο ήρωας σε όλες τις πίστες εκτός την πρώτη. Έτσι ελέγχουμε σε ποια πίστα της RunLand βρισκόμαστε αριθμητικά μέσω της μεταβλητής **runLand**, και αν δεν βρισκόμαστε στην πρώτη, αν έχει μηδέν μοίρες ο άξονας y δηλαδή μεγαλύτερες από το -10 και μικρότερες από το 10, ή αν έχει 360 μοίρες, δηλαδή μεγαλύτερες από 350 και μικρότερες από 370, τότε κινείται προς τα πίσω και θέτουμε την **amBackward** αληθή και τις άλλες ψευδείς.

Επίσης ελέγχουμε αν τερματίστηκε επιτυχώς η πίστα για να υπολογίσουμε αποθηκεύσουμε τα δεδομένα απόδοσης της βαθμολογίας της κάθε πίστας, για να εμφανίζονται στο μενού των πιστών μέσω της **PlayerPrefs**. Η βαθμολογία ορίζεται από τον αριθμό των νομισμάτων και των ψαριών μπόνους που συλλέχτηκαν στο παιχνίδι. Κάθε φορά αποθηκεύουμε τη μεγαλύτερη βαθμολογία για να αναγράφεται η κάθε επίτευξη ενός νέου ρεκόρ.

```
function LateUpdate()
{
if(Controls.canPlay)
{
        //UNSMOOTH TURNS
        if(turnLeft)
        {
                transform .Rotate(Vector3(0, 90, 0));
                turnLeft = false;
                SwipeCtrlRun.canRotateRight = false;
         }
          if(turnRight)
          {
                transform .Rotate(Vector3(0, -90, 0));
                turnRight = false;
                SwipeCtrlRun.canRotateLeft = false;
        }
          //STRAIGHT
        if(transform.eulerAngles.y >= 170 && transform.eulerAngles.y <= 190)//around 180 degrees
        {
                amStraight = true;
                amRight = false;
                amLeft = false;
                amBackward = false;
        }
        //RIGHT
        if(transform.eulerAngles.y >= 260 && transform.eulerAngles.y <= 280)//around 270 degrees
        {
                amStraight = false;
                amRight = true;
                amLeft = false;
                amBackward = false;
        }
        //LEFT
        if(transform.eulerAngles.y >= 80 && transform.eulerAngles.y <= 100)  //around 90 degrees
        {
                amStraight = false;
```

```
                amRight = false;
                amLeft = true;
                amBackward = false;
        }
        if(runLand == 2 ||runLand == 3 || runLand == 4 || runLand == 5 || runLand == 6 ||
                runLand == 7 || runLand == 8)
        {
                //BACKWARD
                if((transform.eulerAngles.y >= -10 && transform.eulerAngles.y <= 10) ||
                (transform.eulerAngles.y >= 350 && transform.eulerAngles.y <= 370))
                {//around 180 degrees
                        amBackward = true;
                        amStraight = false;
                        amRight = false;
                        amLeft = false;
                }
        }
}
        //WIN           //DATA
        if(hasWon)              //we save the data for each level
        {
                //SNOWLAND
                if(runLand == 1)
                {
                        PlayerPrefs.SetInt("WinRun1", 1);
                        //it will automatically check if this value is more then the previous and it will
                        //save automatically on save points.
                        if (PlayerPrefs.GetInt("rCoin1")<coinsR)
                        {
                                PlayerPrefs.SetInt("rCoin1", coinsR);
                        }
                        if(PlayerPrefs.GetInt("rFishBonus1")<FishControl.fishCounter
                        {       )//save the fishes
                                PlayerPrefs.SetInt("rFishBonus1", FishControl.fishCounter);
                        }
                        //to find which fishes we have collected
                        if(FishR1)
                        {
                                PlayerPrefs.SetInt("r1GotFish1",1);
                        }
                        else
                        {
                                PlayerPrefs.SetInt("r1GotFish1",0);
                        }
```

```
                        if(FishR2)
                        {
                                PlayerPrefs.SetInt("r1GotFish2",1);
                        }
                        else
                        {
                                PlayerPrefs.SetInt("r1GotFish2",0);
                        }
                        if(FishR3)
                        {
                                PlayerPrefs.SetInt("r1GotFish3",1);
                        }
                        else
                        {
                                PlayerPrefs.SetInt("r1GotFish3",0);
                        }
                        //SCORE
                        ScoreCounter = coinsR + FishRBonus ;             // each coin is 1 point
                        if (PlayerPrefs.GetInt("rHighScore1")<ScoreCounter)
                        {
                                PlayerPrefs.SetInt("rHighScore1", ScoreCounter);
                        }
                        PlayerPrefs.Save();
                }
        }
}
```

Στη συνάρτηση OnTriggerEnter() ελέγχουμε με ποια αντικείμενα στην πίστα έρχεται σε επαφή ο ήρωας και ενεργούμε κατάλληλα. Αν ο πιγκουίνος συλλέξει ένα μαγικό σμαράγδι γίνεται αθάνατος από τους βίκινγκς και τα εμπόδια θέτοντας την **cannotDie** αληθή και καλείται η συνάρτηση DoubleSize() για να αποκτήσει ειδικές δυνάμεις ο ήρωας. Κάθε φορά που συλλέγει ένα νόμισμα αυξάνουμε κατά ένα τον αριθμό της μεταβλητής **coinsR** για να υπολογίσουμε το σύνολο των νομισμάτων που συλλέχθηκαν. Για κάθε ψάρι που συλλέγει θέτουμε την αντίστοιχη μεταβλητή, **FishR1** ή **FishR2** ή **FishR3**, αληθή για να δώσουμε έντονο χρώμα στα εικονίδια των ψαριών και προσθέτουμε στη μεταβλητή **FishRBonus** τους βαθμούς που αντιστοιχούν σε κάθε ψάρι.

Ορίζουμε τη μεταβλητή **isHit** η οποία δηλώνει αν ο πιγκουίνος πιάστηκε από τον βίκινγκ, και την **gotHit** που δηλώνει ότι ο ήρωας έπεσε πάνω σε ένα εμπόδιο. Η μεταβλητή **runViking** χρησιμοποιείται για να ξέρουμε πότε ο βίκινγκ πρέπει να πλησιάσει περισσότερο τον ήρωα, και η **countHits** για να μετράμε τις φορές που έπεσε ο ήρωας πάνω σε ένα αντικείμενο. Για να γνωρίζουμε με ποιο αντικείμενο έρχεται σε επαφή κάθε φορά ο ήρωας, έχουμε θέσει αντίστοιχες ετικέτες στο καθένα, όπως το όνομα "**Viking**" στον βίκινγκ που τον

κυνηγάει, και το όνομα "**Obstacle**" σε κάθε εμπόδιο της πίστας. Όταν πέφτει σε ένα εμπόδιο και χάνει μια ζωή, και όταν τον πιάσει ο βίκινγκ και χάσει όλες τις ζωές του, η συσκευή δονείται με την **Handheld.Vibrate()**.

Αν τον πιάσει ο βίκινγκ, ανεξάρτητα από τις ζωές που έχει, αν δεν έχει πάρει το μαγικό σμαράδι (cannotDie είναι ψευδή) πεθαίνει. Θέτουμε τη Froze αληθή για να παγώσουμε τη κίνηση του πιγκουίνου, και την isHit αληθή ώστε να ενεργοποιήσουμε τις ενέργειες που εκτελούνται όταν πεθαίνει μία μόνο φορά. Ενεργοποιούμε τη δόνηση του κινητού, θέτουμε την isHit ψευδή για να μην επαναληφθούν οι διαδικασίες, την **Die** αληθή για να ενεργοποιηθεί το animation με το οποίο πεθαίνει, μηδενίζουμε τις ζωές του και περιμένουμε 1,5 δευτερόλεπτα για να ολοκληρωθεί το κλιπ.

Αν έρθει σε επαφή με κάποιο εμπόδιο ενώ η cannotDie είναι ψευδής, δηλαδή δεν έχει πάρει κάποιο σμαράγδι, θέτουμε την gotHit αληθή για να ενεργοποιήσουμε μία μόνο φορά τις διαδικασίες που εκτελούνται όταν χτυπάει. Ενεργοποιούμε τη δόνηση, αρχικοποιούμε τη gotHit, θέτουμε την runViking αληθή ώστε να προφτάσει ο βίκινγκ τον πιγκουίνο, αυξάνουμε την countHits κατά ένα για να υπολογίσουμε τον αριθμό των χτυπημάτων από αντικείμενα, θέτουμε τη Froze αληθή και τη canMove ψευδή ώστε να σταματήσει να κινείται ο ήρωας για λίγο επειδή χτύπησε, ενεργοποιούμε το animation με το οποίο χτυπάει μέσω της παραμέτρου **GotHit** και περιμένουμε 0.8 δευτερόλεπτα μέχρι να ολοκληρωθεί το κλιπ. Έπειτα ενεργοποιούμε πάλι την κίνηση του ήρωα επιστρέφοντας στις μεταβλητές Froze και canMove τις αρχικές τους τιμές, καλείται η συνάρτηση **OnObstacle()** με την οποία αφαιρείται μα ζωή, και καταστρέφουμε το εμπόδιο στο οποίο έπεσε ο ήρωας ώστε να μπορεί να προχωρήσει.

Στην αρχή της πίστας που ο ήρωας βγαίνει έξω από το ιγκλού και κατευθύνεται προς το λαβύρινθο είναι ενεργοποιημένη η δεύτερη κάμερα. Μόλις φτάσει σε ένα συγκεκριμένο σημείο στο οποίο έρχεται σε επαφή με ένα αόρατο αντικείμενο που βρίσκεται στο δρόμο του, θέτουμε την **changeCam** αληθή για να ενεργοποιηθεί η κύρια κάμερα και την showButtons αληθή για να επιτρέψουμε τον έλεγχο της κίνησης από το χρήστη.

Μόλις φτάσει στο θησαυρό, παγώνουμε την κίνησή του ώστε να σταματήσει να προχωράει, αναπαράγουμε τη μουσική της νίκης, ενεργοποιούμε το animation με το οποίο πανηγυρίζει μέσω της παραμέτρου **Winner** και θέτουμε τη hasWon αληθή για να αποθηκευτούν τα δεδομένα της πίστας.

Αν πέσει στο κενό και έρθει σε επαφή με τα σύννεφα, παγώνουμε τη κίνησή του, ενεργοποιούμε το animation με το οποίο πέφτει μέσω της παραμέτρου DeadFall και μηδενίζουμε τις ζωές του και η συσκευή δονείται.

```
function OnTriggerEnter( hit : Collider )
{
```

```
if(Controls.canPlay)
{
	//DIEING from Viking
	if(hit.gameObject.tag == "Viking" && HealthRun.LIVES != 0)
	{
		if(!cannotDie)				//he can die
		{
			Froze=true;
			if(isHit == false)
			{
				Handheld.Vibrate ();		// vibrate the device
				isHit = true;
				anim.SetBool("Die", true);		//die animation
				HealthRun.LIVES = 0;			//if he catches us we die
				yield WaitForSeconds(1.5);		//assuming it takes 1.5 seconds
								//to play the animation
			}
		}
	}
	//GOT HIT
	if(hit.gameObject.tag == "Obstacle" )			//when we hit an obstacle
	{
		if(!cannotDie)				//we can get hurt if we haven't got power up
		{
			if(gotHit == false)
			{
				Handheld.Vibrate ();		// vibrate the device
				gotHit = true;		//so cannot gget hit twice by the same obstacle
				runViking = true;	//run viking towards us
				countHits = countHits + 1;//increase the number of hits by objects
				Froze=true; //cannot move immediately when we fell into the object
				canMove = false;
				anim.SetBool("GotHit", true);		//Got Hit animation
				yield WaitForSeconds(0.8);//wait 0.5 second until he can run again
				Froze=false;				//we can move again
				canMove = true;
				OnObstacle();		//function coroutine for remove one life
				Destroy(hit.gameObject);	//if we have lost one life from it we
								//destroy it so he can pass it
			}
		}
	}
}
}
```

Όταν καλείται η συνάρτηση **OnObstacle()** αφαιρούμε μια ζωή από τον πιγκουίνο καθώς έπεσε πάνω σε κάποιο εμπόδιο, και περιμένουμε λίγο χρόνο μέχρι να καταστραφεί πρώτα το εμπόδιο πριν θέσουμε πάλι την gotHit ψευδή, γιατί αλλιώς μπορεί να χτυπηθεί ξανά από το ίδιο εμπόδιο και να αφαιρεθούν παραπάνω από μία ζωές.

Η συνάρτηση **DoubleSize()** καλείται όταν πάρει το μαγικό σμαράγδι με το οποίο γίνεται αθάνατος. Ενεργοποιείται μια πράσινη ασπίδα γύρω από τον ήρωα για οχτώ δευτερόλεπτα και δεν μπορεί να πεθάνει. Μετά το χρόνο αυτόν η cannotDie γίνεται πάλι ψευδής για να μπορεί να πεθάνει και απενεργοποιείται η ασπίδα.

```
function DoubleSize()                                   //double our size with power up
{
        gameObject.Find("CannotDie").renderer.enabled  = true;      //enable the shield
        yield WaitForSeconds(8);                                    //lasts for 8 seconds
        cannotDie = false;                                          //he is vulnerable again
        gameObject.Find("CannotDie").renderer.enabled  = false;     //disable the shield
}
```

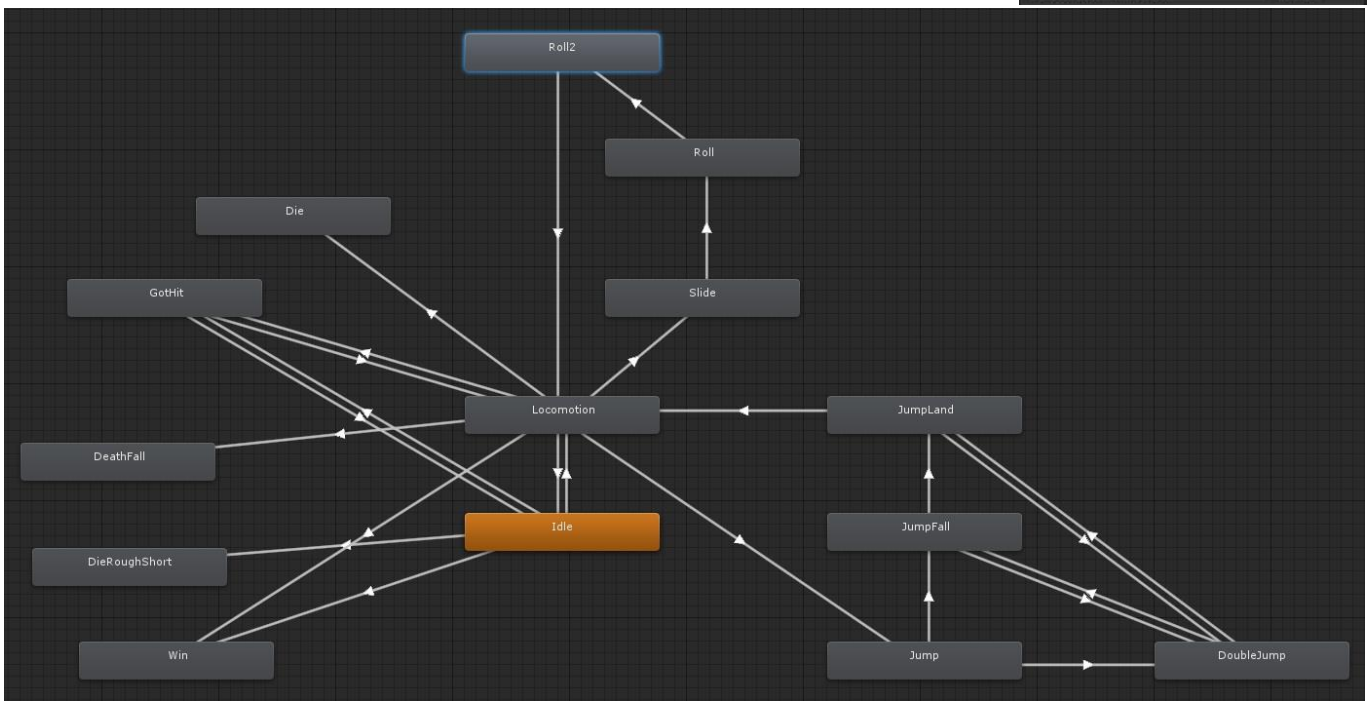
Τα Animations του ήρωα ελέγχονται από τον penguinAnimCtrlRun ρυθμίζοντας τις τιμές των παραμέτρων που οδηγούν στις καταστάσεις των animations μέσα από τον κώδικα. Αρχικά ο ήρωας είναι στην κατάσταση **Idle**, και μόλις ξεκινάει το παιχνίδι μπαίνει στην κατάσταση **Locomotion** αφού αποκτάει αμέσως ταχύτητα. Η Locomotion δεν χρησιμοποιεί το περπάτημα καθώς η ταχύτητα του πιγκουίνου είναι σταθερή ώστε να τρέχει καθόλη τη διάρκεια της πίστας. Έτσι περιλαμβάνει μόνο το Blend Tree **Runs** από το οποίο αναλόγως την κατεύθυνση **Direction** που έχει, τρέχει ευθεία **run**, στρίβει δεξιά τρέχοντας **straferight**, ή στρίβει αριστερά τρέχοντας **strafeleft**.

Αν πηδήξει ενώ τρέχει, ενεργοποιείται από τη Locomotion το animation **Jump**, το οποίο, με την ολοκλήρωσή του, διαδέχονται με τη σειρά το animation της πτώσης **JumpFall** και της προσγείωσης **JumpLand**. Αν σε οποιαδήποτε από αυτές τις καταστάσεις επιλεχθεί διπλό άλμα ενεργοποιείται το αντίστοιχο animation **DoubleJump** το οποίο διαδέχονται οι επόμενες καταστάσεις άλματος, και επιστρέφει στη Locomotion για να συνεχίσει να τρέχει.

Αν επιλέξει να κυλιστεί στο έδαφος κάνοντας τούμπα ενώ τρέχει, ενεργοποιείται από τη Locomotion το animation **Slide** με το οποίο παίρνει φόρα και κλίση για να ετοιμαστεί να κυλιστεί και ολοκληρώνοντας το διαδέχεται το animation με το οποίο κάνει τούμπα στο έδαφος **Roll** δύο φορές ώστε να διανύσει μεγάλη απόσταση, και επιστρέφει στη Locomotion για να συνεχίσει να τρέχει.

Αν πέσει πάνω σε εμπόδιο ενεργοποιείται το animation **GotHit** με το οποίο χτυπάει και επιστρέφει στην κατάσταση στην οποία χτυπήθηκε για να συνεχίζει να τρέχει, είτε τη Locomotion είτε την Idle η οποία είναι η default κατάσταση, όπου είναι καλύτερο να επιστρέφουμε πάντα σε αυτήν για να μην υπάρξει μπέρδεμα.

Αν τον πιάσει ο βίκινγ ή πέσει τρεις φορές πάνω σε ένα εμπόδιο πεθαίνει με το animation **Die**. Αν πέσει στο κενό ενεργοποιείται το **DeathFall** με το οποίο πέφτει, και αν φτάσει στο θησαυρό ενεργοποείται το **Win** με το οποίο πανηγυρίζει. Όταν βρεθεί σε αυτές τις καταστάσεις δεν χρειάζεται να γυρίσει πίσω στη Locomotion ή την Idle γιατί τελειώνει ο γύρος του παιχνιδιού και αρχίζει από την αρχή.



Ορίζουμε τη μεταβλητή **currentBaseState** για να ελέγχουμε τη κατάσταση στην οποία βρισκόμαστε. Για να μεταφερθούμε από την κατάσταση Idle στη Locomotion πρέπει να έχει αναπτύξει ταχύτητα Speed την οποία διαβάζουμε από τη μεταβλητή **v**, και όταν είναι μέσα στη Locomotion στρίβει αναλόγως την διεύθυνση της κίνησης Direction που διαβάζεται από τη μεταβλητή **h**.

Στη συνάρτηση **FixedUpdate ()**, ελέγχουμε αν η showButtons είναι αληθής, δηλαδή αν ο πιγκουίνος έχει φτάσει στο σημείο αλλαγής της κάμερας στο οποίο αναλαμβάνει το χειρισμό ο χρήστης. Αν ο χρήστης έχει επιλέξει να χειρίζεται την κίνηση του ήρωα μέσω κουμπιών αφής,

ορίζουμε στην h τις τιμές των κουμπιών movementX, ενώ αν έχει επιλέξει να τη χειρίζεται μέσω της κλίσης της συσκευής ορίζουμε στην h τις τιμές του επιταχυνσιομέτρου movex. Και στις δύο περιπτώσεις έχει αναπτυγμένη ταχύτητα, οπότε θέτουμε την v ίση με 1. Αν δεν έχει φτάσει ακόμα στο σημείο αλλαγής της κάμερας, μόνο τρέχει και δεν μπορεί να στρίψει, οπότε θέτουμε την h ίση με μηδέν και την v ίση με 1.

Όταν βρισκόμαστε μέσα σε μια κατάσταση και δεν είμαστε στη φάση της μετάβασης, αρχικοποιούμε τις παραμέτρους που μας οδήγησαν σε αυτήν την κατάσταση θέτοντάς τες ψευδείς, ώστε να μπορέσουμε να ξανά εισαχθούμε σε αυτήν.

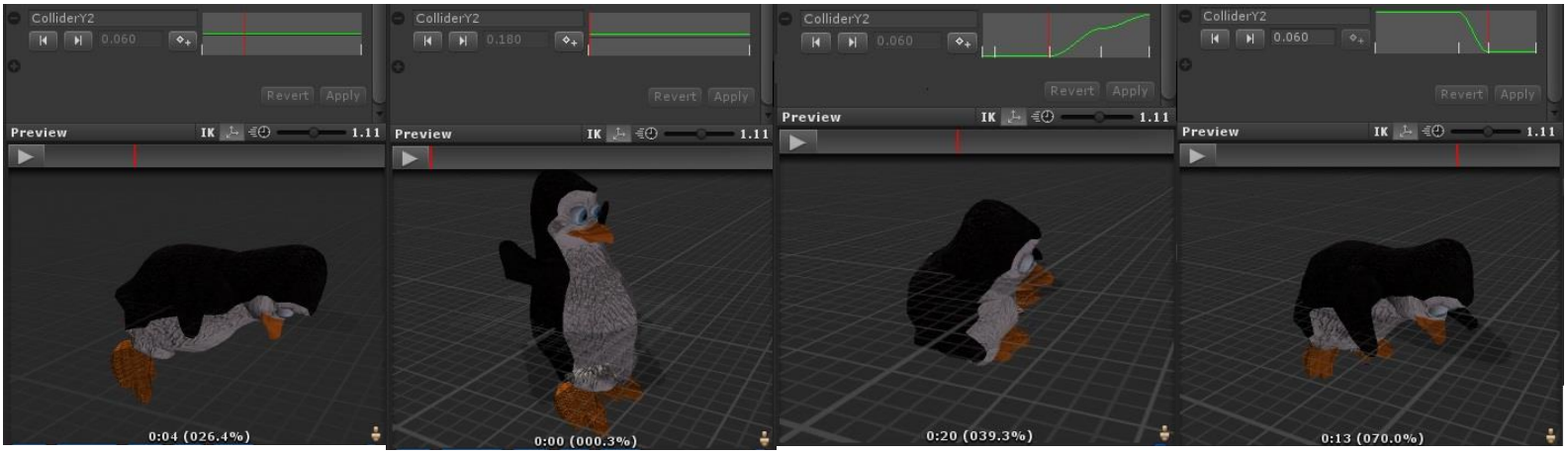
Κατά τη διάρκεια των animations πρέπει να προσέχουμε το μέγεθος και τη θέση του CharacterController του πιγκουίνου και του Sphere Collider του μαγνήτη που είναι προσκολλημένος στον ήρωα, ώστε να αντιστοιχούν στο σχήμα που παίρνει η μορφή του χαρακτήρα για να αλληλεπιδρά σωστά με το περιβάλλον του.

Όταν πηδάει, ή όταν κάνει τούμπα για να αποφύγει κάποιο εμπόδιο αναδιπλώνεται για να μην τα ακουμπήσει. Αν οι επιταχυντές παραμένουν στο αρχικό τους μέγεθος και θέση, θα ακουμπήσει ο ήρωας στο εμπόδιο ανεξάρτητα από το πως φαίνεται στο animation. Για αυτό ελέγχουμε το animation το οποίο εκτελείται και αλλάζουμε την κλίμακα και τη θέση των επιταχυντών.

Στο απλό άλμα ο πιγκουίνος είναι σε όρθια θέση, οπότε δεν χρειάζεται να επέμβουμε. Όμως στο διπλό άλμα εκτελεί μια τούμπα στον αέρα, οπότε μικραίνουμε την τιμή του ύψους του CharacterController μέσω της **CharacterController.height**, και θέτουμε την **magnetColJump** αληθή ώστε να μικρύνουμε τον επιταχυντή του μαγνήτη από το script που τον ελέγχει.

Στη συνέχεια αρχίζει να πέφτει προς το έδαφος και προσγειώνεται. Η επιφάνεια που καλύβει δεν είναι τόσο μικρή όσο όταν έκανε την τούμπα στον αέρα, αλλά εξακολουθεί να είναι μικρή και να αυξάνεται σταδιακά, και πρέπει αντίστοιχα να αυξάνονται τα μεγέθη των επιταχυντών. Επειδή αλλάζει το μέγεθός τους συνεχώς και δεν έχουμε μία μόνο σταθερή τιμή, χρησιμοποιούμε τις καμπύλες **curves** του Mecanim παράλληλα με το animation στις οποίες ορίζουμε κλειδιά στα σημεία που αλλάζει το animation και θέτουμε τη τιμή του ύψους που θέλουμε να έχει ο επιταχυντής στη θέση αυτή. Λαμβάνουμε τις τιμές της καμπύλης **ColliderY2** με την **Animator.GetFloat** και τις θέτουμε στο ύψος των επιταχυντών. Μόλις ολοκληρωθεί και το animation της προσγείωσης, επαναφέρουμε τους επιταχυντές στο αρχικό τους ύψος.

Όταν πεθαίνει ο ήρωας και ξαπλώνει στο έδαφος, μικραίνουμε επίσης το ύψος των επιταχυντών για να μην τον εμποδίζουν να το ακουμπήσει και βρεθεί ξαπλωμένος στον αέρα.



Επίσης, κατά τη κύλιση στο έδαφος πρέπει να προσέξουμε να μην ακουμπήσουν οι επιταχυντές στο εμπόδιο από πάνω, καθώς ο ήρωας περνάει από κάτω σκύβοντας και κάνοντας τούμπα. Και για τα τρία animations που εκτελούνται, ξεκινάει να σκύβει και κάνει τούμπα δύο φορές, μικραίνουμε την τιμή του ύψους τους αλλά και της θέσης τους στον άξονα y με την **CharacterController.center**, καθώς ο ήρωας πλησιάζει πολύ το έδαφος και το κέντρο τους μετατοπίζεται πιο χαμηλά από ότι όταν είναι όρθιος. Με την ολοκλήρωση του τελευταίο animation της τούμπας, επαναφέρουμε το ύψος και τη θέση των επιταχυντών στην αρχική τους θέση ώστε να μην πέσει ο ήρωας από το έδαφος.

```
function FixedUpdate ()
{
if(Controls.canPlay)
{
        if(showButtons)                                    //we can move
        {
                if(Controls.touchChoice)
                {
                        var h : float = movementX;      // setup h variable as our horizontal input axis
                        var v : float= 1;               // setup v variables as our vertical input axis
                }
                if(Controls.tiltChoice)
                {
                        h = movex;                      // setup h variable as our horizontal input axis
                        v = 1;                          // setup v variables as our vertical input axis
                }
        }
        else
        {
                h = 0;
                v = 1;
        }
```

```
        //set our animator's parameter 'Speed' equal to the vertical input axis
        anim.SetFloat("Speed", v);
        //set animator's parameter Direction equal to horizontal input axis
        anim.SetFloat("Direction", h);
        currentBaseState = anim.GetCurrentAnimatorStateInfo(0);
        // set our currentState variable to the current state of the Base Layer (0) of animation
        if (currentBaseState.nameHash == dJumpState)
        {
                if(!anim.IsInTransition(0))
                {
                        controller.height = 0.05;
                        magnetColJump = true;
                        anim.SetBool("DoubleJump", false);
                }
        }
        if (currentBaseState.nameHash == jumpFallState)
        {
                if(!anim.IsInTransition(0))
                {       //set the collider height to a float curve in the clip called ColliderY2
                        controller.height = anim.GetFloat("ColliderY2");
                        anim.SetBool("Fall", false);
                }
        }
        if (currentBaseState.nameHash == jumpLandState)
        {
                if(!anim.IsInTransition(0))
                {
                        controller.height = anim.GetFloat("ColliderY2");
                }
                else
                {
                        controller.height = 0.18;
                        magnetColJump = false;
                }
        }
        if (currentBaseState.nameHash == deadState)
        {
                if(!anim.IsInTransition(0))
                {
                        controller.height = 0.05;
                        anim.SetBool("Dead", false);
                }
        }
        if (currentBaseState.nameHash == slidingState)
```

```
        {
                if(!anim.IsInTransition(0))
                {
                        controller.height = 0.05;
                        anim.SetBool("isSlide", false);
                        magnetCol = true;               //reduce the size of the magnet collider
                }
        }
        if (currentBaseState.nameHash == rollingState)
        {
                if(!anim.IsInTransition(0))
                {
                        controller.height = 0.05;
                        controller.center.y = 0.0;
                        magnetCol = true;               //reduce the size of the magnet collider
                }
        }
        if (currentBaseState.nameHash == rolling2State)
        {
                if(!anim.IsInTransition(0))
                {
                        controller.height = 0.05;
                        controller.center.y = 0.0;
                        magnetCol = true;               //reduce the size of the magnet collider
                }
                else
                {
                        controller.height = 0.18;
                        controller.center.y = 0.03;
                        magnetCol = false;              //increase it to its initial size
                }
        }
}
}
```

Από το script του μαγνήτη **MagnetControl.js** ελέγχουμε πότε πρέπει να αναπαραχθεί κάποιος ήχος με την **audio.PlayOneShot** και πότε να αλλάξουμε το μέγεθος του collider με τη **SphereCollider.radius** και τη θέση του με την **SphereCollider.center**, διαβάζοντας τις μεταβλητές που ορίσαμε στο RunMoveMobile.js οι οποίες σηματοδοτούν την αντίστοιχη ενέργεια.

Ο έλεγχος της κατεύθυνσης της κίνησης του δαχτύλου γίνεται μέσω του **SwipeCtrlRun.js**. Ορίζουμε μεταβλητές τις μεταβλητές **canJump**, **canRotateLeft**,

canRotateRight, και **canSlide**, με τις οποίες ενημερώνουμε την κλάση RunMoveMobile για το αν ο ήρωας πρέπει να πηδήξει, να στρίψει αριστερά ή δεξιά, ή να κάνει τούμπα. Ορίζουμε και μια μεταβλητή **SwipeID** για να ξέρουμε πότε πραγματοποιείται μετακίνηση του δαχτύλου και αρχικά την θέτουμε ίση με -1 που σημαίνει ότι δεν υπάρχει κανένα άγγιγμα, και τη μεταβλητή **StartPos** για να κρατάμε την αρχική θέση ενός αγγίγματος. Στη συνάρτηση Start() αρχικοποιούμε τις τιμές των μεταβλητών.

Στη συνάρτηση Update() για κάθε καρέ ελέγχουμε κάθε άγγιγμα **T** στην οθόνη με την **Input.touches** και στην μεταβλητή **P** αποθηκεύουμε τη θέση στην οποία έγινε. Αν το άγγιγμα μόλις ξεκίνησε **TouchPhase.Began** και δεν υπάρχει άλλο άγγιγμα στη οθόνη με την SwipeID να είναι ίση με -1, τότε θέτουμε στην SwipeID το μοναδικό άγγιγμα που πραγματοποιήθηκε **fingerId**, και στην StartPos αποθηκεύουμε τη θέση του αγγίγματος τη στιγμή που ξεκίνησε.

Αν δεν ξεκίνησε τώρα το άγγιγμα και είναι το μοναδικό άγγιγμα στην οθόνη fingerId που αποθηκεύσαμε στην SwipeID, αποθηκεύουμε στη μεταβλητή **delta** τη διαφορά της θέσης του αγγίγματος που έχει τώρα από την αρχική του θέση. Αν το άγγιγμα μετακινήθηκε **TouchPhase.Moved** και αν το μήκος της μετακίνησης **Vector3.magnitude** ήταν μεγαλύτερο από την ελάχιστη απόσταση **minMovement** που πρέπει να μετακινηθεί για να πραγματοποιηθεί μια ενέργεια, τότε κάνουμε πάλι την SwipeID ίση με -1 για να εντοπίσουμε την επόμενη κίνηση δαχτύλου και ελέγχουμε ποιά ενέργεια θα πραγματοποιηθεί. Αν το μήκος της απόστασης σε απόλυτη τιμή **Mathf.Abs** που διένυσε το δάχτυλο στον άξονα x είναι μεγαλύτερο από αυτό στον άξονα y σε απόλυτη τιμή, σημαίνει πως το δάχτυλο κινήθηκε οριζόντια. Αν η θέση delta έχει θετική τιμή ως προς τον άξονα x, ο χρήστης έσυρε το δάχτυλό του προς τα δεξιά και θέτουμε την canRotateRight αληθή για να στρίψει προς τα δεξιά. Αλλιώς αν έχει αρνητική τιμή ο χρήστης έσυρε το δάχτυλό του προς τα αριστερά και θέτουμε τη canRotateLeft αληθή για να στρίψει ο ήρωας προς τα αριστερά.

Αλλιώς αν η απόλυτη τιμής της θέση στον άξονα x είναι μικρότερη από αυτήν στον άξονα y, ο χρήστης έσυρε το δάχτυλό του κάθετα. Αν η θέση delta έχει θετική τιμή στον άξονα y, ο χρήστης κίνησε το δάχτυλό του προς τα πάνω και θέτουμε την canJump αληθή για να πηδήξει ο ήρωας. Αν η θέση delta έχει αρνητική τιμή στον άξονα y, ο χρήστης κίνησε το δάχτυλό του προς τα κάτω και θέτουμε την canSlide αληθή για να κάνει τούμπα.

Αν το άγγιγμα ακυρωθεί **TouchPhase.Canceled** ή τελειώσει **TouchPhase.Ended** θέτουμε την SwipeID ίση με -1 για να εντοπίσουμε το επόμενο άγγιγμα.

```
#pragma strict
/*
	Script that detects the swipe movements of our finger on the screen
*/
var StartPos : Vector2;
```

```
var SwipeID : int = -1;
var minMovement : float = 20.0f;
static var canJump : boolean = false;
static  var canRotateLeft : boolean = false;
static var canRotateRight : boolean = false;
static var canSlide : boolean = false;

// Use this for initialization
function Start ()
{
	canJump = false;
	canRotateLeft = false;
	canRotateRight = false;
	canSlide = false;
}
// Update is called once per frame
function Update ()
{
	for (var T : Touch in Input.touches) {
		var P = T.position;
		if (T.phase == TouchPhase.Began && SwipeID == -1) {
			SwipeID = T.fingerId;
			StartPos = P;
		} else if (T.fingerId == SwipeID) {
			var delta = P - StartPos;
			if (T.phase == TouchPhase.Moved && delta.magnitude > minMovement) {
				SwipeID = -1;
				if (Mathf.Abs (delta.x) > Mathf.Abs (delta.y)) {
					if (delta.x > 0) {
					Debug.Log ("Swipe Right Found");
					canRotateLeft = false;
					 canRotateRight = true;
					} else {
					Debug.Log ("Swipe Left Found");
					canRotateLeft = true;
					canRotateRight = false;
					}
				}
				else {
					if (delta.y > 0) {
					Debug.Log ("Swipe Up Found");
					canRotateLeft = false;
					canRotateRight = false;
					canSlide = false;
```

```
                                        } else {
                                        Debug.Log ("Swipe Down Found");
                                         canSlide = true;
                                        canJump = false;
                                        canRotateLeft = false;
                                        canRotateRight = false;
                                        }
                                }
                        } else if (T.phase == TouchPhase.Canceled || T.phase == TouchPhase.Ended)
                                SwipeID = -1;
                }
        }
}
```

5.7.4 Υλοποίηση των καμερών

Υλοποίηση δεύτερης κάμερας

Το παιχνίδι ξεκινάει με ένα μικρό βίντεο στο οποίο ο πιγκουίνος βγαίνει τρέχοντας από το ιγκλού και από πίσω τον ακολουθεί ο βίκινγκ. Το σημείο εκκίνησης στο οποίο είναι στραμμένο το βίντεο, είναι στην αντίθετη κατεύθυνση από αυτή που θα κινηθεί ο ήρωας στο παιχνίδι. Για να είναι φωτεινό και το βίντεο αλλά και η πίστα, έχουμε αρχικά το φώς Directional Light στραμμένο προς το ιγκλού. Μια κάμερα δείχνει τον ήρωα από ψηλά βγαίνοντας από το ιγκλού, στη συνέχεια κατεβαίνει και τον δείχνει από κοντά από την μπροστινή του μεριά, και όταν φτάσει σε ένα συγκεκριμένο σημείο, ενεργοποιείται η κύρια κάμερα η οποία βρίσκεται από πίσω του και στρέφουμε το φως προς την πίστα. Η πρώτη κάμερα ελέγχεται από το **CamSwitcherRunMobile.js**.

Ορίζουμε δύο Camera μεταβλητές, **mainCam** και **introCam**, στις οποίες θέτουμε τις δύο κάμερες για να μπορέσουμε να τις χειριστούμε. Στη συνάρτηση Start() ενεργοποιούμε την αρχική κάμερα και απενεργοποιούμε την κύρια με την **Behaviour.enabled**, γιατί θέλουμε μόλις ξεκινάει το παιχνίδι να βλεπουμε τον ήρωα από μπροστά. Στη συνάρτηση Update() ελέγχουμε πότε ο ήρωας φτάνει στο σημείο αλλαγής των καμερών, όπου απενεργοποιούμε τη δεύτερη κάμερα, ενεργοποιούμε την κύρια και στρέφουμε το φως κατάλληλα με την **Transform.eulerAngles** ώστε να φωτιστεί η πίστα.

Η αρχική κάμερα δεν είναι σταθερή, αλλά από κάποιο χρόνο και μετά πλησιάζει τον πιγκουίνο. Τη θέση στην οποία κατευθύνεται την ορίζουμε με ένα κενό GameObject, ελέγχουμε την κίνησή της μέσα από τη συνάρτηση LateUpdate(), και ορίζουμε μια Transform μεταβλητή **introPos2** για να θέσουμε το αντικείμενο αυτό. Μόλις ο χρήστης επιλέξει τρόπο χειρισμού της κίνησης από την αρχική καρτέλα, υπολογίζουμε το χρόνο και μετά από κάποια δευτερόλεπτα θέτουμε στη θέση της κάμερας **Transform.position** τη θέση του στόχου η οποία

αλλάζει ομαλά με την πάροδο του χρόνου μέσω της **Vector3.Lerp**, και φροντίζουμε ώστε να κατευθύνεται στο στόχο κοιτάζοντας προς αυτόν με την **Transform.forward**.

```
#pragma strict
//switching cameras in the scene
public var mainCam : Camera;
public var introCam : Camera;
var count : int = 0;                                    //count the time before change position

function Start()                                                    //at the beginning of the game we can
see the player from the front
{
        introCam.camera.enabled = true;
        mainCam.camera.enabled = false;
        count = 0;
}

function Update ()
{
        if(RunMoveMobile.changeCam)         //after a while we can watch the player from behind
        {
                introCam.camera.enabled = false;
                mainCam.camera.enabled = true;
                gameObject.Find("Directional light").transform.eulerAngles.y = 245;
        }
}

public var smooth : float = 3f;             // a public variable to adjust smoothing of camera motion
var introPos2 : Transform;       // the position to move the camera before the player starts moving

function LateUpdate () {
if(Controls.canPlay)
{
        count = count + 1;
        if ( (count>=40) && introPos2 )
        {
                //lerp the camera position to the intro position,and lerp its forward direction to match
                transform.position = Vector3.Lerp(transform.position, introPos2.position,
                                                Time.deltaTime * smooth);
                transform.forward = Vector3.Lerp(transform.forward, introPos2.forward,
                                                Time.deltaTime * smooth);
        }
}
}
```

Υλοποίηση της κύριας κάμερας

Η κύρια κάμερα ακολουθεί ομαλά τον ήρωα από πίσω του καθόλη τη διάρκεια του παιχνιδιού. Η κίνησή της ελέγχεται από τον κώδικα **CameraRun.js** στον οποίο ακολουθήσαμε τον ίδιο τρόπο με τους κόσμους SnowLand και WaterLand (χωρίς την αλλαγή της θέσης της με την επιλογή των κουμπιών που περιείχαν). Επίσης ελέγχουμε στην LateUpdate() αν ο ήρωας έχει πεθάνει, όπου η κύρια κάμερα μετακινείται ποιο κοντά του για να έχουμε καλύτερη εικόνα, σε ένα κενό GameObject που έχουμε τοποθετήσει δίπλα του και έχουμε θέσει σε μια Transform μεταβλητή **diePos** για να το ελέγξουμε. Η μετακίνηση της κάμερας προς αυτό το σημείο γίνεται ομαλά με την πάροδο του χρόνου, χρησιμοποιώντας την **Vector3.Lerp** για να καθορίσουμε τη θέση της **transform.position** και την κατεύθυνση στην οποία κοιτάει **transform.forward**.

5.8 Κοινές υλοποιήσεις με άλλους κόσμους

5.8.1 Υλοποίηση αντικειμένων συλλογής

Τα αντικείμενα συλλογής τα οποία μπορεί ο ήρωας να συλλέξει κατά τη διάρκεια του παιχνιδιού είναι τα νομίσματα, οι καρδούλες, τα ψάρια μπόνους, και τα μαγικά σμαράγδια τα οποία περιστρέφονται στον αέρα γύρω από τον εαυτό τους με την **Transform.Rotate** στην Update() των κλάσεων **CoinControl**, **HeartCtrlRun**, **FishCtrlRun** και **PowerUpRun** αντίστοιχα. Επίσης μαγνητίζονται από τον ήρωα όταν τα πλησιάσει αρκετά, δηλαδή όταν έρθουν σε επαφή με τον Sphere Collider του μαγνήτη κατευθύνονται προς αυτόν με την **Vector3.MoveTowards**. Στη συνάρτηση OnTriggerEnter() ελέγχουμε πότε έρχονται σε επαφή με τον ήρωα ενεργοποιώντας την αντίστοιχη ενέργεια πριν καταστραφούν με την **Destroy()**.

Επίσης, τα ψάρια και οι καρδούλες ανεβοκατεβαίνουν στον αέρα πάνω και κάτω από την αρχική τους θέση ρυθμίζοντας τις γωνίες περιστροφής και τη θέση τους με την **Transform.localPosition** στη συνάρτηση FixedUpdate().

5.8.2 Υλοποίηση των GUI ετικετών

Στο πάνω στο μέρος της όθονης εμφανίζονται τρεις καρδούλες που αντιπροσωπεύουν τις ζωές του ήρωα, και ρυθμίζονται από την κλάση **HealthRun**, η οποία είναι παρόμοια με αυτή των άλλων κόσμων. Οι καρδούλες φαίνονται από τη συνάρτηση OnGUI(), εφόσον η συσκευή είναι στη σωστή θέση, ο χρήστης έχει επιλέξει τρόπο χειρισμού της κίνησης από το αρχικό μενού, δεν έχει χάσει ή κερδίσει και δεν έχει επιλέξει παύση του παιχνιδιού. Επίσης όταν πεθάνει ο ήρωας εμφανίζει ένα μήνυμα ήττας στο κέντρο της οθόνης που αρχικά αναβοσβήνει και μετά μένει σταθερό μέχρι να εμφανιστεί η καρτέλα τέλος παιχνιδιού.

Εκτός από τις καρδούλες εμφανίζεται ο αριθμός των νομισμάτων που συλλέγει, οι εικόνες των ψαριών μπόνους που παίρνουν έντονο χρώμα αν συλλεχθούν, και η απόσταση του ήρωα από το θησαυρό που αναγράφεται δίπλα σε ένα εικονίδιο με το σεντούκι ώστε να ξέρει

ο χρήστης αν πλησιάζει ή απομακρύνεται από την έξοδο. Επίσης αναπαράγει τη μουσική του παιχνιδιού, και χαμηλώνει την ένταση με την **AudioListener.volume** όταν ο ήρωας χάνει ή κερδίζει, ή επιλέξει παύση ή κρατήσει τη συσκευή λάθος, ή δεν έχει επιλέξει ακόμα τρόπο χειρισμού της κίνησης του ήρωα. Το GUITexture ελέγχεται από τη κλάση **ScoreRunMobile** και είναι συνδεδεμένο στην κύρια κάμερα για να ακούγεται η μουσική παντού.

Το κουμπί παύσης είναι ένα GUITexture όπου η κλάση **PauseTouchRun** που το ελέγχει, σταματάει το χρόνο και θέτει τη μεταβλητή **pauseNow** αληθή όταν ο χρήστης πατήσει με το δάχτυλό του **Input.touches** την περιοχή που βρίσκεται η εικόνα του κουμπιού **GUIElement.HitTest**, ώστε να εμφανιστεί η καρτέλα παύσης.

Το μενού παύσης εμφανίζεται εφόσον πατηθεί το κουμπί παύσης ή τοποθετηθούν δύο δάχτηλα ταυτόχρονα στην οθόνη. Περιλαμβάνει δύο κουμπιά **GUI.Buttons** για τη συνέχιση του παιχνιδιού, ένα για την επανάληψη και ένα για την επιστροφή στο μενού των πιστών αυτού του κόσμου, και ένα μήνυμα προς το χρήστη GUI.Label.

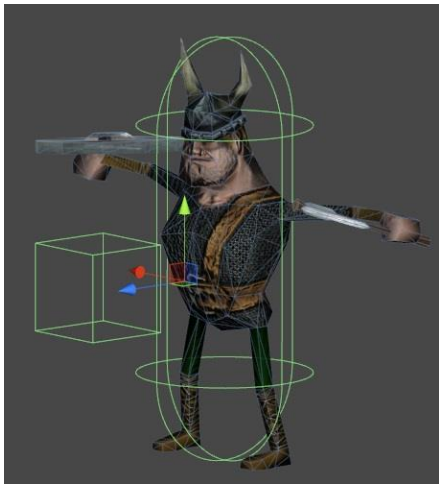
Όταν φτάσει στο θησαυρό, ενεργοποιείται το animation με το οποίο ανοίγει το σεντούκι από την κλάση **TreasureRun**, και εμφανίζεται στο κέντρο της οθόνης ένα μήνυμα νίκης, που αρχικά αναβοσβήνει και μετά παραμένει σταθερό μέχρι να εμφανιστεί η καρτέλα της βαθμολογίας.

Η καρτέλα νίκης ελέγχεται από την κλάση **WinGameRunMobile** η οποία εμφανίζει για κάθε πίστα τη βαθμολογία αναγράφοντας με γαλάζια γράμματα “High Score” αν πέτυχε νέο ρεκόρ, ενώ με κίτρινα γράμματα “Score” αν δεν έκανε νέο ρεκόρ. Από κάτω φαίνονται ο αριθμός των νομισμάτων και εμφανίζονται με έντονο χρώμα τα ψάρια που σύλλεξε. Στο κάτω μέρος υπάρχουν τρία κουμπιά για την επιστροφή στο μενού των πιστών αυτού του κόσμου, για την επανάληψη του παιχνιδού και για την εισαγωγή στην επόμενη πίστα, και από κάτω ένα μήνυμα στο χρήστη.

Η καρτέλα τέλος παιχνιδιού ελέγχεται από την κλάση **GameOverRunMobile** που περιλαμβάνει ένα κουμπί για την επιστροφή στο μενού των πιστών, ένα για την επανάληψη της πίστας, και ένα για την εισαγωγή στην αμέσως επόμενη πίστα μόνο εφόσον η παρούσα έχει τερματιστεί επιτυχώς μια προηγούμενη φορά, και ένα μήνυμα προς το χρήστη. Για να ελέχουμε την πίστα στην οποία βρισκόμαστε θέτουμε στη μεταβλητή **SceneNumber** τον αριθμός της σκηνής στο παιχνίδι, και στην **runland** τον αριθμό της πίστας αυτού του κόσμου.

Σε δυσκολότερες πίστες εκτός από τα εμπόδια κατά μήκος της πίστας, βρίσκονται σε ορισμένα σημεία βίκινγκς ακίνητοι που περιμένουν τον ήρωα να φτάσει κοντά τους για να του επιτεθούν μέσω της κλάσης **VikingStanding**. Ο τρόπος υλοποίησης της συμπεριφοράς τους είναι παρόμοιος με αυτόν στη SnowLand και WaterLand. Στην Update() υπολογίζεται μέσω της Vector3.Distance η απόσταση μεταξύ του βίκινγκ και του ήρωα. Αν δεν τον έχει πιάσει ο

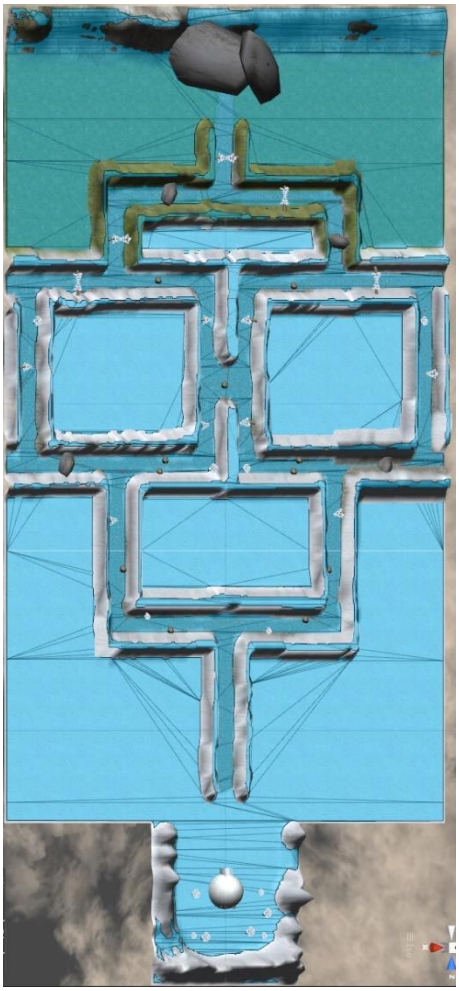
κεντρικός βίκινγκ και ο ήρωας δεν έχει φτάσει στο θησαυρό, ο βίκινγκ είναι σε κατάσταση Idle. Αν πλησιάσει σε μια συγκεκριμένη απόσταση κοιτάει προς το μέρος του με την **Transform.LookAt** και ετοιμάζεται να επιτεθεί (**Shuffle**). Αν πλησιάσει ακόμα πιο κοντά, του επιτίθεται τρέχοντας (**Run**) προς το μέρος του μέσω της **Transform.Translate**. Αν τον ακουμπήσει από μπροστά, ο πιγκουίνος πεθαίνει, ενώ αν πηδήξει από πάνω του και ακουμπήσει στο κεφάλι του δεν πεθαίνει. Την επαφή την ελέγχουμε με την OnTriggerEnter(). Για να καθορίσουμε τον θάνατο του ήρωα μόνο με την μπροστινή επαφή, τοποθετούμε ένα Box Collider μπροστά από αυτόν, από τον οποίο ελέγχεται η επαφή με τον ήρωα, και όχι από τον Capsule Collider που περιλαμβάνει το σώμα του βίκινγκ, ο οποίος χρησιμοποιείται για την κίνησή του.

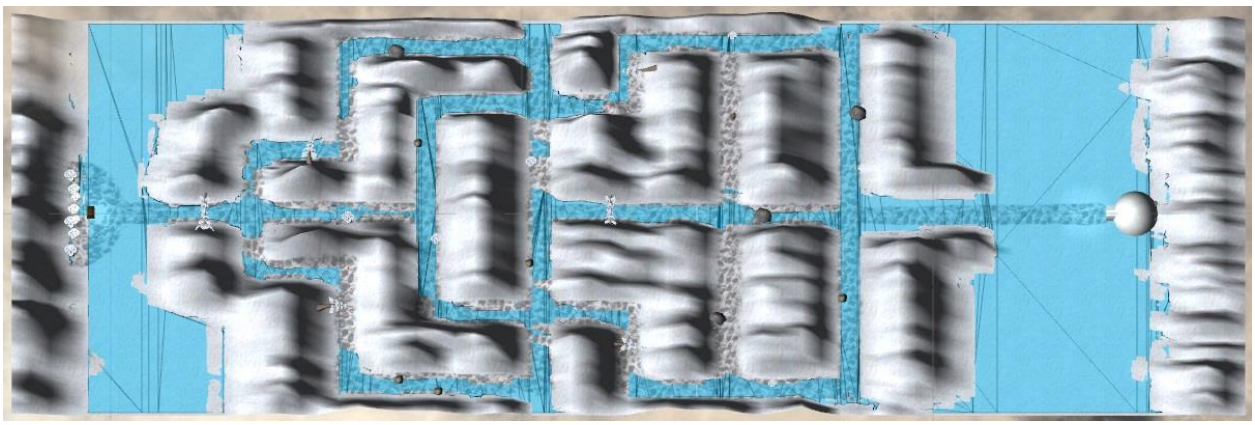


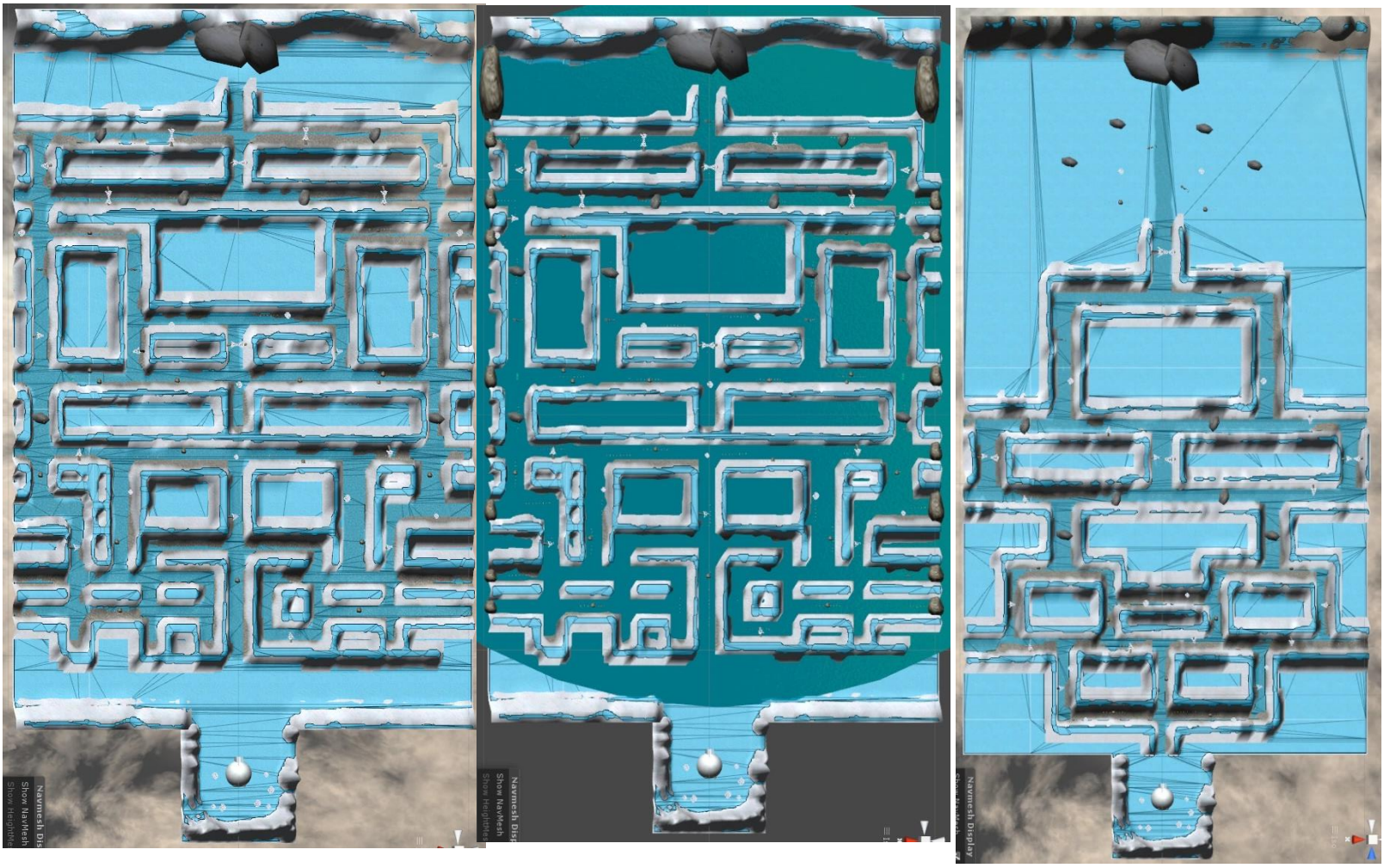
5.9 Υλοποίηση εχθρού βίκινγκ

Ο κεντρικός βίκινγκ βρίσκεται πίσω από τον ήρωα και τον κηνυγάει καθόλη τη διάρκεια του παιχνιδιού. Μέσα στο λαβύρινθο, αν ο ήρωας έχει απομακρυνθεί αρκετά, τον προσεγγίζει ακολουθώντας τη συντομότερη επιτρεπτή διαδρομή. Για να γίνει αυτό, προσθέτουμε στον βίκινγκ τεχνητή νοημοσύνη μέσω της πλοήγησης-Navigation που διαθέτει η Unity, δημιουργώντας πλέγματα πλοήγησης (navigation meshes) για την κάθε πίστα, οι οποίες αναπαριστώνται ως μια γαλάζια επιφάνεια πάνω από το έδαφος.

Οι περισσότερες πίστες αποτελούνται από δύο εδάφη, ένα μικρό terrain στο οποίο βρίσκεται το ιγκλού και ένα μεγάλο που περιλαμβάνει το λαβύρινθο. Η διαδικασία Baking για τη δημιουργία πλεγμάτων, γίνεται επιλέγοντας και τα δύο terrains μαζί, ώστε να δημιουργηθεί ένα ενιαίο πλέγμα πλοήγησης και να μπορεί ο ήρωας να μεταβεί από το ένα στο άλλο. Επίσης για να μπορεί να πλοηγήθεί από αυτά τα πλέγματα, του συνδέουμε το συστατικό NavMeshAgent γύρω από το σώμα του ακουμπώντας το έδαφος.







Ο βίκινγκ αποτελείται από το συστατικό Transform για τον καθορισμός της θέσης περιστροφής και κλίμακάς του, το Animation με το οποίο αναπαράγονται τα animation clip της συμπεριφοράς του, το Sphere Collider με τον οποίο αλληλεπιδρά με το περιβάλλον ως collider, το script **VikingRunMobile.js** που ελέγχει τις λειτουργίες του, το Capsule Collider με το οποίο αντιδρά με τα αντικείμενα στη πίστα και λειτουργεί ως trigger, το Rigidbody για να λειτουργεί υπό την επίδραση της φυσικής, και το NavMeshAgent για να διαβάζει τα πλέγματα πλοήγησης και να μπορεί να αξιοποιήσει τη τεχνητή νοημοσύνη διαλέγοντας τη συντομότερη διαδρομή.

Ανεξάρτητα από το σημείο που βρίσκεται ο βίκινγκ, όποτε ο ήρωας πέφτει πάνω σε ένα εμπόδιο, τον πλησιάζει αρκετά και εμφανίζεται στο πλάνο της κάμερας. Με κάθε ζωή που χάνει πλησιάζει όλο και πιο κοντά και φαίνεται πιο πολύ, και την τρίτη φορά που χάνει όλες τις ζωές του τον πιάνει. Ο ήρωας πρέπει να ξεκινήσει αμέσως να κινείται όταν χάσει μια ζωή, γιατί αν καθυστερήσει ο βίκινγκ που τον έχει πλησιάσει θα τον πιάσει, αλλιώς θα μείνει πάλι αρκετά πίσω καθώς τρέχει πιο αργά από τον ήρωα ώστε να έχει ο χρήστης προβάδισμα. Για να

καθορίσουμε τα τρία σημεία στα οποία μετατοπίζεται ο βίκινγκ γρήγορα με το χάσιμο μιας ζωής, δημιουργούμε δύο κενά GameObject το ένα πίσω από το άλλο τα οποία βρίσκονται λίγο πίσω από τον ήρωα και τον ακολουθούν παντού, και τα χρησιμοποιούμε για να ορίσουμε τις θέσεις στις οποίες θα εμφανιστεί ο βίκινκγ. Στην Transform μεταβλητή **targetCatch1** θέτουμε το πρώτο σημείο στο οποίο θα εμφανιστεί ο βίκινγκ, στην **targetCatch2** θέτουμε το δεύτερο σημείο στο οποίο εμφανίζεται, και στην **targetCatch3** θέτουμε τον ήρωα, καθώς τον πιάνει την τρίτη φορά.

Στην Update() ενεργοποιούμε τη συμπεριφορά του βίκινγκ μόνο εφόσον ο χρήστης επιλέξει τρόπο χειρισμού κίνησης από το αντίστοιχο μενού και η canPlay γίνει αληθής. Υπολογίζεται κάθε καρέ η απόστασή του από τον ήρωα μέσω της **Vector3.Distance**. Αν ο βίκινγκ δεν έχει πιάσει τον πιγκουίνο, ή ο πιγκουίνος δεν έχει φτάσει στο θησαυρό, ο βίκινγκ είναι στραμμένος ως προς τους άξονες x και z προς το μέρος του ήρωα μέσω της **Transform.LookAt**, διατηρώντας τον δικό του άξονα y. Αποκτάμε πρόσβαση στο συστατικό NavMeshAgent και τον κατευθύνουμε προς το στόχο μας με την **NavMeshAgent.destination**, θέτοντάς της τη θέση του ήρωα, και ενεργοποιούμε το animation clip Run για να τρέχει.

Αν ο ήρωας χάσει μια ζωή πέφτοντας πρώτη φορά σε ένα εμπόδιο, με την countHits να είναι ίση με ένα, και την runViking αληθή που σημαίνει ότι πρέπει να μετακινηθεί προς αυτόν ο βίκινγκ, αλλάζουμε τη θέση του βίκινγκ με την **Transform.position** και την κάνουμε ίση με τη θέση στην οποία βρίσκεται την ώρα της σύγκρουσης το πρώτο σημείο στόχου, εισάγοντας στους άξονες x και z τις τιμές του targetCatch1, διατηρώντας την θέση του στον άξονα y. Η μεταφορά του μέχρι εκεί γίνεται μέσω πλοήγηση θέτοντας ως προορισμό το targetCatch1. Αν πλησιάσει αρκετά κοντά σταματάει να τρέχει και απενεργοποιούμε το animation Run.

Αν πέσει πάνω σε κάποιο εμπόδιο δεύτερη φορά, θέτουμε τη θέση του βίκινγκ στους άξονες x και y ίση με τη θέση στους αντίστοιχους άξονες του δεύτερου σημείου στόχου targetCatch2, και τον μεταφέρουμε μέχρι εκεί θέτοντας το σημείο αυτό ως προορισμό του πράκτορα. Αν πλησιάσει αρκετά τον ήρωα σταματάει το **Run** animation.

Αν πέσει πάνω σε εμπόδιο για τρίτη και τελευταία φορά τον μεταφέρουμε μέσω πλοήγησης στον κύριο στόχο, τον ήρωα για να τον πιάσει. Αν τον πλησιάσει αρκετά σταματάει να τρέχει.

Αν τον πιάσει δηλαδή η isHit είναι αληθή, ο βίκινκγ γυρίζει στην κατάσταση Idle. Αν ο πιγκουίνος φτάσει στο θησαυρό δηλαδή η winGUI1 είναι αληθής, σταματάμε τα animation του βίκινγκ και σταματάει να προχωράει. Και στις δύο περιπτώσεις για να σταματήσει να προχωράει θέτουμε ως στόχο τον εαυτό του στην πλοήγηση του NavMeshAgent.

Στη συνάρτηση OnTriggerEnter() ελέγχουμε πότε έρχεται σε επαφή με τον πιγκουίνο ο Capsule Collider. Αν έρθει σε επαφή με τον ήρωα, πηδάει για να τον πιάσει ενεργοποιώντας το animation **Land**, το οποίο αναπαράγεται μία μόνο φορά μέχρι να τελειώσει ο χρόνος που διαρκεί μέσω **WrapMode.Once**.

Η τεχνητή νοημοσύνη που διαθέτει ο βίκινγκ περιορίζεται μόνο στην εύρεση της συντομότερης διαδρομής προς το στόχο και την κατεύθυνσή του προς αυτόν. Αν συναντήσει εμπόδια μπροστά του, τα οποία δεν είχαν συμπεριληφθεί κατά το baking του εδάφους για τη δημιουργία του πλέγματος πλοήγησης και δεν διαθέτουν NavMesh Obstacle, θα κολλήσει σε εκείνο το σημείο γιατί δεν ξέρει πως να τα αποφύγει. Για να μπορέσει να κυνηγήσει τον πιγκουίνο που αποφεύγει τα εμπόδια, όταν ο βίκινγκ πέσει πάνω σε κάποιο από αυτά, τα καταστρέφουμε με την Destroy(), για να μπορέσει να συνεχίσει να ακολουθεί το μονοπάτι της συντομότερης διαδρομής χωρίς επιπλοκές. Αν ο ήρωας ξαναπεράσει από εκεί δεν θα τα συναντήσει ξανά στο δρόμο του.

```
#pragma strict
/*
	Script that makes the viking chases the player through JumpLand
*/
var target : Transform;
var moveSpeed : float = 3.0;
var rotationSpeed = 3;
private var myTransform : Transform;
var runLand : int = 4;

function Awake()	//when the game starts
{
	myTransform = transform;	//the enemy's transform
}

var lookAtDistance : float = 15.0;
var attackRange = 10.0;
var targetCatch1 : Transform;
var targetCatch2 : Transform;
var targetCatch3 : Transform;
var smoothTime = 0.3;
private var yVelocity = 0.0;
```

```
function Update ()
{
if(Controls.canPlay)
{	//the distance between the viking and the player
	var distance = Vector3.Distance(target.position, myTransform.position);
	if(!RunMoveMobile.isHit && !TreasureRun.winGUI1)
	{	//if the viking catches the player he stops attacking
		myTransform.LookAt(Vector3(target.position.x, myTransform.position.y,
				target.position.z));//Rotates the transform so the forward
					//vector points at target's current position.
		GetComponent(NavMeshAgent).destination = target.position;
		animation.CrossFade("Run");
	}
	if(RunMoveMobile.runViking && RunMoveMobile.countHits == 1)
	{
		transform.position = Vector3(targetCatch1.position.x, transform.position.y,
					targetCatch1.position.z);
		GetComponent(NavMeshAgent).destination = targetCatch1.position;
		if( distance<=5 )
		{
			RunMoveMobile.runViking = false;
		}
	}
	if(RunMoveMobile.runViking && RunMoveMobile.countHits == 2)
	{
		transform.position = Vector3(targetCatch2.position.x, transform.position.y,
					targetCatch2.position.z);
		GetComponent(NavMeshAgent).destination = targetCatch2.position;
		if( distance<=7 )
		{
			RunMoveMobile.runViking = false;
		}
	}
	if(RunMoveMobile.runViking && RunMoveMobile.countHits == 3)
	{
		GetComponent(NavMeshAgent).destination = target.position;
		if( distance<=7 )
		{
			RunMoveMobile.runViking = false;
		}
	}
	if(RunMoveMobile.isHit )
	{
```

```
                animation.CrossFade("Idle");
                GetComponent(NavMeshAgent).destination = transform.position;
        }
        if(TreasureRun.winGUI1)
        {
                animation.Stop();       //if the penguin won the viking stops chasing him
                GetComponent(NavMeshAgent).destination = transform.position;
        }
}

function OnTriggerEnter( hit : Collider )
{
        if(hit.gameObject.tag == "Player")
        {
                animation["Land"].wrapMode = WrapMode.Once;
                animation.Play("Land");
        }
        //GOT HIT
        if(hit.gameObject.tag == "Obstacle" )                   //when he hits an obstacle
        {
                Destroy(hit.gameObject);                        //we destroy it so he can pass it
        }
}
```

ΚΕΦΑΛΑΙΟ 6

ΑΞΙΟΛΟΓΗΣΗ ΣΥΣΤΗΜΑΤΟΣ

6.1 Μέθοδος αξιολόγησης

Το παιχνίδι δοκιμάστηκε από δέκα χρήστες και των δύο φύλλων, ηλικίας από 11 έως 52 χρονών, από διαφορετικούς επαγγελματικούς χώρους. Όλοι ήταν κάτοχοι συσκευών αφής smartphone ή και tablet αλλά είχαν διαφορετική εξοικείωση με τις εφαρμογές παιχνιδιών.

Για την αξιολόγηση του παιχνιδιού έγινε χρήση ερωτηματολογίων χρηστικότητας QUIS (Questionnaire for User Interaction Satisfaction). Τα ερωτηματολόγια περιείχαν ερωτήσεις σχετικές με την εμφάνιση, τη λειτουργικότητα, την ευκολία εκμάθησης, τις γενικές δυνατότητες της εφαρμογής, τα πολυμέσα και την εγκατάσταση του προγράμματος.

Το ερωτηματολόγιο QUIS περιλαμβάνει 39 ερωτήσεις, χωρισμένες σε πέντε ενότητες, και οι απαντήσεις δίνονται σε κλίμακα από ένα (1) έως εννιά (9) ή δεν ξέρω δεν απαντώ (NA). Ακολουθούν τα ερωτηματολόγια που χρησιμοποιήθηκαν.

Ερωτηματολόγιο Αξιολόγησης Διαλογικής Χρήσης Συστήματος

Φύλο: Ηλικία: _____

☐ Αρσενικό
☐ Θυληκό

Μέρος 1: Γενική εντύπωση του χρήστη

Παρακαλώ κυκλώστε τους αριθμούς που αντιπροσωπεύουν τις εντυπώσεις σας από τη χρήση αυτής της εφαρμογής παιχνιδιού. Δεν ξέρω/ δεν απαντώ = NA.

1.1 Γενική αντίδραση της εφαρμογής: Απαράδεκτη Υπέροχη
1 2 3 4 5 6 7 8 9 NA

1.2 Γενική αντίδραση της εφαρμογής: Μπερδεύει Ικανοποιεί
1 2 3 4 5 6 7 8 9 NA

1.3 Γενική αντίδραση της εφαρμογής: Αδιάφορο Ευχάριστο
1 2 3 4 5 6 7 8 9 NA

1.4 Γενική αντίδραση της εφαρμογής: Δύσκολο Εύκολο
1 2 3 4 5 6 7 8 9 NA

1.5 Γενική αντίδραση της εφαρμογής: Άκαμπτο Ευέλικτο
1 2 3 4 5 6 7 8 9 NA

Μέρος 2: Οθόνη

2.1 Ο σχεδιασμός της οθόνης βοήθησε: Ποτέ Πάντα
1 2 3 4 5 6 7 8 9 NA

2.2 Ποσότητα πληροφορίας που εμφανιζόταν στη οθόνη: Ανεπαρκής Επαρκής
1 2 3 4 5 6 7 8 9 NA

2.3 Δόμηση της πληροφορίας που εμφανιζόταν στην οθόνη: Χαοτική Οργανωμένη
1 2 3 4 5 6 7 8 9 NA

2.4 Αλληλουχία οθονών: Μπερδεμένη Σαφής
1 2 3 4 5 6 7 8 9 NA

2.5 Επόμενη οθόνη στη σειρά: Απρόβλεπτη Προβλέψιμη
1 2 3 4 5 6 7 8 9 NA

2.6 Επιστροφή στην προηγούμενη οθόνη: Αδύνατη Εύκολη
1 2 3 4 5 6 7 8 9 NA

Σχόλια για τη δομή των οθονών:

_____
_____
_____
_____
_____

Μέρος 3: Ορολογία και επικοινωνία με την εφαρμογή

3.1 Τα μηνύματα εμφανίζονται στην οθόνη με: Ασυνέπεια Συνέπεια
1 2 3 4 5 6 7 8 9 NA

3.2 Τα μηνύματα που εμφανίζονται στην οθόνη είναι: Μπερδεμένα Σαφή
1 2 3 4 5 6 7 8 9 NA

3.3 Η εφαρμογή σας ενημερώνει για το τι κάνει: Ποτέ Πάντα
1 2 3 4 5 6 7 8 9 NA

3.4 Η εκτέλεση μιας κίνησης οδηγεί σε προβλέψιμο αποτέλεσμα:
Ποτέ Πάντα
1 2 3 4 5 6 7 8 9 NA

3.5 Ανάδρασης της εφαρμογής: Αδύνατη Εύκολη
1 2 3 4 5 6 7 8 9 NA

3.6 Καθυστερήσεις: Απαράδεκτες Ανεκτές
1 2 3 4 5 6 7 8 9 NA

3.7 Μηνύματα λαθών: Άχρηστα Χρήσιμα
1 2 3 4 5 6 7 8 9 NA

Σχόλια για την ορολογία και την επικοινωνία με την εφαρμογή:

_____
_____
_____
_____
_____

Μέρος 4: Εκμάθηση χρήσης

4.1 Εκμάθηση χρήσης της εφαρμογής: Δύσκολη Εύκολη
1 2 3 4 5 6 7 8 9 NA

4.2 Χρόνος εκμάθησης της εφαρμογής: Πολύς Λίγος
1 2 3 4 5 6 7 8 9 NA

4.3 Η εξερεύνηση των δυνατοτήτων με τη μέθοδο «προσπάθειας και λάθους (trial and error)» :
Απογοητεύει Αποδίδει
1 2 3 4 5 6 7 8 9 NA

4.4 Οι εργασίες γίνονται σε λογική αλληλουχία: Ποτέ Πάντα
1 2 3 4 5 6 7 8 9 NA

4.5 Τα βήματα για την ολοκήρωση της εργασίας ακολουθούν μια λογική σειρά:
Ποτέ Πάντα
1 2 3 4 5 6 7 8 9 NA

4.6 Η ανάδραση όταν ολοκληρωθεί η εργασία είναι: Ασαφής Σαφής
1 2 3 4 5 6 7 8 9 NA

Σχόλια για την εκμάθηση της χρήσης:

_____
_____
_____
_____
_____

Μέρος 5: Δυνατότητες της εφαρμογής

5.1 Ταχύτητα της εφαρμογής: Πολύ αργή Ικανοποιητική
1 2 3 4 5 6 7 8 9 NA

5.2 Σταθερότητα εφαρμογής: Ποτέ Πάντα
1 2 3 4 5 6 7 8 9 NA

5.3 Χειρισμοί – Λειτουργίες: Αναξιόπιστα Αξιόπιστα
1 2 3 4 5 6 7 8 9 NA

5.4 Η ευκολία του χειρισμού εξαρτάται από την εμπειρία του χρήστη:
Ποτέ Πάντα
1 2 3 4 5 6 7 8 9 NA

Σχόλια για τις δυνατότητες της εφαρμογής:

_____
_____
_____
_____
_____

Μέρος 6: Πολυμέσα

6.1 Ποιότητα των εικόνων/φωτογραφιών: Κακή Καλή
1 2 3 4 5 6 7 8 9 NA

6.2 Εικόνες/Φωτογραφίες: Θολές Καθαρές
1 2 3 4 5 6 7 8 9 NA

6.3 Φωτεινότητα της εικόνας/φωτογραφίας: Σκοτεινή Φωτεινή
1 2 3 4 5 6 7 8 9 NA

6.4 Ηχητική απόδοση: Ασταθής Εύρυθμη
1 2 3 4 5 6 7 8 9 NA

6.5 Ηχητική απόδοση: Αλλοιωμένη Ξεκάθαρη
1 2 3 4 5 6 7 8 9 NA

6.6 Χρώματα: Αφύσικα Φυσικά
1 2 3 4 5 6 7 8 9 NA

6.7 Διαθέσιμη ποσότητα χρωμάτων Ανεπαρκής Επαρκής
1 2 3 4 5 6 7 8 9 NA

Σχόλια για τα πολυμέσα:

_____
_____
_____

_____
_____

Μέρος 7: Εγκατάσταση εφαρμογής

7.1 Ταχύτητα εγκατάστασης της εφαρμογής: Αργή Γρήγορη
1 2 3 4 5 6 7 8 9 NA

7.2 Εξατομίκευση: Δύκολη Εύκολη
1 2 3 4 5 6 7 8 9 NA

7.3 Πληροφόρηση του χρήστη για την πρόοδό του: Ποτέ Πάντα
1 2 3 4 5 6 7 8 9 NA

7.4 Εποικοδομητικές εξηγήσεις κατά την αποτυχία: Ποτέ Πάντα
1 2 3 4 5 6 7 8 9 NA

Σχόλια για την εγκατάσταση του συστήματος:

_____
_____
_____
_____
_____

Μέρος 8: Γραφικό και σχεδιαστικό μέρος

8.1 Εντύπωση από τον ήρωα: Κακή Άριστη
1 2 3 4 5 6 7 8 9 NA

8.2 Εντύπωση από τον ήρωα: Αδιάφορος Ευχάριστος
1 2 3 4 5 6 7 8 9 NA

8.3 Γραφικός σχεδιαμός: Κακός Ικανοποιητικός
1 2 3 4 5 6 7 8 9 NA

8.4 Κινήσεις του ήρωα στο παιχνίδι: Κακές Ικανοποιητικές
1 2 3 4 5 6 7 8 9 NA

8.5 Κινήσεις του ήρωα στο παιχνίδι: Πρόχειρες Ποιοτικές
1 2 3 4 5 6 7 8 9 NA

8.6 Κινήσεις του ήρωα στο παιχνίδι: Λίγες Αρκετές
1 2 3 4 5 6 7 8 9 NA

8.7 Ικανοποίηση από τις πίστες: Αδιάφορες Ευχάριστες
1 2 3 4 5 6 7 8 9 NA

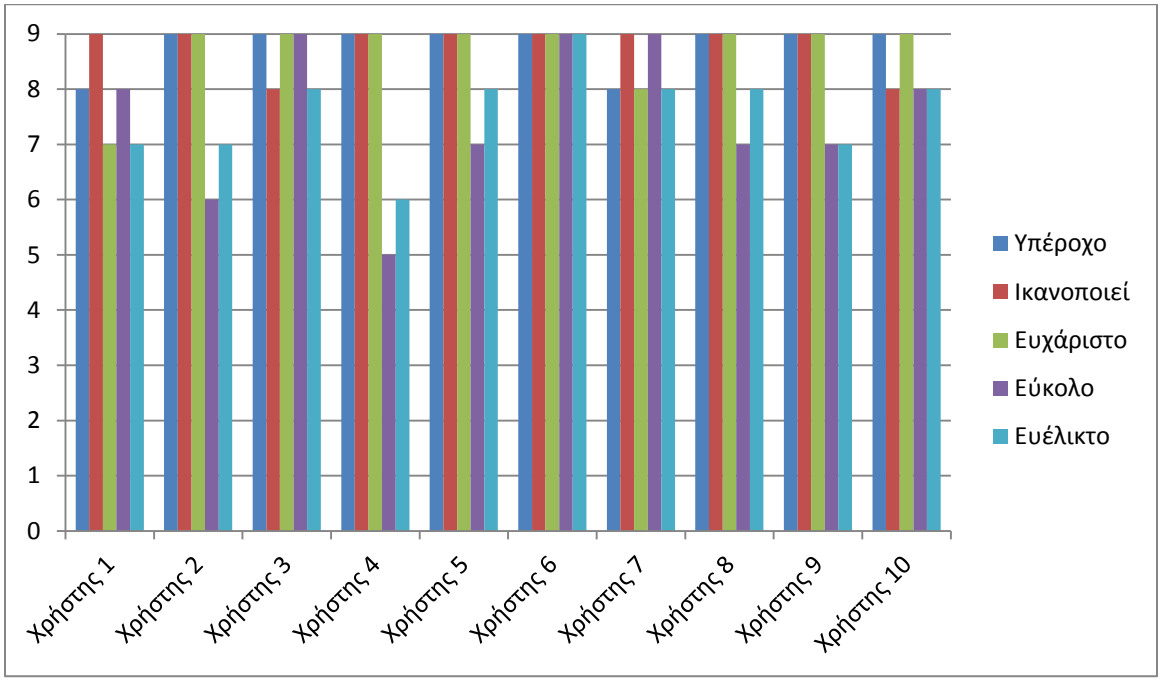
8.8 Ικανοποίηση από τους κόσμους: Αδιάφοροι Ευχάριστοι
1 2 3 4 5 6 7 8 9 NA

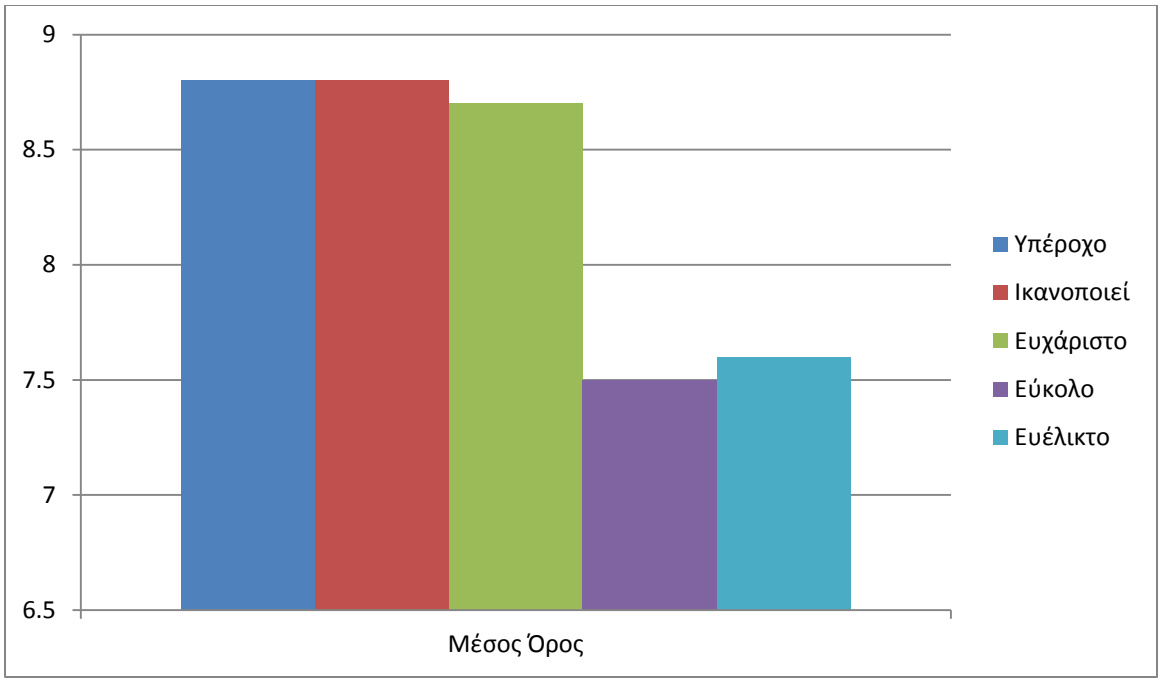
Σχόλια για το γραφικό σχεδιασμό του παιχνιδιού:
_____
_____
_____
_____
_____

6.2 Αποτελέσματα και Συμπεράσματα

Παρακάτω παρουσιάζουμε σε σχεδιαγράμματα τις απαντήσεις των χρηστών για κάθε ερώτηση, ανά μέρος του ερωτηματολογίου. Μέσα από τις απαντήσεις των ερωτήσεων αυτών, μπορούμε να δούμε τη χρηστικότητα της εφαρμογής και τη φιλικότητά της προς τους χρήστες, ώστε να μπορέσουμε να την βελτιώσουμε ή και να την τροποιήσουμε.

Μέρος 1 : Γενική εντύπωση των χρηστών





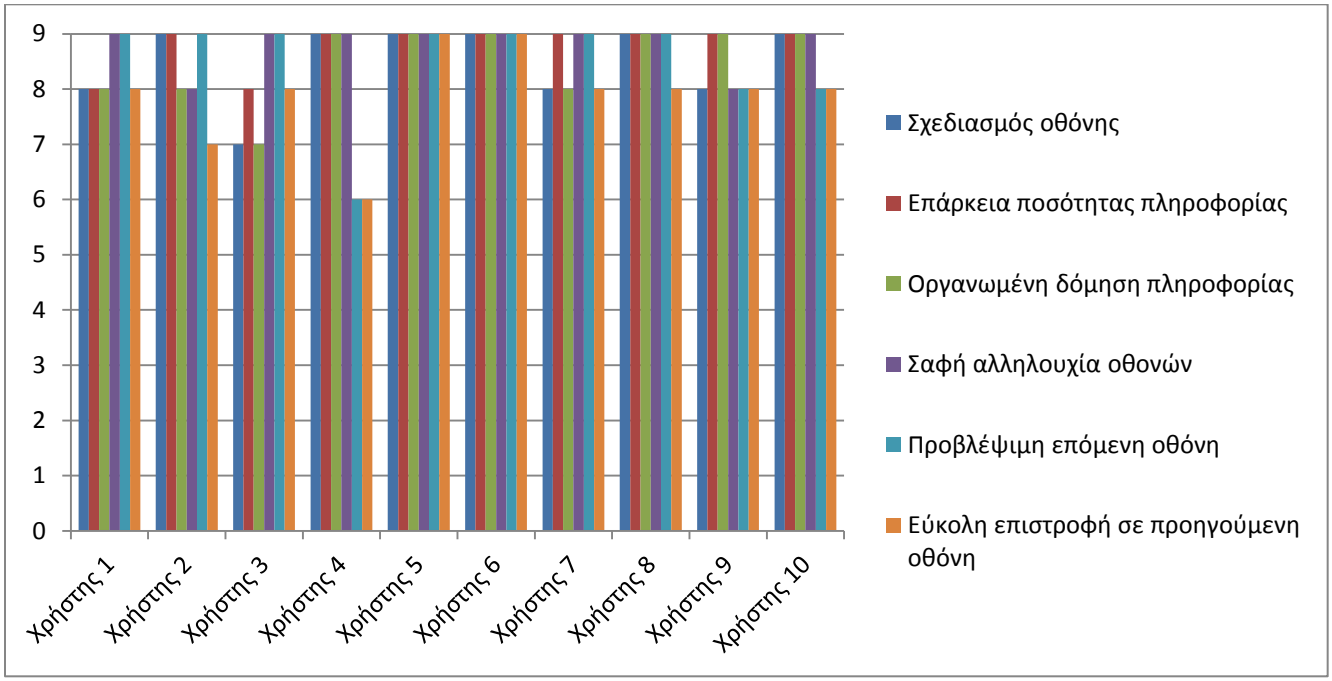
Σχόλια:

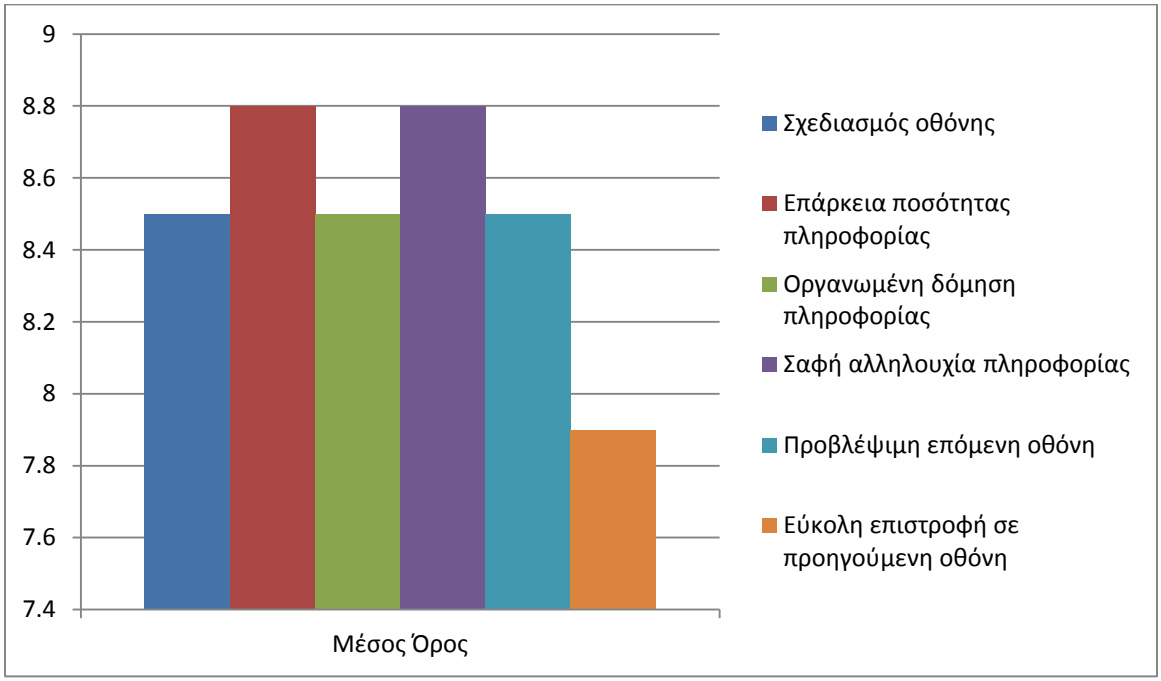
Χρήστης 4: «Καταπληκτικό παιχνιδι!!!! Πολύ καλή ιδέα.»

Συμπεράσματα:

Στους χρήστες έκανε ιδιαίτερη, και κυρίως ευχάριστη, εντύπωση το παιχνίδι μας. Το παιχνίδι είναι μια ολοκληρωμένη εφαρμογή, για αυτό μπόρεσε να εφαρμοστεί στις διάφορες συσκευές των χρηστών. Εντυπωσιάστηκαν από τα γραφικά, τη μουσική, το θέμα, και την εμφάνισή του στην οθόνη της κινητής συσκευής τους. Ανεξάρτητα από την εμπειρία του κάθε χρήστη στα παιχνίδια, δεν δυσκολεύτηκαν στο χειρισμό, καθώς ήξεραν αμέσως πως να αντιδράσουν. Περιηγήθηκαν μέσα στις πίστες του παιχνιδιού εξερευνώντας τες με μεγάλη ευκολία, δοκιμάζοντας τις δυνατότητες του χρήστη. Γενικά έμειναν ικανοποιημένοι από την εφαρμογή μας, όπως μπορούμε να δούμε και από τις απαντήσεις τους.

Μέρος 2: Οθόνη





Σχόλια:

Χρήστης 2: «Γενικά ικανοποιητική.»

Χρήστης 3: «Η εφαρμογή κερδίζει αρκετά από το σχεδιασμό της οθόνης, η οποία την αναδεικνύει καλύτερα. Επί προσθέτως, η ποσότητα των πληροφοριών ειναι απόλυτα επαρκής

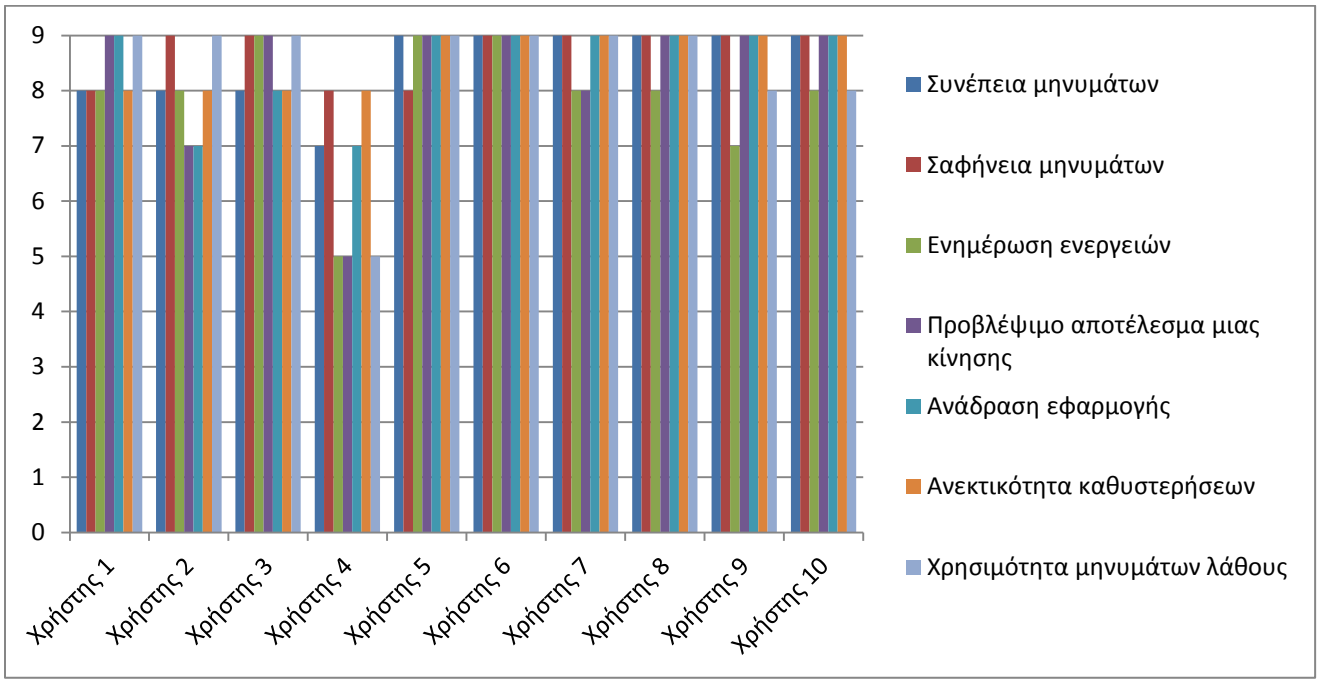
σε αντίθεση με τη δόμηση τους που πιστεύω πως υστερεί. Ενώ, η αλληλουχία των οθονών ειναι σαφής και η επόμενη οθόνη σε σειρά προβλέψιμη. Όσο για την επιστροφή στην προηγούμενη οθόνη πολύ εύκολη.»

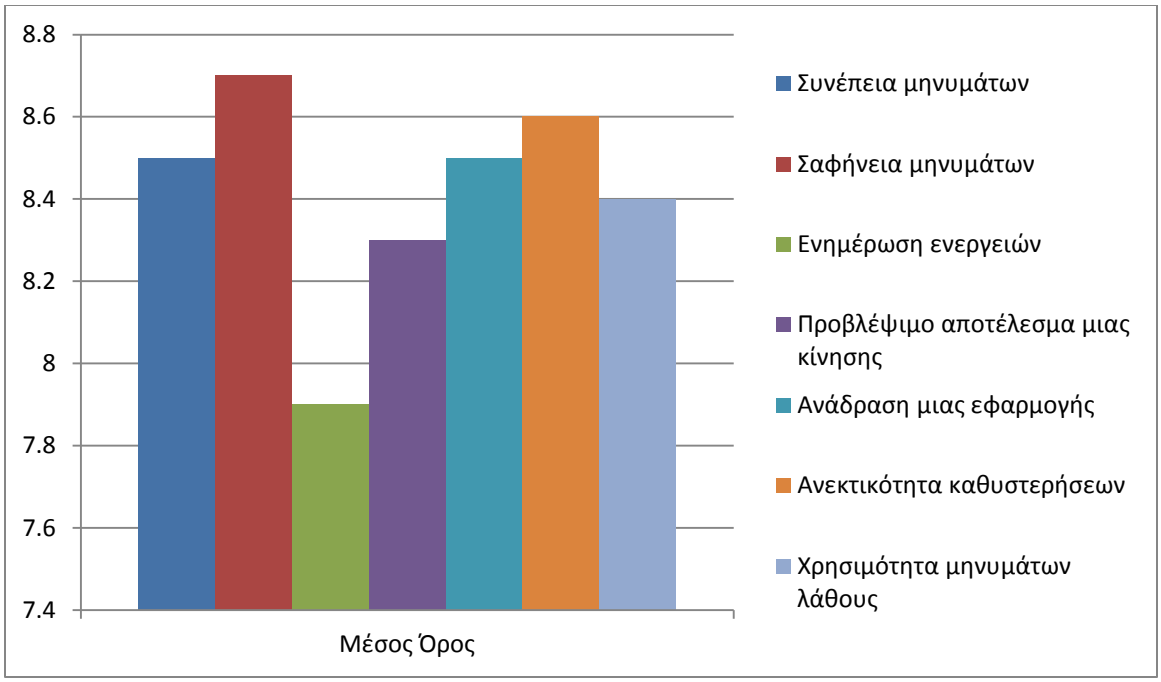
Χρήστης 7: «Είναι αρκετά ευχάριστο.»

Συμπεράσματα:

Οι χρήστες ήταν αρκετά ικανοποιημένοι από το σχεδιασμό της οθόνης στα πλαίσια της συσκευής τους. Η ποσότητα της διαθέσιμης πληροφορίας ήταν αρκετή ώστε να καταλαβαίνουν τι πρέπει να κάνουν και τι συμβαίνει στο παιχνίδι την κάθε στιγμή. Η δόμησή της ήταν κατάλληλη στην οθόνη της συσκευής ώστε να φαίνεται εύκολα στο μάτι των χρηστών και να μην περιορίζει την ορατότητά τους στην πίστα του παιχνιδιού. Δεν δυσκολεύτηκαν καθόλου στο να κατανοήσουν την αλληλουχία των οθονών, καθώς είναι αρκετά απλή και ακολουθεί γνωστά πρότυπα παιχνιδιών. Μέχρι και οι πιο άπειροι χρήστες καταλάβαιναν που θα τους οδηγούσε η κάθε κίνησή τους, και μπορούσαν να προβλέψουν την επόμενη οθόνη που θα εμφανιστεί. Σε γενικές γραμμές, θεώρησαν εύκολη τη μετάβασή τους σε προηγούμενη οθόνη, καθώς η επιλογή αυτή βρισκόταν σε κάθε πίστα του παιχνιδιού με απλά και αντιπροσωπευτηκά κουμπιά, ίσως επίσης γνώριμα από τους χρήστες από άλλα παιχνίδια.

Μέρος 3: Ορολογία και επικοινωνία με την εφαρμογή





Σχόλια:
Χρήστης 1: «Κοινότυπη.»

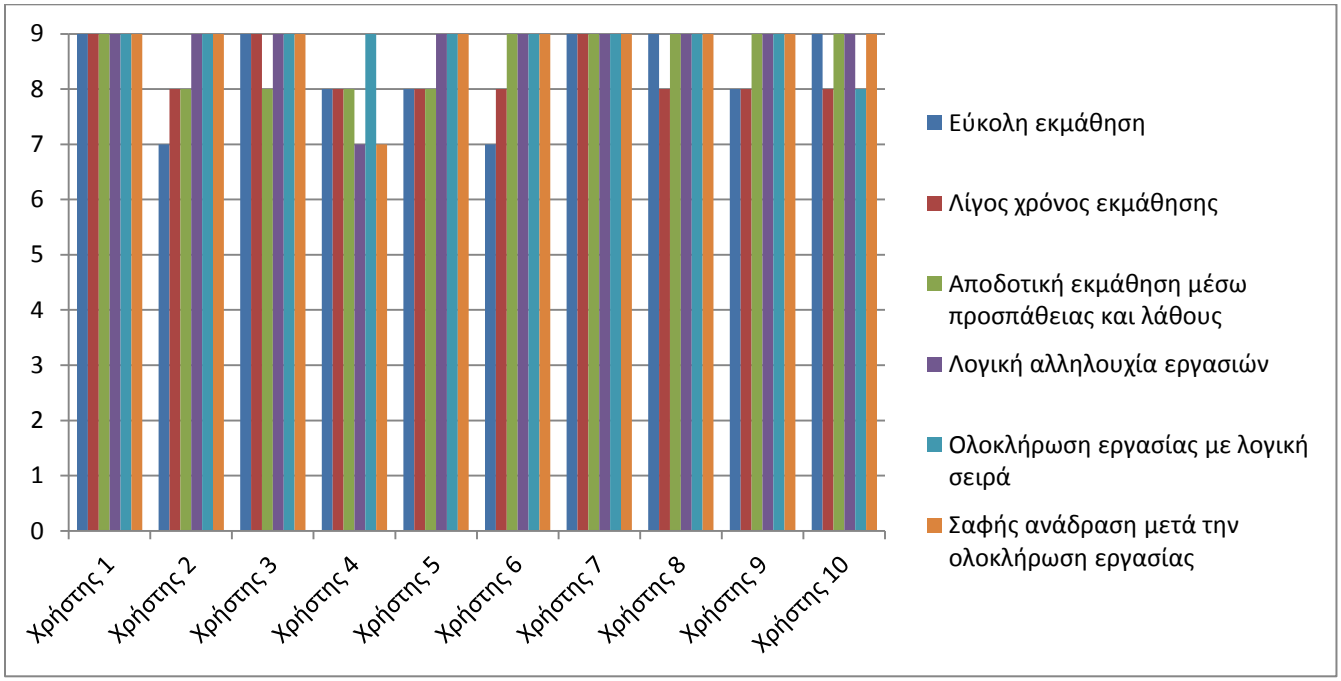
Χρήστης 2: «Δεν εμφανίζονται σοβαρές δυσκολίες.»

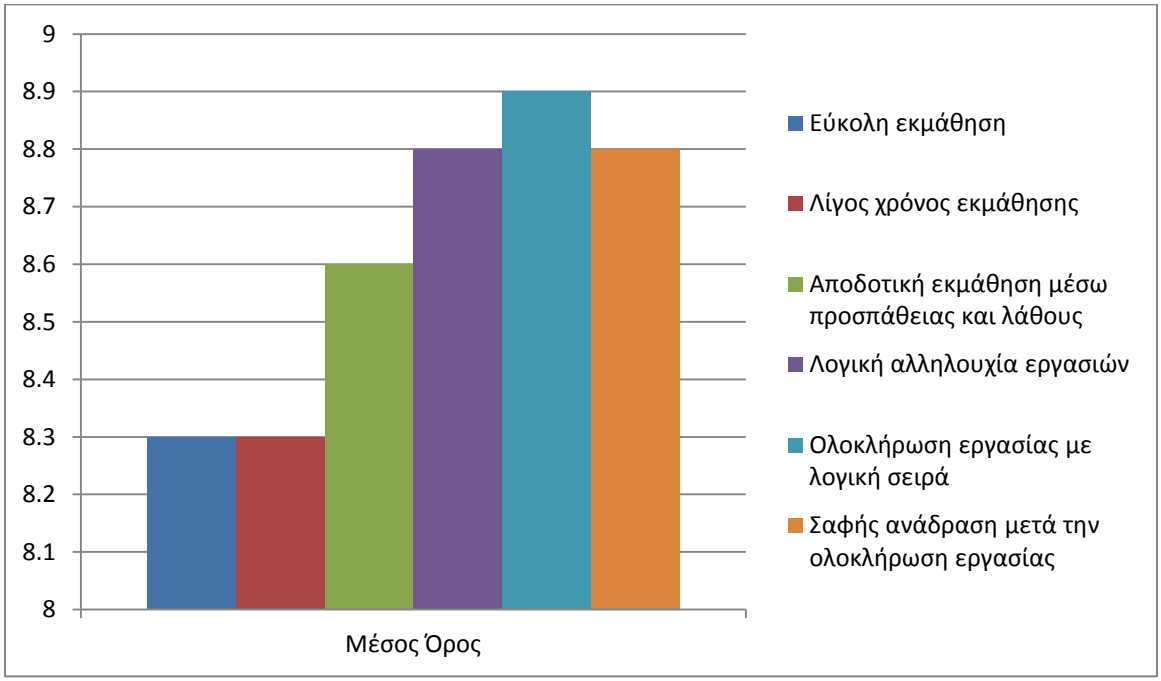
Χρήστης 3: «Η επικοινωνία που έχει ο χρήστης με την εφαρμογή είναι ικανοποιητική, περιορίζεται σε αυτά που χρειάζονται χωρίς να τον κουράζει. Τα μηνύματα μεταδίδονται έγκυρα περιορίζοντας την ενδεχόμενη αναμονή.»

Συμπεράσματα:

Η εμφάνιση των μηνυμάτων στην οθόνη ήταν αρκετή ώστε να ενημερώνονται οι χρήστες σχετικά με τη δράση τους. Κατανοούσαν απόλυτα την ορολογία του παιχνιδιού, καθώς είναι απλή, σαφής και ξεκάθαρη, ακολουθώντας τα πρότυπα των περισσότερων παιχνιδιών. Ήξεραν τι συνέβαινε κατά τη διάρκεια του παιχνιδιού, καθώς τους παρέχεται η απαραίτητη ενημέρωση των ενεργειών τους σε κρίσιμες στιγμές. Επίσης ήξεραν τι να περιμένουν με την κάθε κίνησή τους, καθώς το παιχνίδι είναι απλό και οικείο ώστε να μην μπερδεύονται και να νιώθουν σίγουροι για τις ενέργειές τους. Ήταν αρκετά ικανοποιημένοι από την ανάδραση της εφαρμογής με την κάθε επιλογής τους για επανάληψη της πίστας, και δεν παρατήρησαν ιδιαίτερες καθυστερήσεις. Θεώρησαν επίσης, αρκετά χρήσιμα τα μηνύματα λαθών για τη δράση τους.

Μέρος 4: Εκμάθηση χρήσης





Σχόλια:

Χρήστης 2: «Η ανάδραση είναι ένα σημαντικό μέρος του παιχνιδιού.»

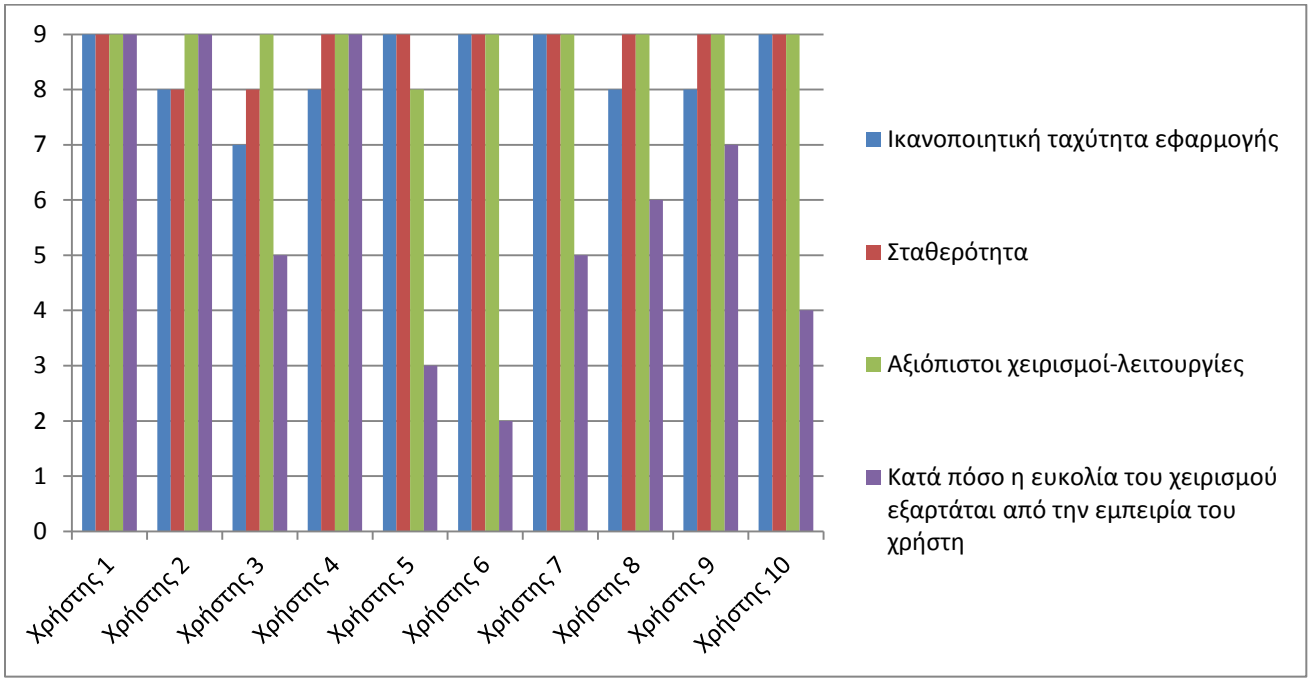
Χρήστης 3: «Η εκμάθηση της χρήσης της εφαρμογής είναι αρκετά εύκολη, ακόμη και στα ενδεχομένως πιο δυσνόητα σημεία λόγω των βοηθητικών μηνυμάτων.»

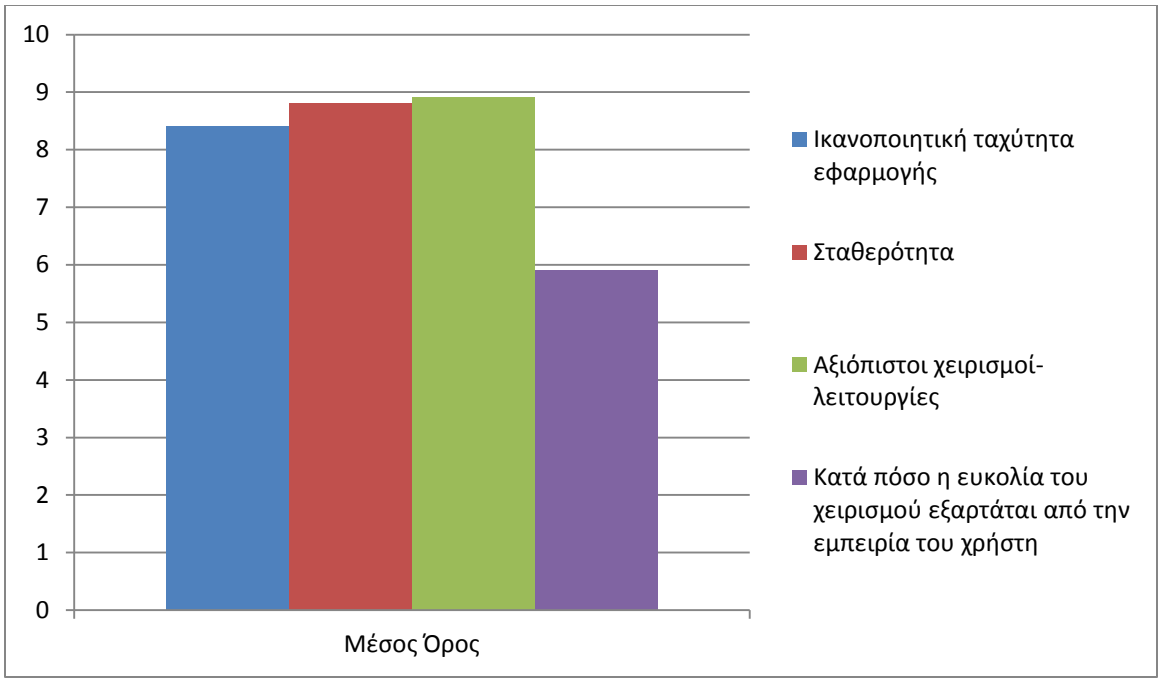
Χρήστης 5: «Πολύ καλή, απλή και γρήγορη εκμάθηση της εφαρμογής.»

Συμπεράσματα:

Η εκμάθηση του παιχνιδιού από τους χρήστες ήταν άμεση και εύκολη. Χρειάστηκαν ελάχιστο χρόνο για να κατανοήσουν τι πρέπει να κάνουν, και ανταποκρίθηκαν άμεσα στο χειρισμό, κατανοώντας τον απόλυτα. Αρκετά αποδοτική ήταν η μέθοδος «προσπάθειας και λάθους», καθώς με μία δοκιμή τους σε κάθε πίστα, καταλάβαιναν πως παίζεται το παιχνίδι. Η η εκμάθηση ήταν εύκολη γιατί οι εργασίες γίνονται με λογική αλληλουχία και σειρά. Επίσης είναι ξεκάθαρη και σαφής η ανάδραση κατά την ολοκλήρωση μιας εργασίας, οπότε οι χρήστες ήξεραν που βρίσκονται και που πάνε την κάθε στιγμή, χωρίς να χάνονται μέσα στο παιχνίδι.

Μέρος 5: Δυνατότητες της εφαρμογής





Σχόλια:

Χρήστης 2: «Απευθύνεται σε έμπειρους χρήστες.»

Χρήστης 3: «Η εφαρμογή έχει την σταθερότητα που επιβάλλεται, οι χειρισμοί και οι λειτουργίες ειναι αξιόπιστες και η ευκολία χειρισμού είναι υψηλή χωρίς να χρειάζεται ιδαίτερη εμπειρία. Η ταχύτητα της εφαρμογής είναι λίγο πιο χαμηλή.»

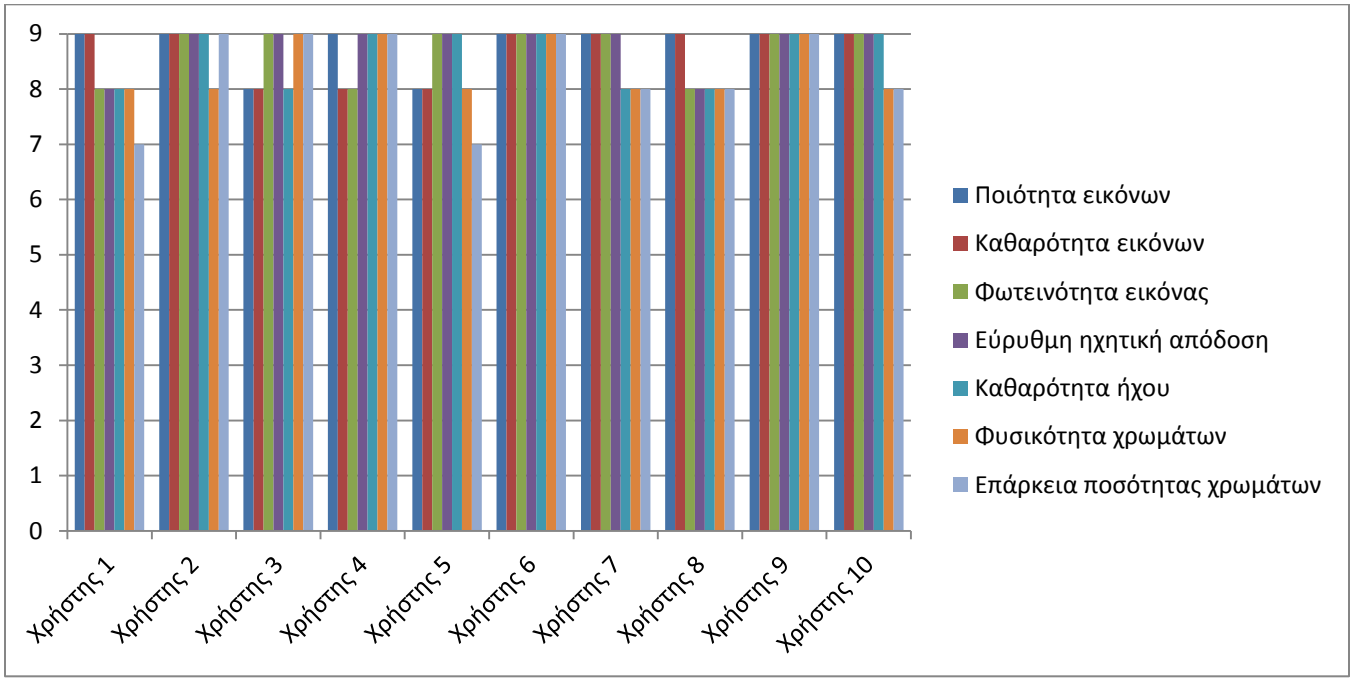
Χρήστης 5: «Θα μπορούσε να είχε καλύτερα, για το κουμπί του χτυπήματος, σύμβολο νιφάδας αντί του κεραυνού. Κατά τα άλλα εξαιρετικός χειρισμός!»

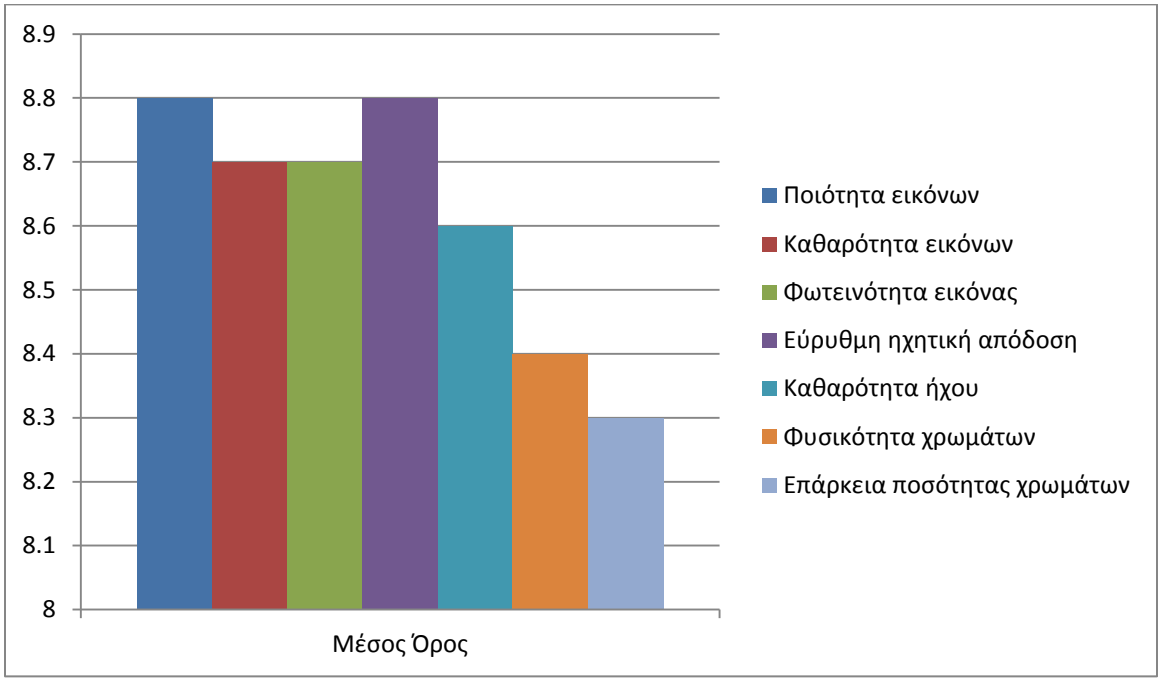
Χρήστης 6: «Δεν λειτουργεί σε μονοπύρηνα κινητά.»

Συμπεράσματα:

Οι χρήστες ήταν αρκετά ικανοποιημένοι από την ταχύτητα και τη σταθερότητα της εφαρμογής. Αρκετά αξιόπιστος είναι ο χειρισμός και οι λειτουργίες του παιχνιδιού, καθώς δεν απογοήτευσαν τους χρήστες και τους επέτρεπαν να παίζουν ευχάριστα το παιχνίδι. Σε όλους τους χρήστες φάνηκε εύκολος ο χειρισμός, καθώς ήταν απλός και περιορισμένος στα πλαίσια μιας κινητής συσκευής. Παρατηρήθηκε έντονη διαφορά στις απαντήσεις των χρηστών για το αν η ευκολία του χειρισμού εξαρτάται από την εμπειρία του χρήστη. Κάποιοι από αυτούς θεώρησαν ότι είναι πολύ απλός και κατανοητός και δεν χρειάζεται να είναι έμπειρος κάποιος για να παίξει, και κάποιοι άλλοι θεώρησαν ότι είναι πιο εύκολο για έμπειρους παίκτες.

Μέρος 6: Πολυμέσα





Σχόλια:

Χρήστης 2: «Τέλεια γραφικά.»

Χρήστης 3: «Η ποιότητα των εικόνων/φωτογραφιών είναι αρκετά καλή και ειδικά η φωτεινότητα είναι ακριβώς στα όρια που πρέπει. Η ηχητική απόδοση είναι εξίσου καλή ως

προς την ποιότητα και λιγότερο καλύτερη είναι η σταθερότητα του ήχου, η οποία χάνει σε κάποια σημεία.»

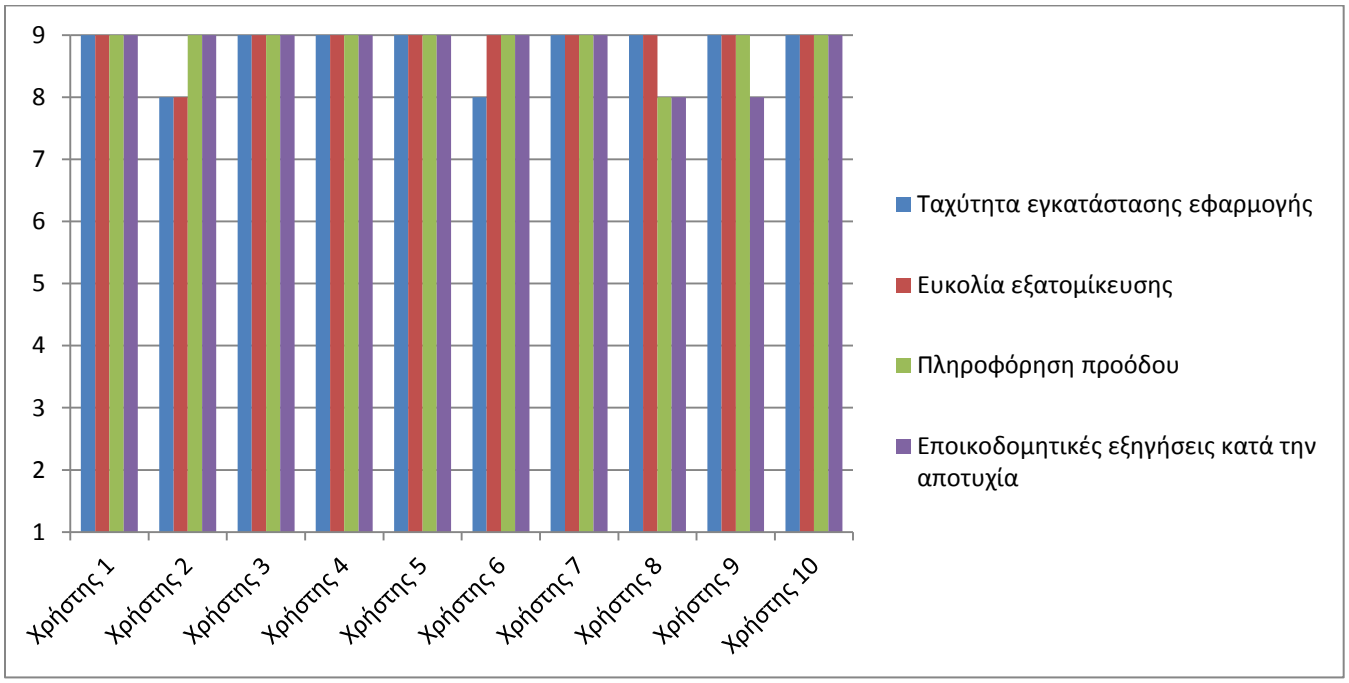
Χρήστης 4: «Πολύ καλή και παιχνιδιάρικη μουσική. Ίσως η μουσική του θησαυρού θα έπρεπε να είναι πιο φαντασμαγορική.»

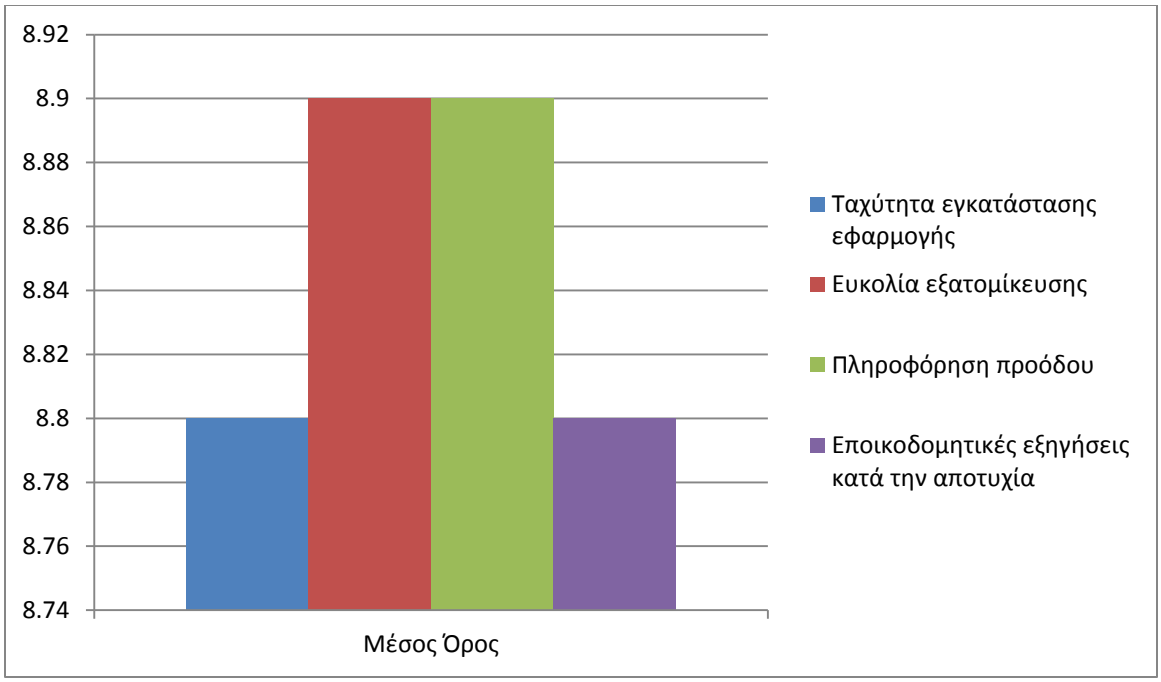
Χρήστης 5: « Πολύ ωραία και λεπτομερής η σχεδίαση του ήρωα.»

Συμπεράσματα:

Οι χρήστες ήταν πολύ ικανοποιημένοι από τα γραφικά στοιχεία του παιχνιδιού, ως προς την ποιότητα, την καθαρότητα και την φωτεινότητα των εικόνων, καθώς χρησιμοποιήσαμε προγράμματα σχεδίασης υψηλής ποιότητας για τη δημιουργία τους. Επίσης ικανοποιημένοι ήταν και από την απόδοση και καθαρότητα του ήχου, τον οποίο επεξεργαστήκαμε κατάλληλα για να μην αλλοιώνεται κατά τη δράση του ήρωα, και κατά την αναπαραγωγή περισσότερων του ενός ήχων. Λίγο λιγότερο ικανοποιημένοι ήταν ως προς τη φυσικότητα και την ποσότητα των χρωμάτων, που είναι λογικό να είναι περιορισμένα καθώς οι πίστες δράσης περιβάλλονται κυρίως από πάγο, μη μπορώντας να προσθέσουμε πολλά στοιχεία στο περιβάλλον αυτό. Επίσης δεν θέλαμε να είναι πολύ φυσικά τα χρώματα καθώς είναι κινούμενα σχέδια και στοχεύαμε σε έντονα χρώματα, ώστε να τραβάνε την προσοχή.

Μέρος 7: Εγκατάσταση εφαρμογής





Σχόλια:

Χρήστης 2: «Ικανοποιεί τις ανάγκες του χρήστη.»

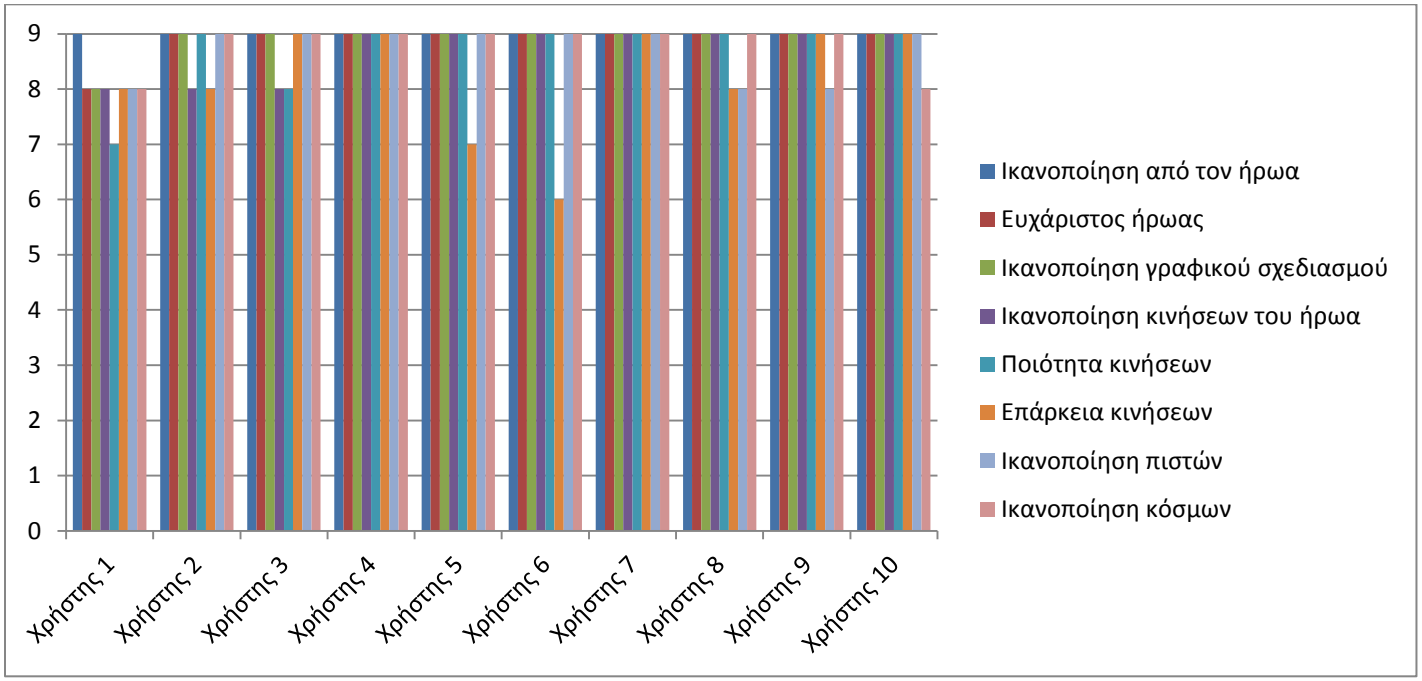
Χρήστης 3: «Η εγκατάσταση της εφαρμογής είναι πολύ εύκολη και γρήγορη. Επιπλέον, η ενημέρωση για την πρόοδο του χρήστη όπως και η επεξήγηση σε περίπτωση ήττας δεν παρουσιάζει δυσκολίες ή ελαττώματα.»

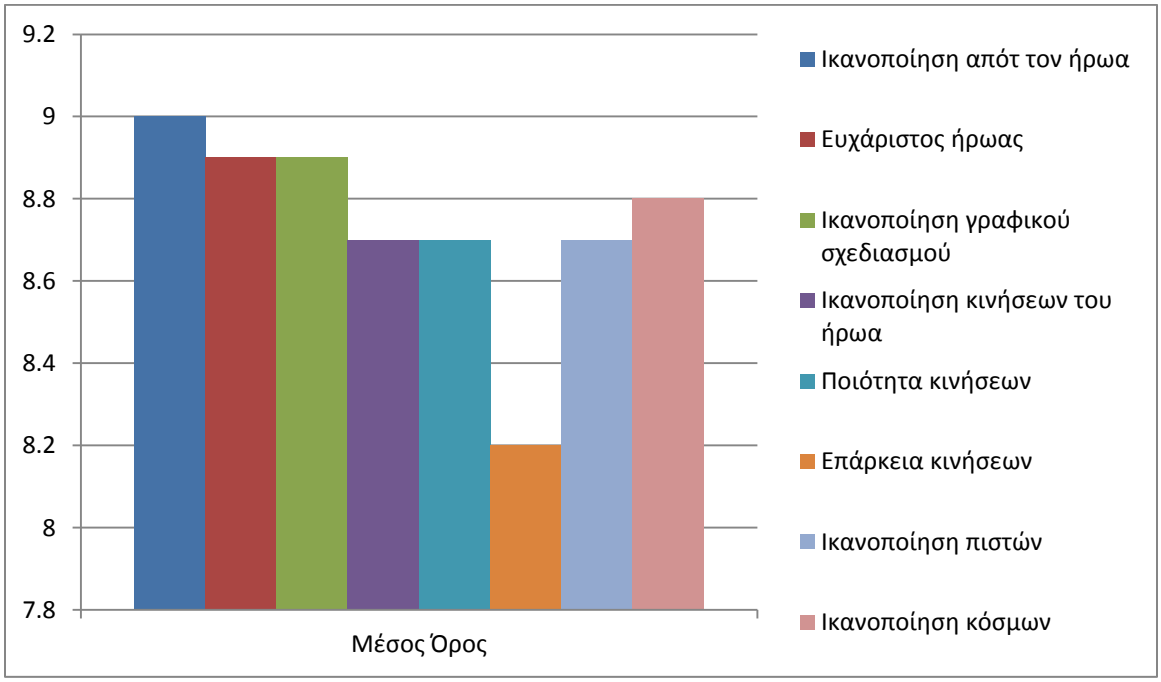
Χρήστης 5: «Υποστηρίζει όλες τις εκδόσεις του Android!»

Συμπεράσματα:

Οι χρήστες ήταν ικανοποιημένοι από την ταχύτητα της εγκατάστασης της εφαρμογής στη συσκευή τους. Επίσης ήταν ικανοποιημένοι από τη διαδικασία εγκατάστασης της εφαρμογής θεωρώντας, ο κάθε χρήστης, την εξατομίκευση εύκολη, τις πληροφορίες προόδου ικανοποιητικές, και τις εξηγήσεις κατά την αποτυχία αρκετά εποικοδομητικές, ώστε να ξέρουν σε ποιο στάδιο βρίσκονται και τι συμβαίνει κατά τη διαδικασία αυτή.

Μέρος 8: Γραφικό και σχεδιαστικό μέρος





Σχόλια:

Χρήστης 1: «Ο ήρωας μοιάζει με έναν ήρωα γνωστής ταινίας κινουμένων σχεδίων, τον πιγκουίνο από την “Μαδαγασκάρη”.»

Χρήστης 2: «Είναι από τα καλύτερα που έχω δει!»

Χρήστης 4: «Πολύ ωραίες πίστες. Ο πιγκουίνος φοβερός. Πιστεύω ότι ο πιγκουίνος θα έπρεπε το βάδισμα του να έχει πιο πολύ πλευρικές κινήσεις (όπως περπατούν οι πραγματικοί πιγκουίνοι) σε συνδιασμό με τα άνω άκρα (τα οποία συμπεριφέρονται πιο πολύ σαν χέρια ανθρώπου). Επίσης στον πιγκουίνο μου φάνηκαν λίγο φαρδιά τα πόδια.»

Χρήστης 5: «Οι κινήσεις του ήρωα θα μπορούσαν να ήταν λίγο περισσότερες. Εκείνο που είναι πολύ εντυπωσιακό και πρωτότυπο είναι ότι μπορείς να παίξεις σε διαφορετικές πίστες που αλλάζει αρκετά το στυλ του παιχνιδιού!»

Συμπεράσματα:

Στους χρήστες άρεσε πολύ ο ήρωάς μας, βρίσκοντάς τον διασκεδαστικό, ευχάριστο και αστείο. Ήταν πολύ ικανοποιημένοι από τη γραφική σχεδίαση του πιγκουίνου. Εντυπωσιάστηκαν από τις κινήσεις του ήρωα, οι οποίες είναι κωμικές και προσφέρουν γέλιο στο παιχνίδι. Αν και ήταν αρκετά ικανοποιημένοι από την ποιότητα και την ποικιλία των κινήσεων, ορισμένοι χρήστες περίμεναν τον πιγκουίνο να περπατάει πιο πολύ σαν τους αληθινούς πιγκουίνους, με περίεργους πλαϊνούς βηματισμούς και όχι τόσο ανθρώπινους. Ξεναγήθηκαν μέσα στις πίστες όλων των κόσμων, τις οποίες βρήκαν πολύ διασκεδαστικές και όμορφα σχεδιασμένες. Ιδιαίτερη εντύπωση τους έκανε η ποικιλία των πιστών, καθώς όλες ήταν διαφορετικές μεταξύ τους με το ίδιο θέμα αναλόγως τον κόσμο, και η ποικιλία των κόσμων, γεγονός που τους τραβούσε το ενδιαφέρον για να τα εξερευνήσουν.

Προτάσεις χρηστών για επέκταση

Χρήστης 4:

«Στο τέλος, αφού ο πιγκουίνος βρει τον θησαυρό και μεταβαίνει στην εικόνα με την συνολική αποτίμηση των νομισμάτων θα μπορούσε ο πιγκουίνος να πανηγυρίζει ή αν δεν συλλέγει τον απαραίτητο αριθμό πόντων να απογοητεύεται με την ανάλογη μουσική. Θα ήταν επίσης ωραίο να βάλεις και χιόνι στην πίστα. Μπορείς στην αρχή του παιχνιδιού να βάλεις και ένα μικρό βίντεο με τον πιγκουίνο και το τέρας να διαλέγονται (δηλαδή να έχει μια υπόθεση, όπως για παράδειγμα να κυνηγούν φαγητό-ψάρια για να επιβιώσουν). Η προηγούμενη ιστορία συνδέεται με την προϋπόθεση ο θησαυρός να είναι ψάρια η θαλασσινά. Αν κατάλαβα καλά επειδή μπορεί να πάρει πολλους δρόμους ο πιγκουίνος θα μπορούσες να βάλεις επιμέρους θησαυρούς προς τον τελικό θησαυρό. Μπορείς επίσης να βάλεις σκηνικό στο οποίο πηδάει απο πάγο σε πάγο ή να κάνει σέρφινγκ σε κομμάτι πάγου με μουσική Χαβάη.»

ΚΕΦΑΛΑΙΟ 7

ΑΠΟΤΕΛΕΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

7.1 Στόχος

Στόχος της διπλωματικής εργασίας είναι η δημιουργία ενός τρισδιάστατου παιχνιδιού για κινητές συσκευές με αυτόνομους διαδραστικούς χαρακτήρες. Εκμεταλλευτήκαμε τις δυνατότητες που προσφέρουν οι φορητές συσκευές και τις χρησιμοποιήσαμε για το χειρισμό του παιχνιδιού.

Το παιχνίδι είναι σχεδιασμένο ώστε να προσαρμόζεται στα διάφορα μεγέθη των οθονών των κινητών συσκευών, ώστε να είναι να είναι εύχρηστο και ευχάριστο ανεξάρτητα από το μέγεθος, και να μην επηρεάζεται η εκτέλεσή του από τις διαφορετικές δυνατότητες των επεξεργαστών που διαθέτουν. Επίσης υποστηρίζει πολλές εκδόσεις Android, καινούργιες και παλιές, για να καλύβει ένα μεγάλο εύρος κινητών συσκευών.

Δημιουργήσαμε τέσσερις διαφορετικούς κόσμους παιχνιδιού, οι οποίο χωρίζονται αναλόγως το θέμα τους, τον τρόπο αξιοποίησης των δυνατοτήτων της συσκευής για τον χειρισμό της κίνησης του ήρωα, και την θέση της συσκευής στην οποία πρέπει να βρίσκεται για να ξεκινήσει το παιχνίδι. Υπάρχει μεγάλη ποικιλία και στο περιβάλλον αλλά και στο σενάριο του κάθε κόσμου ώστε να διατηρείται το ενδιαφέρον του χρήστη και να είναι πάντα ευχάριστο στο μάτι.

Δίνεται στο χρήστη όσο το δυνατόν περισσότερος έλεγχος του παιχνιδιού για να μην περιορίζεται από τις προεπιλεγμένες ρυθμίσεις. Υπάρχουν μενού με τα οποία μπορεί να περιηγηθεί στους κόσμους και στις πίστες του παιχνιδιού, να αποθηκεύσει ή να διαγράψει τα δεδομένα, να διαλέγει τον τρόπο του χειρισμού που επιθυμεί, και να ελέγξει τον ήχο του παιχνιδιού. Τον κρατάμε ενήμερο για τις λειτουργίες του παιχνιδιού με ετικέτες που περιέχουν πληροφορίες σχετικά με τον ήρωα και τα στοιχεία που τον αφορούν, όπως ζωές, εχθροί και βαθμολογία, τα οποία ενημερώνονται συνεχώς. Επίσης διαθέτουμε οδηγίες με εικόνες και επεξήγηση, και μηνύματα στην οθόνη που αναφέρονται στο χρήστη και τον ενημερώνουν για την δράση του. Με τον τρόπο αυτό διατηρούμε την επικοινωνία μεταξύ του χρήστη και του παιχνιδιού μέσω γραφικής διεπαφής.

Στους δύο πρώτους κόσμους ο χειρισμός γίνεται μέσω κουμπιών αφής και ενός χειριστηρίου αφής, όπου η συσκευή μπορεί να βρίσκεται σε πλάγια ή όρθια θέση και να

περιστρέφεται ο κόσμος αντίστοιχα. Στον τρίτο κόσμο ο χειρισμός γίνεται από την κλίση που δίνει ο χρήστης στη συσκευή μέσω του επιταχυνσιομέτρου, όπου η συσκευή πρέπει να βρίσκεται μόνο σε όρθια θέση για να ξεκινήσει το παιχνίδι. Στον τέταρτο κόσμο χρησιμοποιείται η κλίση της συσκευής από το επιταχυνσιόμετρο, και οι αισθητήρες αφής για τον υπολογισμό της κατεύθυνσης του δαχτύλου στην οθόνη, όπου επίσης πρέπει η συσκευή να βρίσκεται σε όρθια θέση.

Ο κάθε κόσμος αποτελείται από οχτώ πίστες κλιμακώμενης δυσκολίας. Σε προχωρημένα επίπεδα συναντάμε αυτόνομους χαρακτήρες με τεχνητή νοημοσύνη για να αντιδράσουν στο παιχνίδι αναλόγως τη συμπεριφορά του ήρωα. Η τεχνητή νοημοσύνη που χρησιμοποιείται αφορά την εύρεση της συντομότερης διαδρομής προς κάποιον στόχο, κινούμενο ή ακίνητο. Μια φάλαινα οδηγεί τον ήρωα προς το σημείο τερματισμού της πίστας ακολουθώντας τη συντομότερη διαδρομή μέσα από ένα επιτρεπτό μονοπάτι. Επίσης ένας βίκινγκ τον κυνηγάει μέσα σε ένα λαβύρινθο για να τον πιάσει, πλησιάζοντάς τον από την πιο σύντομη διαδρομή αποφεύγοντας τα εμπόδια στο δρόμο του. Άλλοι έξυπνοι εχθροί, όπως βίκινγκ και κανόνια, εντοπίζουν τη θέση του ήρωα στον κόσμο και περιμένουν να τους πλησιάσει αρκετά για να του επιτεθούν ή να τον πυροβολήσουν.

Μεγάλη έμφαση δώθηκε στο σχεδιαστικό μέρος του παιχνιδιού. Ο πιγκουίνος δημιουργήθηκε από το μηδέν, ακολουθώντας όλα τα απαραίτητα βήματα για την εφαρμογή του στο παιχνίδι. Οι μέθοδοι που υλοποιήθηκαν περιλαμβάνουν την τρισδιάστατη μοντελοποίηση του χαρακτήρα, την δημιουργία και τοποθέτηση των υφών στην επιφάνεια του χαρακτήρα, τη κατασκευή του σκελετού για την υποστήριξη των animations και την σύνδεσή του με το δέρμα του, τη δημιουργία χειρισμών ελέγχων πάνω στο σκελετό για τον έλεγχο της κίνησης του μοντέλου, και τη δημιουργία των κινουμένων σχεδίων.

Το παιχνίδι αποτελείται από μια μεγάλη ποικιλία κινουμένων σχεδίων ώστε να προσεγγιστεί μια ομαλή, πολύπλοκη και κυρίως αστεία συμπεριφορά του πιγκουίνου. Η αναπαραγωγή των κινουμένων σχεδίων σχετίζεται με τη δράση του ήρωα στο παιχνίδι. Υπάρχουν όμως και μέρη στα οποία παίρνει πρωτοβουλίες και κινείται μόνος του. Στην εισαγωγή του κυρίως μενού κοιτάει το χρήστη και του δείχνει να πατήσει το κουμπί Play. Επίσης κατά την είσοδο σε ορισμένες πίστες, εμφανίζεται ένα βίντεο μικρής διάρκειας, με το οποίο παρουσιάζουμε το σενάριο του κόσμου αυτού, στο οποίο ο πιγκουίνος βγαίνει τρέχοντας από το σημείο εκκίνησης, εμφανίζεται από πίσω του ο εχθρός και κατευθύνονται προς το σημείο έναρξης της πίστας. Όλες οι κινήσεις προσομοιώνονται με βάση τη φυσική, ώστε να έχουμε ρεαλιστικά αποτελέσματα.

Η υλοποίηση του παιχνιδιού ως μια αυτόνομη και πλήρης εφαρμογή έγινε με τη μηχανή παιχνιδιών Unity 3D. Η σχεδίαση του ήρωα και των αντικειμένων στο παιχνίδι έγινε με το πρόγραμμα τρισδιάστατης μοντελοποίησης Maya 2013 της Autodesk. Οι δισδιάστατες

εικόνες του παιχνιδιού δημιουργήθηκαν και επεξεργάστηκαν με το πρόγραμμα Adobe Photoshop CS6. Οι ήχοι προέρχονται από το YouTube, και ορισμένα αντικείμενα και animations του παιχνιδιού προέρχονται από το έτοιμα assets του Unity Asset Store. Η μετατροπή σε εφαρμογή Android της μορφής .apk έγινε μέσω του προγράμματος Android SDK.

7.2 Αποτελέσματα

Από τα αποτελέσματα των ερωτηματολογίων που δόθηκαν στους χρήστες που δοκίμασαν το παιχνίδι και από τις προτιμήσεις που δήλωσαν μέσω της μεθόδου Think Aloud που χρησιμοποιήσαμε, πραγματοποιήσαμε κάποιες αλλαγές και διορθώσεις για τη βελτίωση και τελειοποίηση του παιχνιδιού.

Οι κόσμοι JumpLand και RunLand επιτρέπουν την αναπαραγωγή του παιχνιδιού μόνο όταν η συσκευή βρίσκεται σε όρθια θέση. Όταν βρισκόταν σε πλάγια ή ανάποδη όρθια θέση, εμφανιζόταν ένα μήνυμα περιστροφής της συσκευής σε όλη την οθόνη. Για κάθε θέση εμφανιζόταν μια διαφορετική εικόνα που ενημέρωνε το χρήστη προς τα ποια μεριά να περιστρέψει τη συσκευή για να ξεκινήσει το παιχνίδι. Επειδή αυτό το μήνυμα κατά κύριο λόγο εμφανιζόταν κατά τη μετάβαση από το μενού των πιστών στην αντίστοιχη πίστα, το πρώτο πράγμα που αντίκριζαν ήταν η εικόνα αυτή, καθώς χειριζόμαστε το μενού σε πλάγια θέση. Οι χρήστες προτίμησαν να μην εμφανίζεται καθόλου το μήνυμα περιστροφής, αλλά να είναι εξαρχής τοποθετημένη η πίστα στη θέση που πρέπει να περιστραφεί η συσκευή για να καταλαβαίνουν αμέσως πως πρέπει να την κρατήσουν. Έτσι αφαιρέθηκαν αυτές οι εικόνες και απλά ο χρόνος του παιχνιδιού σταματάει σε άλλες θέσεις.

Το παιχνίδι σταματάει όταν η συσκευή τοποθετηθεί παράλληλα προς το έδαφος με την οθόνη προς τα πάνω ή προς τα κάτω, και εμφανίζεται ένα μήνυμα στην οθόνη το οποίο ενημερώνει το χρήστη ότι πρέπει να σηκώσει τη συσκευή για να συνεχιστεί. Οι χρήστες προτίμησαν να αφαιρεθεί αυτή η λειτουργία, καθώς μπορεί να θέλουν να κρατάνε τη συσκευή με χαμηλή κλίση για να παίξουν και δεν μπορούν καθώς σταματάει το παιχνίδι και εμφανίζεται η εικόνα με το μήνυμα.

Κατά την εισαγωγή σε μία πίστα από το αντίστοιχο μενού, μέχρι να ξεκινήσει το παιχνίδι, εμφανίζεται μια εικόνα με οδηγίες χρήσης και πληροφορίες σχετικά με τη συγκεκριμένη πίστα. Οι χρήστες δεν μπορούσαν να καταλάβουν τι έπρεπε να κάνουν με την εμφάνιση αυτής της εικόνας και πατούσαν πάνω στην οθόνη. Πρότειναν να εμφανίζεται στο κάτω μέρος μια μπάρα φόρτωσης, που να συμβολίζει το χρόνο που απομένει μέχρι να ξεκινήσει η πίστα, την οποία και υλοποιήσαμε.

Στο μενού των κόσμων παρατηρήσαμε ότι κάποιοι χρήστες δεν χρησιμοποιούσαν τα κουμπιά για να κινηθούν δεξιά και αριστερά, αλλά έσερναν το δαχτυλό τους στην οθόνη. Έτσι

προσθέσαμε τον έλεγχο της οριζόντιας κίνησης ενός δαχτύλου στην οθόνη για τη μετατόπιση των καρτελών των κόσμων, παράλληλα με τα κουμπιά που βρίσκονται δεξιά και αριστερά.

Στους κόσμους SnowLand και WaterLand ο χειρισμός της κίνησης γίνεται μέσω ενός χειριστηρίου αφής, ο οποίο είναι αρκετά δύσκολος. Αρκετοί μη έμπειροι χρήστες σε τέτοιες κατηγορίες παιχνιδιών, δυσκολεύτηκαν πολύ να κατευθύνουν τον ήρωα ακριβώς στο σημείο που επιθυμούσαν, και ζήτησαν να μειώσουμε την ταχύτητα κίνησης και περιστροφής ώστε να ελέγχεται πιο εύκολα.

Παρατηρήθηκε επίσης ότι η απενεργοποίηση του ήχου δεν διατηρούνταν κατά την είσοδο σε ορισμένες πίστες. Αυτό διορθώθηκε αποθηκεύοντας την επιλογή του χρήστη σε μια μεταβλητή PlayerPref ώστε να μπορεί να ανακαλεστεί η τιμή της σε άλλες σκηνές, ελέγχοντάς την κάθε φορά για την αναπαραγωγή του ήχου.

Το κουμπί που χρησιμοποιούσαμε για την ρίξη μιας χιονόμπαλας από τον πιγκουίνο, είχε ως συμβολισμό έναν κεραυνό. Οι χρήστες δεν μπορούσαν να καταλάβουν άμεσα τη λειτουργία του κουμπιού αυτού, και προτίμησαν ένα πιο παραστατικό κουμπί σχετικά με τη λειτουργία του. Έτσι αντικαταστήσαμε τον κεραυνό με μια χιονόμπαλα.

7.3 Μελλοντικές βελτιώσεις και επεκτάσεις

Το παιχνίδι που δημιουργήσαμε μπορεί να δεχθεί πολλές διαφορετικές τροποποιήσεις, και βελτιώσεις. Θα μπορούσαμε να κάνουμε περισσότερο αυτόνομους τους χαρακτήρες του παιχνιδιού και πιο έξυπνους βελτιώνοντας την τεχνητή νοημοσύνη που διαθέτουν.

Με την τεχνητή νοημοσύνη που διαθέτουν ήδη, μπορούν να εντοπίσουν τον στόχο και να κατευθυνθούν προς αυτόν ακολουθώντας την πιο σύντομη διαδρομή, αλλά δεν μπορούν να αποφύγουν τα εμπόδια που θα συναντήσουν στο δρόμο τους, με αποτέλεσμα να κολλήσουν εκεί. Θα μπορούσαμε να δημιουργήσουμε έναν πιο έξυπνο αλγόριθμο που να μπορεί να αντιμετωπίσει τέτοιες καταστάσεις και να αποφεύγει οτιδήποτε τον εμποδίζει χρησιμοποιώντας το Navmesh Obstacle της Unity ώστε να αναγνωρίζονται από τον NavMesh Agent.

Οι βίκινγκς στέκονται ακίνητοι στην πίστα και περιμένουν να τους πλησιάσει ο ήρωας για να του επιτεθούν. Θα μπορούσαμε να φτιάξουμε έναν αλγόριθμο με τον οποίο ο βίκινγκ θα περιπλανιέται τυχαία μέσα στην πίστα, μέχρι να συναντήσει τον πιγκουίνο και να του επιτεθεί, χωρίς όμως να περνάει συνεχώς από το ίδιο σημείο, αλλά να περνάει από όλα τα σημεία της επιφάνειας του εδάφους, αποφεύγοντας τα εμπόδια που θα συναντάει.

Στις πίστες που διαδραματίζονται γρήγορα, επειδή ο χρήστης δεν έχει πολύ χρόνο να καταλάβει τι πρέπει να αποφύγει, μπορούμε να εμφανίζουμε ένα σύμβολο στην οθόνη και έναν ήχο που να τον ειδοποιεί ότι πλησιάζει σε κάποιο εμπόδιο και πρέπει να το αποφύγει. Η

θέση στην οποία θα εμφανίζεται θα δηλώνει και την θέση στην οποία θα είναι το εμπόδιο-εχθρός. Για παράδειγμα στην JumpLand ο πιγκουίνος πετάει προς τα πάνω και η έκταση που καλύβει η κάμερα είναι μικρή. Όταν εμφανιστεί ένας βίκινγκ στην οθόνη, ο χρήστης έχει πολύ μικρό περιθώριο να αντιδράσει γιατί πλησιάζει πολύ γρήγορα προς αυτόν. Θα μπορούσε να εμφανιστεί μια εικόνα με έναν ήχο προειδοποίησης λίγο πριν εμφανιστεί ο βίκινγκ, στη θέση στην οποία θα βρίσκεται. Αντίστοιχα και στην RunLand, το οπτικό πεδίο του ήρωα είναι περιορισμένο, ειδικά αν το εμπόδιο βρίσκεται αμέσως μόλις στρίψει στο λαβύρινθο. Μπορεί και εκεί να εμφανίζεται κάποιο σημάδι συνοδευόμενο ίσως με έναν ήχο όταν πλησιάζει σε κάποιο εμπόδιο.

Όσον αφορά την επέκταση, θα μπορούσαμε να ακολουθήσουμε την τακτική των περισσότερων εφαρμογών των παιχνιδιών και να δώσουμε ένα λόγο και μια πρόκληση στο χρήστη για να συλλέγει τα αντικείμενα της κάθε πίστας. Στο παιχνίδι που υλοποιήσαμε, τα νομίσματα που συλλέγονται αφορούν την κάθε πίστα ξεχωριστά, και χρησιμοποιούνται απλά για την απόδοση της βαθμολογίας και την απόκτηση νέου ρεκόρ. Θα μπορούσαμε να αποθηκεύουμε όλα τα νομίσματα που συλλέγονται ανεξαρτήτως την πίστα, τα οποία θα μπορεί να χρησιμοποιήσει ο χρήστης για να αγοράσει με το αντίστοιχο αντίκτυπο κάποια επιπλέον πράγματα στο παιχνίδι ώστε να διευκολυνθεί. Όπως να αγοράσει ζωές όποτε χάνει για να μπορέσει να συνεχίσει από το σημείο που πέθανε και όχι να ξαναπάει από την αρχή. Επίσης θα μπορούσε να αγοράσει κάποια επιπλέον δύναμη για να κάνει πιο δυνατό τον ήρωά του, ή να αγοράσει εξαρτήματα ή ρούχα για να τον καλλωπίσει, ή ακόμα και να αγοράσει άλλον ήρωα προχωρώντας επίπεδο, ή να ξεκλειδώσει κάποιες πίστες.

Οι πίστες που έχουμε υλοποιήσει είναι περιορισμένου μήκους. Θα μπορούσαμε να δημιουργήσουμε μια ατελείωτη πίστα, η οποία θα δημιουργείται καθώς ο ήρωας θα προχωράει. Θα μπορούσαμε να δημιουργήσουμε τη RunLand από μια πίστα οποία θα τρέχει για πάντα και θα στρίβει δεξιά και αριστερά, ή την JumpLand από μια πίστα στην οποία θα πετάει για πάντα. Για να μην επιβαρύνεται το σύστημα, δεν θα μπορεί να γυρίσει πίσω καθώς θα καταστρέφεται το μέρος που προσπερνάει. Η υλοποίηση αυτή μπορεί να γίνει, κατασκευάζοντας μικρά τμήματα της πίστας από τα οποία θα αποτελείται, που θα επαναλαμβάνονται και θα εναλλάσονται διαδέχοντας το ένα το άλλο. Για την ποικιλία του παιχνιδιού η δημιουργία των τμημάτων αυτών θα τοποθετούνται τυχαία κάθε φορά και που θα εμφανίζονται μπροστά από τον ήρωα για να έχει πρόσβαση σε αυτά. Η πίστα θα είναι χωρισμένη σε τμήματα αναλόγως τη δυσκολία, η οποία θα εξαρτάται από την απόσταση που θα διανύει. Για το κάθε τμήμα δυσκολίας, θα έχουμε δημιουργήσει διαφορετικά τμήματα πίστας τα οποία θα εναλλάσονται στα αντίστοιχα πλαίσια.

Μπορούμε να εισάγουμε την υποστήριξη πολλαπλών παικτών από το παιχνίδι, χρησιμοποιώντας την υπομονάδα της Unity για να μπορούν να συνδεθούν πολλοί παίκτες.

Κάνοντας το παιχνίδι multiplayer, θα μπορεί ο χρήστης να στείλει μια πρόκληση (challenge) σε άλλους χρήστες της εφαρμογής, όπου θα συγκρίνεται η βαθμολογία που πέτυχε ο καθένας. Στον νικητή μπορεί να αποδοθεί κάποιο δώρο για να τον βοηθήσει στο παιχνίδι, όπως κάποιο ποσό νομισμάτων, ή κάποια παραπάνω δύναμη που θα μπορεί να χρησιμοποιήσει (είτε μόνιμα είτε μία μόνο φορά), ή το ξεκλείδωμα κάποιας επόμενης πίστας.

Για να γίνει αυτό πρέπει να αποθηκεύονται τα ονόματα των χρηστών σε κάποιον sever από όπου θα μπορεί ο χρήστης να στείλει την πρόκληση. Στο όνομα του κάθε χρήστη μπορεί να αναγράφεται το ρεκόρ ή το επίπεδο στο οποίο βρίσκεται ώστε να ξέρει ο κάθε χρήστης πόσο καλός θα είναι ο αντίπαλός του. Όσο μεγαλύτερου επιπέδου είναι οι χρήστες που θα προκληθούν τόσο μεγαλύτερη θα είναι η ανταμοιβή του νικητή.

7.4 Επίλογος

Πάντα μου άρεσαν τα παιχνίδια και τα κινούμενα σχέδια! Από μικρή ήθελα να φτιάξω το δικό μου παιχνίδι και να ζωντανέψω έναν δικό μου ήρωα. Είμαι ιδιαίτερα χαρούμενη που τα κατάφερα και πραγματοποίησα το όνειρό μου! Θαύμαζα τα παιχνίδια και τα γραφικά κινουμένων σχεδίων που έβλεπα, και αναρωτιόμουν με ποιον τρόπο δημιουργούνται και σχεδιάζονται. Όταν πήρα την απόφαση να αναλάβω τη δημιουργία του ήρωα και του παιχνιδιού, δεν ήξερα απολύτως τίποτα ούτε για τη διαδικασία που απαιτείται αλλά ούτε για τα προγράμματα που χρειάζονται. Το ήθελα όμως πολύ και ξεκίνησα από το μηδέν.

Το παιχνίδι ολοκληρώθηκε μετά από πάρα πολύ κόπο, χρόνο, επιμονή και υπομονή καθώς ακολουθήθηκαν δύσκολες διαδικασίες και έγιναν πολλές έρευνες για την εκμάθησή τους. Η δημιουργία ενός παιχνιδιού συνήθως απαιτεί μια ομάδα δημιουργών που αναλαμβάνει ο καθένας κάποιο ξεχωριστό κομμάτι, άλλος τον προραμματατισμό, άλλος τη μοντελοποίηση. Έτσι χρειάστηκε να καλύψω διαφορετικούς τομείς για τις ανάκες του παιχνιδιού. Θα μπορούσα να πάρω έναν έτοιμο ήρωα, αλλά ήθελα να μάθω πως γίνεται η μοντελοποίηση και η δημιουργία κινουμένων σχεδίων, καθώς μου αρέσει πολύ που είναι δημουργικές και ευχάριστες διαδικασίες! Χρησιμοποίησα επαγγελματικά προγράμματα που έχουν μεγάλες εταιρείες, και απαιτήθηκε αρκετός χρόνος για την εκμάθηση των λειτουργιών τους.

Το αποτέλεσμα όμως επιβράβευσε την προσπάθειά μου! Το παιχνίδι προσεγγίζει τα παιχνίδια της αγοράς και είναι έτοιμο να κυκλοφορήσει στο Android Market της Google Play. Είναι ένα ολοκληρωμένο παιχνίδι, με μια ποικιλία διαφορετικών πιστών και κόσμων, και είναι αυτόνομο καθώς υποστηρίζεται από πολλές συσκευές.

Πλέον, γνωρίζω τι απατείται για την δημιουργία ενός παιχνιδιού, ενός τρισδιάστατου χαρακτήρα και των animations, και μπορώ να πω με σιγουριά ότι θέλω να ασχοληθώ με τον συγκεκριμένο τομέα! Έχει μεγάλα περιθώρια εξέλιξης με ταχύτατους ρυθμούς, καθώς η

τεχνολογία των κινητών συσκευών (στις δυνατότητες του επεξεργαστή και της κάρτας γραφικών) εξελίσσεται συνεχώς και οι χρήστες ανταποκρίνονται άμεσα σε αυτές. Οι εφαρμογές παιχνιδιών σε κινητές συσκευές έχουν μεγάλη απήχηση από τον κόσμο και οι ιδέες για καινούργια παιχνίδια είναι πολλές τα οποία γίνονται ανάρπαστα σε ολόκληρο τον κόσμο σε μικρά χρονικά διαστήματα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Unity: http://unity3d.com/unity

[2] Unity: http://unity3d.com/pages/what-is-unity

[3] Unity: http://unity3d.com/unity/workflow/asset-workflow

[4] Unity Manual: http://docs.unity3d.com/Documentation/Manual/index.html

[5] Unity Tutorials: https://www.youtube.com/channel/UCG08EqOAXJk_YXPDsAvReSg

[6] Unity Tutorials: https://www.youtube.com/channel/UCoT3jV5skJaRQZwwE6_1-Zw

[7] Unity System Requirements: http://unity3d.com/unity/system-requirements

[8] Unity AI: https://unity3d.com/unity/quality/ai

[8] Android Developers: http://developer.android.com/index.html

[9] Android Developers: https://developer.android.com/tools/index.html

[10] Android SDK: http://developer.android.com/sdk/index.html?hl=sk

[11] Apple: http://developer.apple.com/iphone/program/

[12] Maya Autodesk: http://www.autodesk.com/products/autodesk-maya/overview

[13] Maya 2013 Manual:
http://download.autodesk.com/us/maya/maya2013_getting_started/index.html

[14] Maya 2013 Manual:
http://download.autodesk.com/global/docs/maya2013/en_us/index.html?url=files/BoL_Create_a_Maya_light_source.htm,topicNumber=d30e596461

[15] Maya Tutorials: https://www.youtube.com/channel/UCHmAXsicpLK2EHMZo5_BtDA

[16] Maya Tutorials: https://www.youtube.com/channel/UC-Eh-OCUPr_EsOGe7qEtIQA

[17] Maya Tutorials: https://www.youtube.com/channel/UCHuWqUmCMgtCGKCtkzC76JQ

[18] Wikipedia Arcade game: http://en.wikipedia.org/wiki/Arcade_game

[19] Wikipedia Action game: http://en.wikipedia.org/wiki/Action_game

[20] Wikipedia Adventure game: http://en.wikipedia.org/wiki/Adventure_game

[21] Wikipedia Unity game engine: http://en.wikipedia.org/wiki/Unity_(game_engine)

[22] Wikipedia Modo: http://en.wikipedia.org/wiki/Modo_%28software%29

[23] Wikipedia LightWave: http://en.wikipedia.org/wiki/LightWave_3D

[24] Wikipedia Blender: http://en.wikipedia.org/wiki/Blender_(software)

[25] Wikipedia Cinema 4D: http://en.wikipedia.org/wiki/Cinema_4D

[26] Wikipedia Autodesk 3ds Max: http://en.wikipedia.org/wiki/Autodesk_3ds_Max

[27] Wikipedia Cheetah: http://en.wikipedia.org/wiki/Cheetah3D

[28] Wikipedia Autodesk Maya: https://en.wikipedia.org/wiki/Autodesk_Maya

[29] Wikipedia Game engine: http://en.wikipedia.org/wiki/Game_engine

[30] Wikipedia 3D Computer Graphics: http://en.wikipedia.org/wiki/3D_computer_graphics

[31] Wikipedia List of Game Engines: http://en.wikipedia.org/wiki/List_of_game_engines

[32] Wikipedia Mobile Game: http://en.wikipedia.org/wiki/Mobile_game

[33] Wikipedia API: http://en.wikipedia.org/wiki/API

[34] Wikipedia Direct 3D: http://en.wikipedia.org/wiki/Direct3D

[35] Wikipedia OpenGL: http://en.wikipedia.org/wiki/OpenGL

[36] Wikipedia GPU: http://en.wikipedia.org/wiki/GPU

[37] Wikipedia OpenGL vs Direct3D:
http://en.wikipedia.org/wiki/Comparison_of_OpenGL_and_Direct3D

[38] Wikipedia Texture Mapping: http://en.wikipedia.org/wiki/Texture_mapping

[39] Wikipedia Lightmap: http://en.wikipedia.org/wiki/Lightmap

[40] Wikipedia Shader: http://en.wikipedia.org/wiki/Shader

[41] Wikipedia Computer Animation: http://en.wikipedia.org/wiki/Computer_animation

[42] Wikipedia Polygon Mesh: http://en.wikipedia.org/wiki/Polygon_mesh

[43] Wikipedia Game Physics: http://en.wikipedia.org/wiki/Game_physics

[44] Wikipedia OpenGL ES: http://en.wikipedia.org/wiki/OpenGL_ES

[45] Wikipedia Graphics Engine: http://en.wikipedia.org/wiki/Graphics_engine

[46] Wikipedia 3D Graphics Libraries: http://en.wikipedia.org/wiki/List_of_3D_graphics_libraries

[47] Wikipedia Unreal Engine: http://en.wikipedia.org/wiki/Unreal_Engine

[48] Wikipedia List of 3D Modeling Software: http://en.wikipedia.org/wiki/List_of_3D_modeling_software

[49] Wikipedia 3D Modeling Software: http://en.wikipedia.org/wiki/3D_modeling_software

[50] Wiki CG Society 3D Tools: http://wiki.cgsociety.org/index.php/Comparison_of_3d_tools

[51] Statistics Apple 2012: https://www.apple.com/pr/library/2013/01/07App-Store-Tops-40-Billion-Downloads-with-Almost-Half-in-2012.html

[52] Statistics Apple: http://www.apple.com/pr/library/2013/05/16Apples-App-Store-Marks-Historic-50-Billionth-Download.html

[53] Statistics Apple 2013: http://www.statista.com/statistics/263794/number-of-downloads-from-the-apple-app-store/

[54] Latest Statistics Android: http://www.appbrain.com/stats/android-market-app-categories

[55] Statistics Android 2014: http://www.statista.com/statistics/270291/popular-categories-in-the-app-store/

[56] Statistics Apple Store: http://daveaddey.com/appstore/

[57] Statistics Google Play 2012: http://officialandroid.blogspot.gr/2012/09/google-play-hits-25-billion-downloads.html