

3D Driving Simulator Synchronized With Cardiac Timing Intensifying Fear and Threat Processing



Thanos Ypermahos

Thesis submitted for the degree of
Diploma of Science in Electrical and Computer Engineering

School of E.C.E.
Technical University of Crete
Chania 2017

Acknowledgements

Σε αυτή την ενότητα θα ήθελα να ευχαριστήσω την επιβλέπουσα καθηγήτρια μου, Κατερίνα Μανία για την τεράστια ευκαιρία που μου έδωσε να ασχοληθώ με ένα διαφορετικό, απαιτητικό και πάνω από όλα πολύ ενδιαφέρον θέμα, όπως επίσης για τη συνεχή της παρουσία σε οτιδήποτε χρειαζόταν κατά τη διάρκεια του τελευταίου έτους φοίτησής μου. Είναι πολύ σημαντικό να συνεργάζεσαι με ανθρώπους πρόθυμους και ευχάριστους ώστε να φέρνεις εις πέρας οποιαδήποτε δυσκολία προκύπτει. Η συγκεκριμένη εργασία, μου έδωσε τη δυνατότητα αρχικά να ασχοληθώ με τον τομέα ο οποίος με ελκύει όσο τίποτα άλλο στη σχολή μου, τα γραφικά. Ακόμα με βοήθησε να μάθω να δουλεύω με συνέπεια, σε μια ξένη γλώσσα και με ανθρώπους απαιτητικούς από διαφορετικές κουλτούρες και διαφορετικό επιστημονικό τομέα. Ενώ τα γραφικά πραγματικά είναι ένας υπέροχος τομέας και πολύ εντυπωσιακός, όταν κανείς τον συνδυάζει με επιστήμες όπως η ιατρική στην προκειμένη περίπτωση, τα αποτελέσματα ξεπερνούν κάθε προσδοκία.

Δε θα μπορούσα να παραλείψω τις ευχαριστίες μου στους συνεργάτες μας στο Brighton and Sussex University, τον Hugo Chritchley, την Sarah Garfinkel όπως επίσης και τον David Watson για την υπέροχη συνεργασία, την φιλική αλλά και ταυτόχρονα επαγγελματική αντιμετώπισή τους και την μακρόχρονη εμπειρία τους που συνέβαλαν στην ολοκλήρωση αυτής της εργασίας. Τους ευχαριστώ θερμά για το ενδιαφέρον τους, για τον χρόνο που κατέβαλαν όπως επίσης και για τη προσπάθειά τους να ξεπεραστούν οι οποιεσδήποτε τεχνικές δυσκολίες που προέκυψαν. Επιπλέον, θα ήθελα να ευχαριστήσω τους κυρίους Λαγουδάκη και Ζερβάκη για το χρόνο που θα αφιερώσουν στην ανάγνωση αυτής της εργασίας και να τονίσω πως κάθε σχόλιο και διόρθωσή τους είναι καλοδεχούμενη.

Δε θα μπορούσα να μην ευχαριστήσω τους γονείς μου, Μαργαρίτα και Μιχάλη, τη γιαγιά μου Μαρία όπως επίσης και τον αδερφό μου Φώτη για την συμπαράστασή τους σε όλους τους τομείς αυτά τα χρόνια και για τις όμορφες στιγμές που περάσαμε σε αυτή τη πόλη.

Τέλος ένα μεγάλο ευχαριστώ στη κοπέλα μου Αναστασία για την συντροφιά της στο μεγαλύτερο μέρος των σπουδών μου, τη κατανόηση σε κάθε λογής προβλήματα που προέκυψαν και τη στήριξή της όπου και όποτε χρειαζόταν! Επίσης ένα μεγάλο ευχαριστώ στους φίλους μου σε Χανιά και Αθήνα για την παρέα τους και τις όμορφες στιγμές χωρίς τις οποίες δε θα μπορούσα να βγάλω εις πέρας αυτή τη σχολή.

Abstract

This thesis aims to investigate whether intense fear or threat responses, while a person is driving, are induced faster and more efficiently if threat stimuli appears exactly when the heart is beating, at cardiac systole, or in between heartbeats, at cardiac diastole. For instance, in a driving simulation, if debris flies towards a person or the windshield, or a child runs in front of the car, and these stimuli come into view when the heart is exactly making a beat, are these stimuli seen as more “threatening” and are reaction times faster?

The application system was developed in close collaboration between the Technical University of Crete where the technical implementation took place and the Brighton & Sussex Medical School where the experiments are about to begin to be conducted.

A photorealistic driving simulator has been implemented, displayed on an Oculus Head Mounted Display, controlled by an actual wheel interface and pedals. The software is programmed in Unity and communicated a realistic experience of a driving a car on an endless road in nature. A steering wheel as well as controlling pedals have been integrated in the application and have been calibrated in terms of the efficiency and sensitivity in order to achieve a simulation environment which is as close as possible to realistic driving.

The novelty of this thesis is based on the investigation of cardiac timing on responses during driving. A highly accurate method of monitoring the heart rate of individuals is put forward while users are driving. An oximeter attached to the earlobe was integrated in the application and was synchronized at 95% accuracy with an Electrocardiogram attached to the chest. Cardiac timing detection provides input to the driving simulator, which in turn, guides the occurrences of threat-inducing events while a user is immersed in a synthetic environment. These events appear at distinct points of the heart rate, either at cardiac systole or at cardiac diastole. It is important to determine if the reactions of the users are induced faster at cardiac systole when the baroreceptors are fired, or at cardiac diastole when the baroreceptors are quiet. These occurrences of threat-inducing events consist of: living obstacles such as horses, human-beings and non-living obstacles such as barrels.

Table of Contents

Acknowledgements.....	1
Abstract.....	2
Table of Contents	3
1. Chapter 1 - Introduction.....	6
1.1. Scope	9
1.2. Thesis Outline.....	10
2. Chapter 2 – Background.....	12
2.1. Virtual Reality	12
2.2. Head Mounted Displays (HMDs)	14
2.2.1. History of HMDs	14
2.2.2. HMDs in 2017.....	19
2.3. Game Engines.....	22
2.3.1. Unity.....	22
2.3.2. Unreal Engine.....	24
2.3.3. CryEngine	24
2.4. Modeling 3d objects	25
2.5. Animating 3d objects.....	28
2.6. Virtual Environments for Neuroscientific Research	32
2.7. Cardiac Timing Cycle effects.....	33
3. Chapter 3 – Utilized Software	38
3.1. Unity Game Engine	38
3.1.1. Hierarchy.....	39
3.1.2. Project	40
3.1.3. Console.....	41
3.1.4. Scene	42

Table of Contents

3.1.5. Game	44
3.1.6. Inspector	45
3.1.7. Scripting	46
3.2. Modeling.....	50
3.2.1. Blender	50
3.2.2. SketchUp	51
3.3. MySQL Database	53
3.3.1. PhpMyAdmin.....	53
3.3.2. XAMPP	54
3.4. Web Development.....	55
3.4.1. HTML	55
3.4.2. CSS.....	55
3.4.3. JavaScript	56
3.4.4. PHP	56
4. Chapter 4 – Implementation.....	58
4.1. UI implementation.....	59
4.1.1. UI Scripting	63
4.2. 3D Virtual Scene	64
4.2.1. Car modeling.....	64
4.2.2. Environment modeling.....	68
4.2.3. Obstacles modeling	71
4.2.4. Obstacles animation	73
4.2.5. Main Screen Scripting.....	77
4.3. Car Controller.....	78
4.3.1. Keyboard	79
4.3.2. Steering Wheel - Pedals	80
4.4. Audio	81
4.4.1. Audio Scripting	82

Table of Contents

4.5. Oximeter integration.....	83
4.5.1. Connection with Unity	83
4.5.2. Cardiac Beat prediction.....	84
4.5.3. Synchronization with ECG	85
4.6. Events storage.....	86
4.7. Database implementation	87
4.8. Webpage implementation.....	90
5. Chapter 5 - Experiments.....	95
5.1. Experimental Procedure	95
5.1.1. Consent of the Users	96
5.1.2. Demographics form.....	96
5.1.3. Questionnaires.....	99
5.1.4. Heart Detection Tests.....	105
5.1.5. Driving	107
5.2. Results	107
6. Chapter 6 - Conclusions and Future Work.....	110
6.1. Main Contributions.....	111
6.2. Future Work.....	112
References	113

1. Chapter 1 - Introduction

This project is based on previous studies at the Brighton and Sussex Medical School and aims to determine whether the responses of users are faster and more efficient at cardiac systole, or at cardiac diastole. The goal of this project was to develop from scratch a realistic VR driving environment where the users would feel like they are actually driving a car. It is essential to implement a VR simulator that would induce similar driving behavior to the users as in the real world.

Virtual reality could be defined as a believable, interactive 3D computer-generated world that the users can explore so that they feel as if they are really “there”, both mentally and physically.

Putting it another way, virtual reality is essentially:

- **Believable:** The users feel as if they inhabit a virtual world and should keep believing that while interacting with it, or the illusion of virtual reality will disappear.
- **Interactive:** As the users move-around, the VR world moves with them through tracking the head or based on interaction hand-held controllers. They could watch a 3D movie and be transported up to the Moon or down to the seabed.
- **Computer-generated:** Powerful computers, able to display realistic 3D computer graphics, are essential so that believable, interactive, alternative worlds are implemented with minimum latency.

Chapter 1 - Introduction

- **Explorable:** A VR world is often large, consisting of endless terrains and detailed enough for someone to explore.
- **Immersive:** To be both believable and interactive, VR needs to engage both the user's body and mind. Paintings by war artists can give us glimpses of conflict, but they can never fully convey the sight, sound, smell, taste, and feel of battle. Someone can play a flight simulator game on their home PC and be lost in a very realistic, interactive experience for hours (the landscape will constantly change as their plane flies through it), but it's not like using a real flight simulator-where the user sits in a hydraulically operated mockup of a real cockpit and feel actual forces as it tips and tilts, and even less like flying a plane.

Reading a book, looking at a painting, listening to a classical symphony, or watching a movie do not qualify as virtual reality. All of them offer partial glimpses of another reality, but none of them is interactive, explorable, or fully believable. If users are sitting in a movie theater looking at a giant picture of Mars on the screen, and they suddenly turn their head too far, they will realize that they are actually on Earth and the illusion will disappear. If they see something interesting on the screen, they can't reach out and touch it or walk towards it; again, the illusion will simply disappear. So these forms of entertainment are essentially passive: however plausible they might be, they don't actively engage the user in any way.

VR is quite different. It makes users think they actually live inside a completely believable virtual world, in which they are partly or fully immersed. It is two-way interactive: as users respond to what they see, what they see responds to them: if they turn their head around, what they see or hear in VR changes to match their new perspective.

One of the most important features of VR is the immersion in an artificial environment where the user feels the environment is reality. There are two types of the Immersive Virtual Environment, **semi-immersive virtual environments** and **fully-immersive virtual environments**. An example of a semi-immersive virtual environment is a flight simulator in which the user does not become fully immersed ([see Figure \[1.1\]](#)). In a fully-immersive virtual environment (e.g. CAVE system) the user becomes fully immersed within the virtual world ([see Figure \[1.2\]](#)).

In this thesis, a fully immersive 3D driving simulator is implemented displayed in stereo on a head-tracked Head Mounted Display, such as the Oculus

Chapter 1 - Introduction

Rift system. While users are driving, events in the form of obstacles are synchronized with cardiac timing. The user wears an ear lobe oximeter and uses a physical wheel and pedals to drive the car.

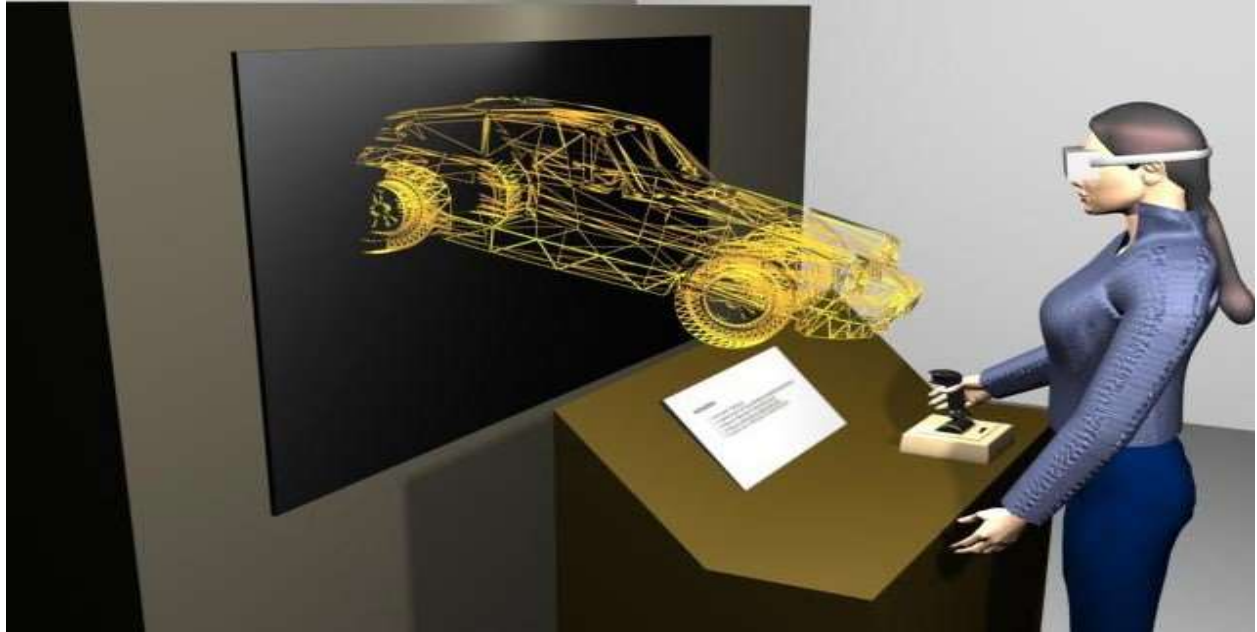


Figure [1.1]: *Example of Semi-immersive virtual environment.*



Figure [1.2]: *Example of Fully-immersive virtual environment.*

Chapter 1 - Introduction

1.1. Scope

Experimentalists have long used text, graphics or computer-based abstractions of real-world objects or situations for experimental psychology or neuroscientific experiments. Such highly controlled but contextually impoverished stimuli greatly simplify the world for research but leave us guessing as to their generalizability. Conversely, therapists and practicing psychologists often relinquish control in order to observe or influence behavior in complex real-world surroundings. Virtual reality provides a middle ground, supporting naturalistic and contextually rich scenarios along with an exacting degree of control over key variables (C. J. Bohil et al., 2011).

VR environments are increasingly being used by neuroscientists to simulate natural events and social interactions. They create interactive, multimodal sensory stimuli that offer unique advantages over other approaches to neuroscientific research and applications. In addition, VR systems may provide limited but compelling haptic feedback that simulates the feel of forces, surfaces and textures as users interact with virtual objects (C. J. Bohil et al., 2011).

The general goal of this project was to create a realistic 3D car driving simulator to be used for neuroscientific experimentation, in collaboration with the Brighton and Sussex Medical School. This project actually builds on earlier work at the Brighton and Sussex Medical School which shows that although many studies suggest that perceptual processing, particularly to painful stimuli, tend to be inhibited at cardiac systole, fear detection and processing is selectively enhanced at systole (S. Garfinkel & H. Critchley, 2016). Because of this conflict, the goal of the experiments to be conducted is to investigate whether users react faster at cardiac systole expressing, in this way, enhanced fear processing or whether they react slower at cardiac systole based on their inhibited processing. This work also builds upon previous research at the Brighton and Sussex medical School looking at responses to ballistic stimuli (H. Prins et al., 2015) that are time-locked to distinct parts of the cardiac cycle and also demonstrated that at cardiac systole people tend to have faster reactions.

The aim was to develop a Virtual Environment which, as previously described is believable, explorable, semi-immersive, interactive, and computer generated. The VR driving simulator implemented was installed at the Brighton and Sussex Medical School in May 2017. User's pulse rate was being monitored with a pulse oximeter

Chapter 1 - Introduction

fully synchronized with an ECG, while driving the virtual car for about 15 minutes. Several obstacles such as barrels, horses were directed either towards the car from different directions, for instance exactly in front of the car, or other obstacles, such as human beings, were crossing the road some meters ahead of the car. The user had to try to be focused and avoid as many obstacles as possible and as fast as possible. In that way, the reaction times for either brake or swerve were stored and allowed the experimenters to further analyze them.

The first phase of pilot experiments utilizing the VR driving simulator has been already conducted in May of 2017 at Brighton and Sussex Medical School. Users were comfortable and excited to use the VR driving simulator and completed the tasks successfully. Preliminary results indicated that, indeed, for certain users, their driving behavior is affected by cardiac timing as their reaction time for all categories of obstacles were faster at diastole. The main experiments will take place in the summer of 2017 at the Brighton and Sussex Medical School and relevant data analysis will correlate the average reaction time of every individual with a distinct point of their cardiac cycle (e.g. systole-diastole) or with any personal characteristics, such as their age, gender, awareness of pulse rate, or even with any obstacle characteristics, such as the category of obstacle, their speed, direction etc.

1.2. Thesis Outline

This thesis is divided into six chapters and organized to provide a continual narrative to the reader. The structure is explained below.

Chapter 2 – Background: This chapter acts as a prologue in order to explain the fundamental principles of Virtual Reality (VR). Firstly, we define the term and the background of Virtual Reality, what exactly VR is and how people experience it. Afterwards, a background analysis of Head Mounted Display (HMD) technology is presented. Then, a description of existing HMDs in 2017 follows by explaining some of their most important features. The most well-known Game Engines are also described as well as a technical analysis of 3D modeling and animation. Finally, we discuss the significance of Virtual Reality in Neuroscience and detail existing literature concerning the cardiac cycle effects for which the technical implementation of this thesis is going to be used to investigate.

Chapter 1 - Introduction

Chapter 3 – Software Architecture and Development: The third chapter presents the software that was used to develop the application to be utilized for the experiments. A detailed description of the Unity 3D software platform is provided including the most important components of it, mainly focusing on its capabilities of programming geometry behaviors, UIs and integration of multiple hardware components. A brief description of two industry-standard 3D modeling software was used in order to edit and create the 3D models required for our project. Furthermore, a description of the fundamental tools to create a MySQL Database follows based on the platforms XAMPP and PhpMyAdmin. Finally, the Web Development tools that were used are briefly presented.

Chapter 4 – Implementation: In this section, the implementation of the 3D Driving Simulator is described in detail. The User Interface is fully described and every decision made is fully explained. The 3D Virtual Scene follows, including its components such as the car, the environment, the obstacles as well as the way we modeled and animated them is clearly presented. The controllers of the car follow and are briefly described. The significance and the implementation of the audio and the oximeter integration with the VR driving simulator is explained in detail. In addition, the events that are being stored while the user is driving, are discussed as well as the relevant database architecture utilized to store such events. Finally, the way the data is presented in a web interface as well as the generic system architecture is implemented from scratch.

Chapter 5 – Experiments: The fifth chapter focuses on the presentation of the experimental procedure, which shows in detail the steps followed to conduct the experiments. The procedure consists of the consent form, a demographics form, the questionnaires, the heart detection tests and finally the driving. In addition, the results of the preliminary phase of the experiments is analyzed and explained in this chapter.

Chapter 6 – Conclusions & Future Work: This chapter presents the conclusions, as well as hints about potential future work that could possibly extend this thesis.

2. Chapter 2 – Background

This chapter aims to offer to the reader background knowledge relevant to this thesis. A definition of virtual reality is provided. A brief history of Head Mounted Displays follows from 1930 till today. HMDs at the present time are described and a concise description of them in order to show how much they evolved since the previous available devices. Game Engines which provide the software framework for the creation and development of 3D interactive applications, are also discussed in this chapter. Furthermore, the reason why the modeling and animation are such important steps in Virtual Reality is described in detail. The significance of VR as a tool providing stimuli for neuroscientific experiments as well as previous research relevant to the cardiac timing cycle effects affecting threat responses, are provided in this chapter.

2.1. Virtual Reality

Virtual reality has been notoriously difficult to define over the years. Many people take "virtual" to mean fake or unreal, and "reality" to refer to the real world. This results in an oxymoron. The actual definition of virtual, however, is "to have the effect of being such without actually being such". The definition of "reality" is "the property of being real", and one of the definitions of "real" is "to have concrete

Chapter 2 – Background

existence". Using these definitions "virtual reality" means "to have the effect of concrete existence without actually having concrete existence", which is exactly the effect achieved in a good virtual reality system. There is no requirement that the virtual environment match the real world. Inspired by these considerations, for the virtual driving simulator we adapt the following definition:

Virtual reality is the use of computer technology to create the effect of an interactive three-dimensional world in which the objects have a sense of spatial presence.

In this definition, "spatial presence" means that the objects in the environment effectively have a location in three-dimensional space relative to and independent of user's position. It should be noted that this is an effect, not an illusion. The basic idea is to present the correct cues to the user perceptual and cognitive system so that his brain interprets those cues as objects "out there" in the three-dimensional world. These cues have been surprisingly simple to provide using computer graphics: simply render a three-dimensional object (in stereo) from a point of view which matches the positions of users' eyes as they move about. If the objects in the environment interact with the user then the effect of spatial presence is greatly heightened. We should also bear in mind that it is not required that the virtual reality experience is "immersive". While for some applications the sense of immersion is highly desirable, it is not obligatory for virtual reality. The main point of virtual reality, and the primary difference between conventional three-dimensional computer graphics and virtual reality is that in virtual reality the user is working with things as opposed to pictures of things.

Virtual reality is an artificial environment that is created with software and presented to the user in such a way that the user suspends belief and accepts it as a real environment. On a computer though, virtual reality is primarily experienced through two of the five senses: sight and sound. Virtual reality means creating immersive, computer-generated environments that are so convincing users that they will react the same way they would in real life. The idea is to block out the sensory input from the outside and use the visual and auditory cues to make the virtual world seem more real. While the concept is simple, actually building virtual reality systems has proven difficult to do, until recently.

Chapter 2 – Background

2.2. Head Mounted Displays (HMDs)

2.2.1. History of HMDs

When we commonly think of Virtual Reality our first thought is to turn to a modern VR headset as well as all of the various PC applications which are beginning to include virtual-reality support. Virtual-reality actually has an extensive history with a concept that dates all the way back to the 1930s. In this section we will present briefly the history of VR and how it has evolved since an early concept into the amazing simulation experience that anyone can have in his home today. In the schema below ([see Figure \[2.1\]](#)), we can see how the VR technology has developed.

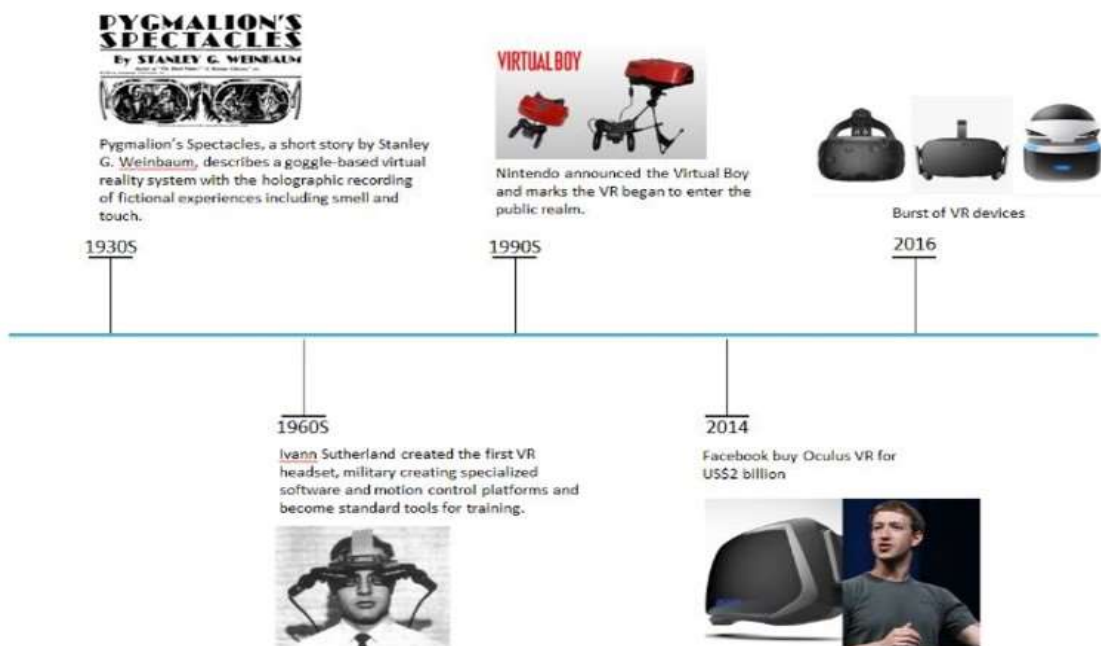


Figure [2.1]: *Virtual Reality History from 1930-2016.*

In Pygmalion's Spectacles 1930, Stanley G. Weinbaum explains a goggle based game in which individuals can watch a holographic recording of virtual stories including touch and smell. This fantastic vision of the future would actually turn into in what we think today of virtual-reality. While it is difficult to introduce touch and smell elements into the average virtual-reality experience, these are visions that creators have in mind for the very near future of virtual-reality experiences. It is

Chapter 2 – Background

amazing to think that 85+ years ago, people were already thinking about creating simulation experiences using technology. We still think of these types of plans as we look towards the future of VR. With rapidly changing technology however, these improvements to the VR simulation experience may be far closer than decades away.

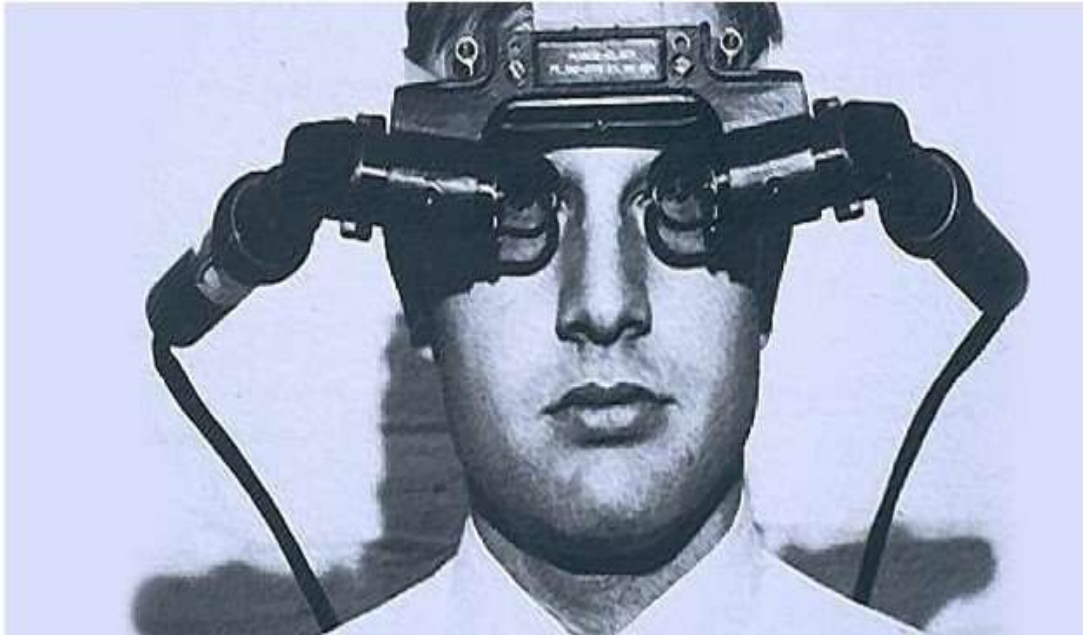


Figure [2.2]: *First VR headset in 1960.*

In 1960's the VR headsets actually started being developed ([see Figure \[2.2\]](#)). Only after 30 years from the original thought of a VR headset, Ivan Sutherland created the very first VR headset for use in military applications. Using a specialized military software as well as a motion control platform, the first VR headsets were designed for use in training exercises. These VR training tools have now become the standard in the military training for flight exercises, combat situations and more. An immersive experience is definitely required to push military personnel and prepare them in a safe training environment before they enter into the field. Many air forces around the world require an extensive amount of VR training simulations before they will even let a pilot into one of their aircraft. With the initial development using specialized software and motion controls, VR research would continue to pave the way for training in the military and beyond. Today's military VR headsets are far more advanced, compact and immersive and these training programs as well as the

Chapter 2 – Background

technology will continue to develop a little bit faster than some of the products that we might find as regular consumers.



Figure [2.3]: *Virtual boy by Nintendo in 1990.*

In 1990, VR headsets begin to make their way into a few arcade games for simulations and Nintendo announced the first home VR system ([see Figure \[2.3\]](#)). Virtual Boy was one of the first home systems available for use with a widespread appeal. Sega also introduced a Sega VR headset for the Sega Genesis console in the year 1993. These wraparound prototypes had stereo sound, LCD screens and head tracking. Technical development in this VR headset doom the project and the cost

Chapter 2 – Background

of the headset was extensive making it a massive flop for Sega. The Virtual Boy however was a 3-D game console that experienced a little more success. Virtual Boy was released in North America at a price of \$180. The games were entirely made in red and black and there were only a few pieces of software available with the device. Users would wear a VR headset and control the action on a regular Nintendo controller. Unfortunately the console was very uncomfortable to use and because of the lack of games as well as the lack of color, it didn't present the same strong sales as the other Nintendo consoles out at the time.



Figure [2.4]: *Oculus Development Kit 1 in 2014.*

Oculus VR represents the latest revolution in VR technology. When Facebook officially acquired the Oculus VR system, this showed that virtual-reality was becoming a huge concern for many of the world's top developers. Although the Oculus Rift was formed out of a kick starter campaign in 2012, the deal in 2014 represented a huge boost in their funding and confidence ([see Figure \[2.4\]](#)). Later in 2015, Oculus began to acquire other companies like Surreal Vision and built

Chapter 2 – Background

partnerships with Samsung to develop the Samsung gear VR. As one of the leading companies in VR development, Oculus has fully functioning VR systems for consumers to purchase and use at home. With support for a wide range of applications and further applications being developed for specific use with their VR systems, this was a huge leap forward for VR. After gaining international attention with the demand after the Facebook deal, this would propel many other developers into creating their own VR development firms. Oculus in a way started a brand-new VR renaissance with a call to create immersive and simulated experiences for the average consumer. Previous VR headsets were very technical and inaccessible for the average computer user, but with plug-and-play compatibility and a wide host of supported applications, Oculus gave the average consumer hope that they can enjoy VR again.

With the initial surge after Oculus Rift, companies all over the world began building their own VR headsets and producing some fantastic new tech. With so many new devices coming out from many of the world's top manufacturers, we are seeing huge developments when it comes to apps, 360° cameras, inexpensive headsets, VR glass experiences and more. As 3-D graphics continue to get better and better, while they are processing power lines at an exponential rate, VR is becoming a focus for many developers in the future. There are many consumer products coming out to compete with Oculus based on its demand as well as generic products for use with smart phone technology. As many smart phones have the accelerometer data, advanced soundcards and graphic sets for 3d rendering users are opting to watch 360 videos and try virtual-reality apps with their smartphone and home devices like Google Cardboard.

Oculus has released the consumer-ready Rift, HTC and Valve have put out the Steam-friendly Vive, Sony has launched the excellent PlayStation VR, Samsung recently added a separate controller to its Gear VR, and Google's Daydream platform is just starting to emerge like a butterfly from its Cardboard cocoon. There are a lot of promising headsets across a lot of different price and power spectrums.

Chapter 2 – Background

2.2.2. HMDs in 2017

Modern VR headsets fit under one of two categories: mobile or tethered. Mobile headsets are shells with lenses into which the user places his smartphone. The lenses separate the screen into two images for their eyes, turning user's smartphone into a VR device. Mobile headsets like the Samsung Gear VR and the Google Daydream View are inexpensive at \$100 or less, and because all of the processing is done on user's phone, they do not need to connect any wires to the headset.

However, because phones aren't designed specifically for VR, they cannot offer the best picture even with special lenses, and they're notably underpowered compared with PC- or game console-based VR. Qualcomm showed off some cool Snapdragon 835-powered prototype headsets at CES that let the user walk around a virtual space without needing to be plugged into anything or have sensors installed around user's room. And Google recently announced standalone Daydream headsets from HTC and Lenovo that don't require a phone and use built-in position tracking.

Tethered headsets like the Oculus Rift, HTC Vive, and PlayStation VR are physically connected to PCs (or in the case of the PS VR, a PlayStation 4). The cable makes them a bit unwieldy, but putting all of the actual video processing in a box so that the user do not need to directly strap to his face means his VR experience can be a lot more complex. The use of a dedicated display in the headset instead of users' smartphone, as well as the use of built-in motion sensors and an external camera tracker, drastically improves both image fidelity and head tracking.



Figure [2.5]: *HTC Vive - Sony PlayStation VR.*

Chapter 2 – Background



Figure [2.6]: *Oculus Rift DK 3 – Windows 10 VR headset.*

Sony's PlayStation VR ([see Figure \[2.5\]](#)), offering the most polished and easy-to-use tethered VR experience with a relatively reasonable price tag. Users can only play proprietary games on it, but a theater mode lets them play any PS4 game as if they were sitting in front of a large screen. Like the Rift, it also requires an additional investment for full functionality; it is required to have a PlayStation Camera for the headset to work at all, and a PlayStation Move controller bundle for motion controls. Still, for a \$400 headset, that means the total is still less than the price of the Rift.

HTC's Vive ([see Figure \[2.5\]](#)) is a comprehensive package that includes a headset, two motion controllers, and two base stations for defining a "whole-room" VR area. It's technically impressive, and is the only VR system that tracks user's movements in a 10-foot cube instead of from their seat. It also includes a set of motion controllers more advanced than the PlayStation Move. But its \$800 price tag is a pretty high amount of money, and PC-tethered VR systems like the Vive need plenty of power, with HTC recommending at least an Intel Core i5-4590 CPU and a GeForce GTX 970 GPU.

The Oculus Rift ([see Figure \[2.6\]](#)) has become synonymous with VR, even if the brand has lost some of its luster against the HTC Vive and the PlayStation VR. The retail version of Oculus Rift is out, and while it's more expensive than the developer kits were, it's also much more advanced. From a technical standpoint, the headset is nearly identical to the Vive. It costs \$200 less than the Vive as well, but it lacks the Vive's whole-room VR.

Microsoft has announced partnerships with multiple hardware manufacturers to offer a variety of Windows 10-compatible VR headsets ([see Figure \[2.6\]](#)) on top of the Vive and Oculus Rift (which have their own software ecosystems with Steam and the Oculus Store), and the recent Windows 10 Creators Update has introduced some VR features into the operating system. These headsets will use outward-facing

Chapter 2 – Background

sensors for motion sensing, so they won't need external cameras or sensors like the Rift, Vive, and PS VR. Microsoft has also been working on the HoloLens, an expensive and still developing augmented reality headset with a lot of potential. Just keep in mind that AR is not VR.



Figure [2.7]: *Google Daydream – Samsung Gear VR.*

Google's Daydream ([see Figure \[2.7\]](#)) is similar to Google Cardboard in concept. The user still has to put his phone in an inexpensive headset (the \$79 Daydream View), and it functions as his display thanks to a set of lenses that separate the screen into two images. A pairable remote that user hold in his hand (similar to the Oculus Remote) controls the action. It's impressive when somebody can find apps that work with it, but the software library is currently very light and it isn't backward compatible with Google Cardboard apps (though Google is working on that with an SDK update).

Samsung's Gear VR ([see Figure \[2.7\]](#)) is one of the most accessible VR systems, with a catch. To use the newest Gear VR, it is required to own a compatible Samsung Galaxy smartphone (currently eight devices, ranging from the Galaxy S6 to the S8). This narrows down potential users to people who already own compatible Samsung phones, since buying one just to use with the Gear VR pushes the price to HTC Vive levels. On the bright side, Samsung regularly bundles the Gear VR with its flagship phones, so if users are planning to pick up a Galaxy S8, they might get a headset for free with the purchase.

The now-\$130 Gear VR is a bit more expensive than both the previous iteration and the Google Daydream View, but it comes with a new Bluetooth controller equipped with both a touch pad and motion sensing, in addition to the

Chapter 2 – Background

touch pad built onto the headset itself. Samsung collaborated with Oculus to build the Gear's software ecosystem, which features a solid handful of apps and games, and multiple ways to consume 360-degree video.

It is undeniable that in 2017, HTC Vive and Oculus Rift have made a dent in the VR revolution, made better by the constant introduction of new peripherals and features that make it even tougher to decide between the two of them. The main reason why we selected to proceed in the development of this project with Oculus Rift instead of the HTC Vive is because the 3D driving simulator implemented did not require users to move around in the room wearing the headset so the main advantage of the Vive was negated. It is also, way easier setting up the Oculus Rift and it takes much less effort and time to start using it, so we decided to select it for this project's needs.

2.3. Game Engines

2.3.1. Unity

Unity programming environments supports both 2D and 3D game development, which is quite unusual for a game engine. That said, Unity was really designed for 3D games with 2D support bolted on afterwards; the 2D features were initially just for building menus and other 2D screens needed in a 3D game, to avoid the need for an external tool. The features were quite generic and developers started building games with them; probably due to the broad cross-platform support. To their credit, Unity have supported this and continue to invest in the area.

Three development languages are officially supported: C#, UnityScript (basically JavaScript with type annotations) and Boo. The last of these, Boo is not widely used and probably best avoided. The Unity's development kit community has widely adopted C# and the majority of plugins and examples use it. If anyone prefers JavaScript and only has a very simple project in mind then UnityScript is a good option. After starting using plugins written in C# that potentially need to call back into UnityScript code, issues will probably come up with compilation order.

Additionally, Unity has a lot of great futures such as:

Chapter 2 – Background

- Strong community of asset and plugin creators – there are a lot of free and reasonable priced content available.
- Visual editing tools are excellent and the editor can be extended with plugins.
- It supports a wide range of asset formats and converts automatically to optimal formats for the target platform.
- It supports a very wide range of platforms: mobile, desktop, web and console.
- Deployment to multiple platforms is very easy to manage.
- The 3D engine produces high quality results without any complex configuration
- There is a free license that covers the majority of features.
- Paid licenses are very affordable for most professional developers, available on subscription for \$75 per platform currently (some platforms are free).

On the other hand, there are some consequences, such as:

- Collaboration is difficult. Unity has an expensive asset server product to help teams collaborate. If somebody does not use it, sharing code and assets between team members can be painful. The best option is to enable and use external source control but there are several binary files that cannot be merged and updating assets often causes them to break things in scenes, losing connections to scripts and other objects.
- Performance is not great – until very recently Unity ran almost entirely in a single thread and made almost no use of the extra cores in most mobile devices – this is improving in Unity 5. The compilers are not at all well optimized for the ARM processors in almost all mobile devices – Unity have decided to transpire to C++ and use LLVM to get a more optimized build rather than solve this problem directly in future releases.
- The engine source code is not available. Even paying users do not get to see the Unity source code, which means if users come across a bug in the engine they have to wait for them to fix it or work around it. It is always going to be more critical for users than it is for them. This also limits the ways in which user can extend or customize the engine.

Chapter 2 – Background

2.3.2. Unreal Engine

Unreal is one of the most popular game engines to develop high-end triple-A titles for years now. Gears of War, Batman: Arkham Asylum, Mass Effect, and many other blockbuster games were developed with this engine. Below are its pros and cons.

Some of the best features of Unreal Engine are:

- With so many developers using it, Unreal offers the largest community support. Several lifetime hours of video tutorials and assets are available.
- Best support and update mechanism of all engines, with a new tool introduced with each new update.
- There is widest range of easy to maneuver tools up under its sleeve. There are few tools that can be maneuvered even by a school kid.
- Compatible with diverse operating platforms including iOS, Android, Linux, Mac, Windows, and most game consoles.
- The new licensing terms of \$19 a month and a 5 percent royalty only if user's game makes over \$5,000 make Unreal Engine 4 much more competitive than it had been in the past.

On the contrary, there are some developers who complain a lot about the unfriendly tools that involve a bit of a higher learning curve.

2.3.3. CryEngine

This game engine has received praise for beautiful graphics output. If somebody has a knack for pretty game visuals, this can be the ideal game engine for him. But this powerful game engine has its problems, too.

Some of the pros are:

Chapter 2 – Background

- CryEngine makes the game ambience pretty with its artist-level programming capability in its Flowgraph tool.
- It has the most powerful audio tool, Fmod, inside it, so sound designers love this engine as well as programmers.
- The game engine also offers the easiest A.I. coding of any tech on the market.
- For a beginning developer, UI scale form comes handy.

On the other hand, there are some cons too:

- The free version of the game engine lacks proper customer support.
- Being relatively new to the industry, the engine is yet to find a robust community.
- Learning curve is pretty challenging for a starter.

There are a lot of game engines that offer a plethora of features to users such as: HeroEngine, Rage Engine, Project Anarchy, GameSalad, GameMaker: Studio, App Game Kit, Cocos 2D and more, but we are not going to describe them in the context of this thesis.

Unreal Engine and Unity are currently ahead of the competition as the two most popular game engines available to the public. This is due to the fact that they both succeed in providing high-end graphics, a large variety of usable tools, great support for platforms and devices, without compromising usability and efficiency. It is important to note that these 2 Game Engines offer a large community support, which is also something that has to be considered when choosing the right Game Engine. CryEngine is also great and powerful engine with remarkable capabilities, however its complicated structure and smaller community excluded it from our consideration. In conclusion, taking into account the advantages and disadvantages of each engine, Unity proved to be the ideal choice for this project, mainly due to its efficiency, large community support and ease of use.

2.4. Modeling 3d objects

3-D modeling is the use of software to create a virtual three-dimensional model of some physical objects ([see Figure \[2.9\]](#)). Actually, it is used in many different industries, including virtual reality, video games, 3D printing, marketing, TV and motion pictures, scientific and medical imaging and computer-aided design

Chapter 2 – Background

and manufacturing CAD/CAM. The software generates a model through a variety of tools and approaches including: **simple polygons**, **3-D primitives** (simple polygon based shapes such as pyramids, cubes, spheres, cylinders and cones), **spline curves**, **NURBS** (non-uniform rational b-spline) -- smooth shapes defined by bezel curves, which are relatively computationally complex.

2-D geometric polygon shapes are used extensively in motion picture effects and 3-D video game art. Creating approximations of shapes made with polygons is much more efficient in raster graphics, which are required for real time 3-D gaming.

In art for video games and motion picture effects, a model might start as a rough-out using polygon primitives or NURBS, or as a design made by following contours on multiple 2-D isometric views. If the model is to be animated, careful consideration of the arrangement of continuous edge loops must be maintained in the model's polygons around areas of deformation such as joints. A model that looks good stationary will fold very fast in animation when the appearance of the stationary end model is all that's considered during building.

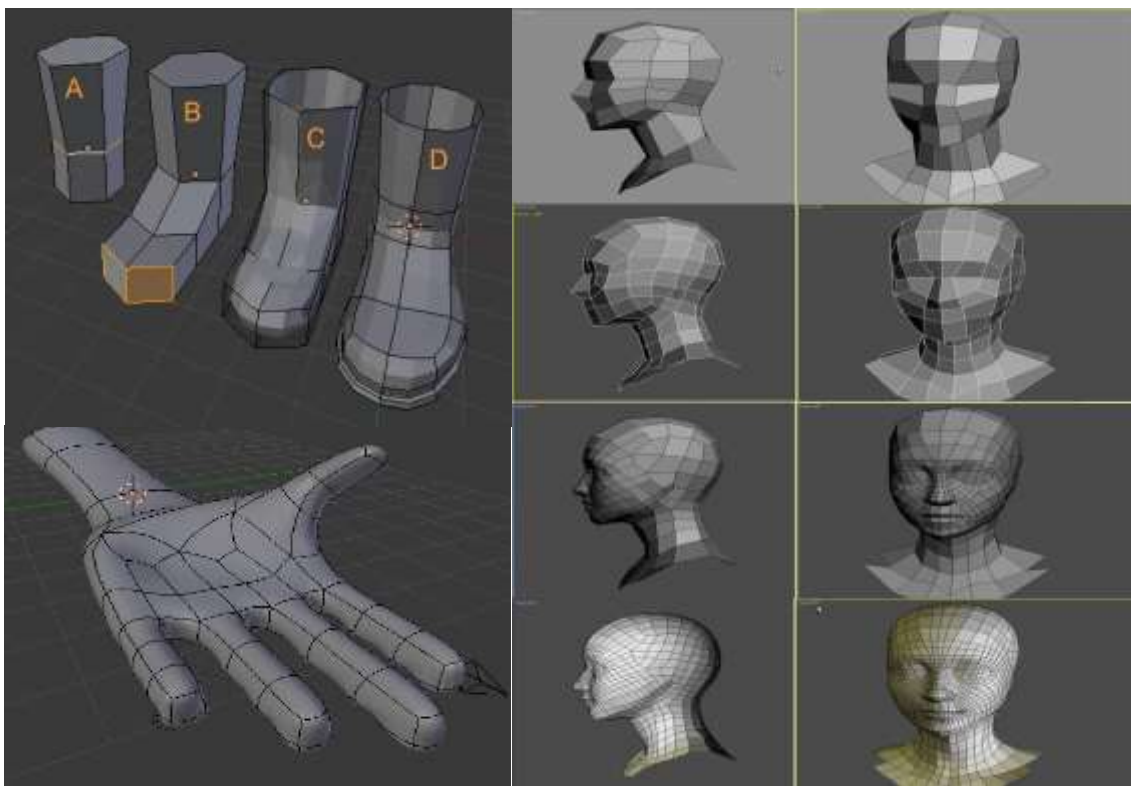


Figure [2.9]: *Examples in modeling.*

Chapter 2 – Background

Once a model is adequately built, an artist might arrange the coordinates of the model to match its 2-D textures in a process called UV mapping, a process that is kind of like trying to design and tailor with a computer mouse. Areas that require more detail are given more space in the UV map. This can be done either using a repeating texture such as a checker board as a place holder or by using an existing texture.

Generally, the next step might be to texture the model, which is to apply either hand-painted or photograph-based 2-D images, usually TGA (Targa bitmap), to the model that will define: the color (color map), the reflectivity (specular map) and the surface texture (bump/normal/deformation map).

Animated models require an extra step of rigging, which is like giving them a virtual skeleton with bones and joints along with the controllers to manage it. The way the texture of these joints influences the surface texture under deformation must be defined in skinning, where someone paints the weight of joint influence on the textures directly on the model's polygons; a polygon painted more heavily is more heavily influenced by a selected joint's movement. The model is then ready for the animator.

More computational and expensive methods of making models such as NURBS may be used, along with complex shaders that interact with particle-based light, in rendered graphics when real time is not a necessity.

VR applications are becoming more feasible due to better and faster hardware, and due to new technology and faster network connections they also start to appear on the Internet. However, the development of such applications is still a specialized, time-consuming and expensive process.

More than 40 years ago, (Masahiro Mori 1970), then a robotics professor at the Tokyo Institute of Technology, wrote an essay on how he envisioned people's reactions to robots that looked and acted almost like human. In particular, he hypothesized that a person's response to a humanlike robot would abruptly shift from empathy to revulsion as it approached, but failed to attain, a lifelike appearance. This descent into eeriness is known as the uncanny valley ([see Figure \[2.8\]](#)).

Chapter 2 – Background

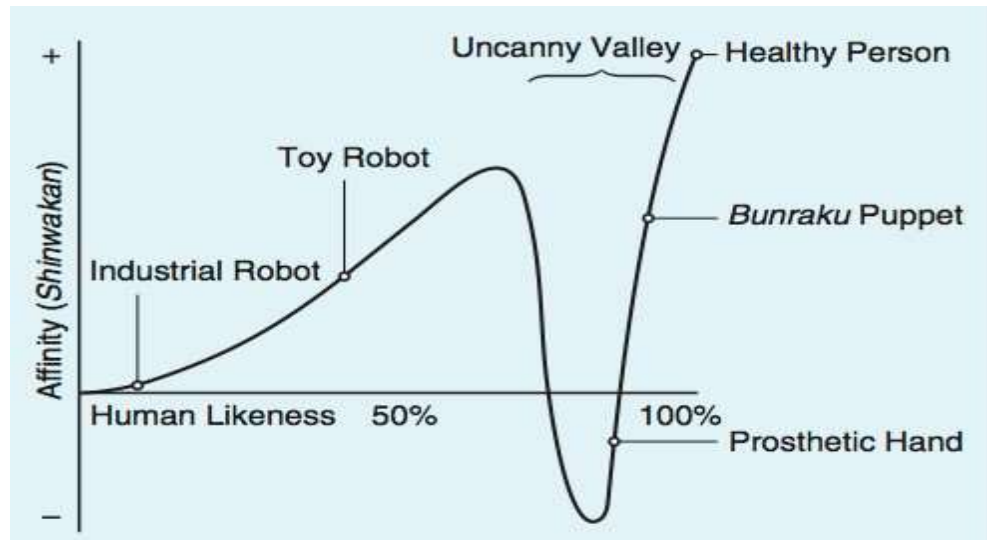


Figure [2.8]: *Uncanny Valley.*

Recently, the concept of the uncanny valley has rapidly attracted interest in robotics and other scientific circles as well as in popular culture. Some researchers have explored its implications for human-robot interaction and computer-graphics animation, while others have investigated its biological and social roots.

Despite the common belief that VR technology can be used as an alternative to reality, no evidence exists of neural equivalence between the perception and the elaboration of VR and real stimuli (Perani et al., 2001). This equivalence is assumed, however, in neuroscience, when VR is used to investigate spatial (Aguirre et al., 1996), visuomotor (Decety et al., 1994; Ghahramani and Wolpert, 1997) and emotional (Nakamura et al., 1999) processing.

In this project, Blender and Sketch Up were selected as the modeling suites which provide us the opportunity to create some essential game objects for our application or even edit some existing models to adjust them in our needs. A variety of tools was used so that the objects would have the intended shape.

2.5. Animating 3d objects

Animation is created by moving a sequence of images rapidly. It is a type of illusion created because of the persistence vision of the eyes. Among different types

Chapter 2 – Background

of animation presently 3D animation has evolved with the help of advanced software and technology. 3D animation consists of varying properties of a 3-dimensional scene defined in numerical quantities. A 3D model can change properties such as position, rotation, shape and surface style. An animated 3D scene is defined by the change of these numerical properties through time ([see Figure \[2.10\]](#)).



Figure [2.10]: *Animation by creating a sequence of images rapidly.*

Apart from 3D objects, a scene contains a camera (point of view) and lights which can also be animated. In order to create an animation each "state" of the 3D scene needs to be rendered to create a frame. The playback of these individual renderings at a certain rate (24, 25 30 frames per second) creates the illusion of animation. As all properties are defined numerically, one can take advantage of various processes for automating the process of generating different "states" for each frame.

In addition, a central notion to animation is the use of "keyframes"; with its origin in classic (paper-based) animation; keyframes allow for the definition of the main "states" in a particular movement or action while the continuity between these states is generated. In classic animation, these in-between frames would be drawn by "junior" animators. In 3D animation and other forms of computer animation, these frames are generated by interpolating between the numerical values that are defined in any two consecutive keyframes. Typically, in 3D animation this interpolation takes the form of 3 dimensional Bézier curves (paths) which are

Chapter 2 – Background

constructed as series of control points, allowing the interactive manipulation of smooth 3D curves.

The computer-intensive process of generating hundreds of rendered images together with the difficulty and labor-intensive processes required for defining 3 dimensional movements makes the exploration of 3D animation the almost exclusive property of large budget film and TV productions.

The recent development of various automation algorithms together with the increasing power of desktop computers make it possible to create very high-quality animations with software such as 3D Studio Max and Electric Image.

Apart from the basic properties of movement (changes in the position of a particular object), rotation and scale, there are tools which provide higher levels of control for the animation of one or more objects in a particular scene.

Deformation tools can be applied to virtually any object; by defining a notional cube around an object; its points can serve as paths for transformations such as twisting, bulging the lines between two points, bending etc. The transformation of the notional cube is then applied to the faces, points and lined of the object "inside". Two states of this deformation process can be used as key frames for the generation of an animated twist or bend.

This linear type of transformation is often a long way away from the realistic movement required in most professional animation. In order to simulate "real world" behaviors several tools provide various definitions for the interdependence between two or more objects; in a 3D model consisting of two objects, a hierarchy can be defined in such a way that one object always "looks" at the other (i.e. the rotation parameter of one object is derived from the position of the other). This kind of behavior can be applied to a spotlight following a particular object, thus automating the animation of one of the objects necessary to create a "realistic" 3D scene.

Similarly, physical laws can be applied to an object in order to modulate its movement; e.g. gravity will automatically make an object fall on to the ground plane at a speed dependent on the object's mass.

Kinematics allow for the definition of the motion of bodies in relation to their own structure. This is particularly useful in the animation of human- or animal-based objects. A humanoid figure is a hierarchical group of limbs consisting of several segments. These segments (e.g. upper arm, lower arm) are defined as joint structures. This type of hierarchy can define the possible movements of the limb: raising the leg

Chapter 2 – Background

forces the foot to a certain movement (i.e. the foot must raise with the leg). A more useful and common interface is known as "inverse kinematics". This allows one to define the movement of a limb by changing the position of an extremity (e.g. moving the foot up will make the lower leg, knee and upper leg move with it). The variety of ways in which our limbs move in relation to each other makes it hard for kinematics to predict the appropriate movements when realistic results are required.

Motion capture technology is behind some of the most realistic results in the animation of physical world-based actions. Using an array of cameras, objects (or more typically the human body) are "marked" with light reflecting points. As the body moves, it is possible to track the position of each of these markers and define them as moving points in a 3D scene. Provided the markers are placed in strategic parts of the body (joints) a skeleton figure can then be constructed with the motion data mapped on to it. There are various motion capture standards as far as data structure is concerned. The integration of some capture systems with animation packages such as 3D Studio Max provides powerful tools for editing and creating skeleton-based figures. Motion capture techniques can also be used to animate abstract models which follow a hierarchy compatible with the motion data format.

The construction of complex 3D scenes is dependent on numerous techniques and is often achieved through the combination of various tools such as motion capture, keyframe interpolation, kinematics, generators such as particle systems and smoke creators. In addition to this variety of animation techniques the use of digital photography and video is essential for the capturing of real-world textures and environments that create contexts for animated objects. The integration of all of these elements is certainly the most challenging aspect of professional animation.

For this project's needs, Unity Animator has been used in order to animate any game object that needed an animation. Unity has a rich and sophisticated animation system which offers a lot of things, such as: Easy workflow and setup of animations for all elements of Unity including objects, characters, and properties, simplified workflow for aligning animation clips, management of complex interactions between animations with a visual programming tool, animating different body parts with different logic etc. Unity's animation system (Known as "Mecanim") comes with a lot of concepts and terminology so that anytime the developer can solve any question that might come up.

Chapter 2 – Background

2.6. Virtual Environments for Neuroscientific Research

This thesis is based on the implementation of a 3D VR simulator which is going to be utilized for neuroscientific studies investigating the effect of threat inducing events on the reaction times of individuals at distinct times of cardiac cycle and its goal is to clarify a neuroscientific conflict. On one hand, there are some studies which support that perceptual processing, particularly of painful stimuli are inhibited at cardiac systole (McIntyre et al. 2008, Edwards et al. 2001, Edwards et al. 2008), but on the other hand there are certain studies which demonstrate that short-term interoceptive fluctuations enhance perceptual and evaluative processes specifically related to the processing of fear and threat and counter the view that baroreceptor afferent signaling is always inhibitory to sensory perception (S. Garfinkel and H. Chritchley, 2016).

Neuroscience is the collective branch of fields of academia which seek to study and understand the human nervous system. It's a smorgasbord of knowledge for a pretty complex system: one that coordinates our voluntary and involuntary movements. Essentially, it's this intermingled web that helps our body send signals it receives from stimulus in the physical world to our brain, and from our brain back to our organs and limbs. In other words, it is what allows us to interact with the organs responsible for interpreting our five senses.

VR system components work in concert to create sensory illusions that produce a more or less believable simulation of reality. The goal is to foster brain and behavioral responses in the virtual world that are analogous to those that occur in the real world. VR systems are best at displaying visual and auditory information. Increasingly, these are approaching the sensory vividness of the physical environment. In addition, VR systems may provide limited but compelling haptic (tactile) feedback that simulates the feel of forces, surfaces and textures as users interact with virtual objects. VR systems also include a way of interacting with the simulation. In fully 'immersive' VR systems, movement of the body and the sensory flow of the virtual environment are coupled. Movements of the head and body are often tracked so that the visual experience changes in a way that corresponds to real world head and body movements. (G. Corey et al., 2011).

The use of VR in neuroscience research offers several unique advantages. Firstly, and perhaps most importantly, VR allows naturalistic interactive behaviors

Chapter 2 – Background

to take place while brain-heart or any other activity is monitored via imaging or direct recording. This allows researchers to directly address many questions in a controlled environment that would simply not be possible by studying performance ‘in the wild’. Secondly, VR environments allow researchers to manipulate multimodal stimulus inputs, so the user’s sensorimotor illusion of being ‘present’ is maximized. By providing realistic stimulation to multiple sensory channels at once, VR engages the sensorimotor system more fully than the simple stimuli used in most psychological research, increasing the potential to elicit realistic psychological and behavioral responses. (G. Corey et al., 2011).

Based on the Review of VR in Neuroscience (Pedro Gamito et al. 2011), VR holds two chief properties that enable users to experience the synthetic environment as being real: immersion and interaction. The first relates to the sensation of being physical present and perceptually included in the VR world. The second stands for the ability to change the world properties, i.e. the environment and its constituents react according to participants actions. Along with imagination, interaction and immersion concur to create the so called “sense of being there” or presence.

This characteristic of VR settings has been acknowledged by the psychotherapists as a media to expose patients with anxiety disorders (AD) to anxiogenic cues within an ecologically sound and controlled environment. VR designed for therapeutic purposes can replicate any of the anxiogenic situations, enabling a better approximation to the anxiogenic world and inducing higher levels of engagement when compared to traditional imagination exposure (G. Riva, 2005). Hyperrealistic threatening stimuli provided by VR leads to higher attention, and subsequent encapsulation, which means, once the fear system is activated the participant perceives the synthetic world as being real (A.O. Re. Hamm & A.I. Weike, 2005). Also, VR reduces the decalage between reality and imagination, by diminishing potential distraction or cognitive avoidance to the threatening stimuli (F. Vincelli & G. Riva, 2000).

2.7. Cardiac Timing Cycle effects

This thesis is putting forward a 3D VR driving simulator utilized for neuroscientific experiments. In these experiments, the users will drive a car in a virtual world while their heart rate will be monitored with very high accuracy. The

Chapter 2 – Background

users will have to avoid obstacles which will be directed towards them at specific distinct points of their heart rate. The hypothesis is that the fear detection and processing is enhanced at cardiac systole because of the baroreceptors are fired in bursts after each heartbeat, but on the other hand there are some studies which demonstrated that perceptual processing is inhibited at cardiac systole. We now explain the cardiac timing cycle effects on threat responses.

It is investigated (S. Garfinkel et al., 2014) whether the processing of brief fear stimuli is selectively gated by their timing in relation to individual heartbeats. Two experiments took place which resulted in the conclusion that fearful faces were detected more easily and were rated as more intense at systole than at diastole. It was demonstrated that fear processing is modulated by cardiac timing. Specifically, the processing of fear faces was enhanced at systole and attenuated at diastole. Behaviorally, during an attentional blink task, the detection advantage typically afforded to emotional stimuli was seen for fear faces presented at systole, but was suppressed at diastole.

Likewise, intensity ratings of fear faces were significantly greater at systole relative to ratings of fear faces presented at diastole. Moreover, these effects were weighted by individual differences in the expression of anxiety symptoms. Specifically, the relative inhibition of fear at diastole was perturbed by heightened anxiety, suggesting a potential mechanism through which altered heart–emotion coupling could contribute to sustained threat processing in highly anxious individuals. These novel findings highlight a major channel by which short-term interoceptive fluctuations enhance perceptual and evaluative processes specifically related to the processing of fear and threat and counter the view that baroreceptor afferent signaling is always inhibitory to sensory perception.

Based on a study, (S.Garfinkel & H.Chritchley, 2016), emotions encompass internal physiological changes which, through interoception (sensing bodily states), underpin emotional feelings, for example, cardiovascular arousal can intensify feelings of fear and anxiety. The brain is informed about how quickly and strongly the heart is beating by signals from arterial baroreceptors. These fire in bursts after each heartbeat, and are quiet between heartbeats. The processing of fear stimuli is selectively enhanced by these phasic signals and these inhibit the processing of other types of stimuli including physical pain. In addition, heart-timing experiments highlight the role of individual heartbeats in gating conscious access of emotionally potent material. The fluctuating physiological state of the body can determine how

Chapter 2 – Background

we sample the world. We detect and process threat better when it looms at the time of our heartbeat than between heartbeats.

Many studies suggest that perceptual processing, particularly of painful stimuli, is inhibited when stimuli are presented at systole. Thus, neural and behavioral responses to painful stimuli (e.g., brief electric shocks to the skin) are weaker if the pain is delivered during baroreceptor activation at systole. Systole attenuates electroencephalographic (EEG) signatures of pain, inhibits pain-related motor reflexes (blink or limb withdrawal), and decreases the perception of pain. At systole, the delivery of brief pain stimuli boosts the baroreflexive inhibition of sympathetic nerves that supply muscle vascular beds. This effect fails to habituate in people who faint at the sight of blood, who are also more prone to anxiety, suggesting a link with constitutional aspects of emotion (S. Garfinkel & H. Chritchley, 2016).

Emotions, and their coupling with bodily physiology, are evolutionarily linked to biological imperatives and motivations that help to ensure the survival of an individual. The influence of interoception is apparent from studies exploring the malleability of body ownership and the strength of self–other boundaries. For example, there was an experiment about called the rubber-hand illusion: while watching the stroking of a nearby rubber hand, when simultaneously feeling one's own hand being stroked, can induce the illusory feeling that the artificial hand is part of one's own body. Interoceptive signals determine the degree to which this feeling of ownership occurs: individuals are less likely to experience the illusory extension of self if they can perform accurately on a heartbeat detection task, suggesting a strong interoceptive basis to self-representation. Moreover, if the external, artificial hand pulsates in time with the participant's own heartbeat this can enhance the experience of illusory ownership of the artificial hand.

The framework of predictive coding has been applied to account for interoceptive influences on consciousness. At this experiment, the brain is proposed to make sense of an abundance of information by generating predictive models about expected sensory inputs. High-level predictions may ‘cancel-out’ expected ascending sensory data, leaving ‘prediction errors’ to be further processed. The feeling of selfhood may arise from internal agency associated with accurate predictions about the inner physiological state of the body, while negative emotions such as anxiety may emerge from interoceptive prediction errors. (S. Garfinkel & H. Chritchley, 2014)

Chapter 2 – Background

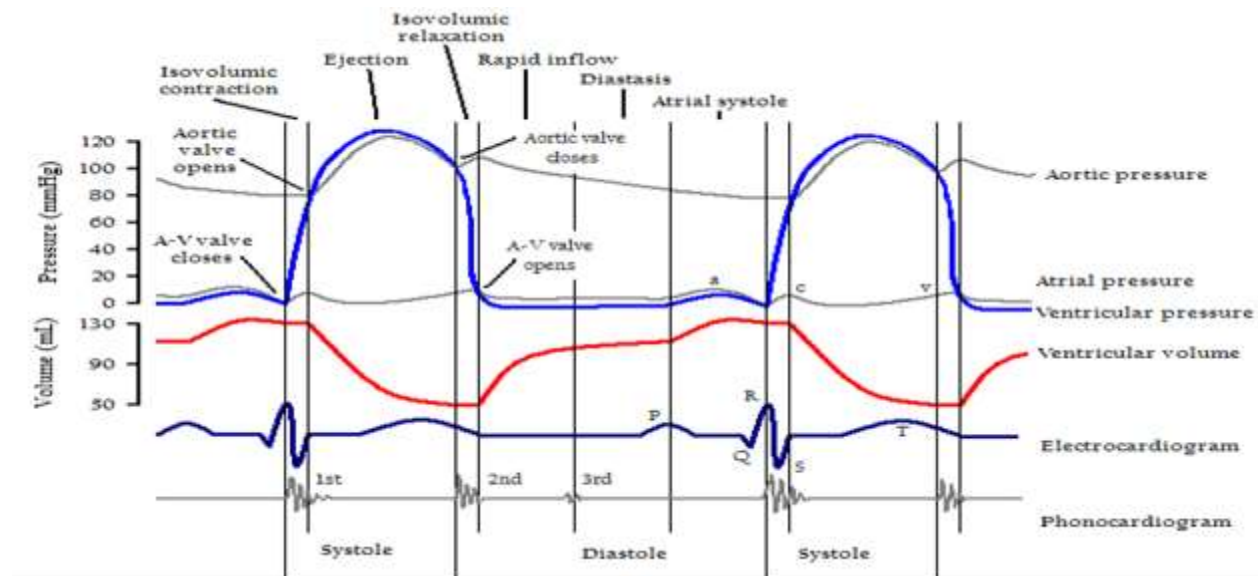


Figure [2.11]: *Cardiac events occurring in the cardiac cycle. Two complete cycles are illustrated.*

According to influential theories of emotional decision making, interoceptive signals serve to guide behavior by influencing the incentive value of stimuli. It follows that the effect of interoceptive signals on perception and action should be particularly pronounced for potentially threatening stimuli. Reaction times are influenced by their timing within the cardiac cycle. (H. Prins et al., 2015)

During cardiac systole, when arterial baroreceptors fire to individual heartbeats, larger objects evoked faster motor reactions, and their perceived strength of impact was less affected by looming speed. There was no demonstrable effect of cardiac cycle on the response to, or perceived impact of, the smaller objects. People with greater interoceptive accuracy manifest an attenuation of effects of the cardiac phase on reactions to large looming stimuli. These findings highlight the influence of brief interoceptive signals from the heart on behavioral and subjective responses to dynamic physical threats to the body and the modulation of these effect by specific attributes of the stimuli and by individual differences in interoceptive accuracy of the observer. These short term physiological influences on stimulus processing and behavior have implications for the efficiency of rapid protective reactions and the optimization of human machine interaction. (H. Prins et al., 2015)

Subjective ratings correlate with speed, size and initial distance of the stimuli do not seem to interact with the main effect of cardiac timing. Early detection of

Chapter 2 – Background

stimuli is directly facilitated by interoceptive signals from the heart, as long as their detection is relevant to the observer to provide an incentive. (H. Prins et al, 2015)

In another experiment that was conducted in 2016 (L. Pramme et al., 2016), participants had to classified target letters in a visual selection task where their attentional selection was investigated. Stimulus onsets were aligned to the R wave of the electrocardiogram and stimuli presented either during the ventricular systole or diastole. Selection efficiency was operationalized as difference in target selection performance under conditions of homogeneous and heterogeneous distractors. The results of this study demonstrated that the onset of stimulus presentation in regard to the cardiac cycle affects attentional abilities to select a target stimulus from distractors in a way that selection efficiency is enhanced during phases of baroreceptor activation (systole) compared to baroreceptor inactivation (diastole). In other words, increased interference due to context conditions that hamper selection are of less consequence in phases of increased baroreceptor activation.

After taking into consideration all of the above, an application was developed for a specific experimental procedure to be conducted at the Brighton and Sussex Medical School, where users will take part in a 3d car driving simulation while their cardiac rate will be monitored. Several obstacles will appear in front of the car shield while they are driving, timed at distinct points of their cardiac cycle and their goal would be to avoid these obstacles. The aim is to clarify whether the users' reaction times are faster at cardiac systole or at cardiac diastole taking input from previous research which demonstrated both of these theories.

3. Chapter 3 – Utilized Software

In this chapter, every single program that contributed in the development of this project is going to be fully explained. The Unity Game Engine which was the programming framework that has been selected to develop our game will be presented in detail. The modeling software, consisting of Blender and Sketch up programs is going to be explained as well as the MySQL database implementation. Finally, the web interface tools that were used in order to develop the web interface for this application are presented in detail.

3.1. Unity Game Engine

For the purposes of this project, the Unity Game Engine has been selected to provide the development environment. Unity was selected because of its ease of use, the numerous online guides/tutorial, the ability to create 2D graphics for user interfaces, the familiar scripting language (C#), the Unity's Asset Store, where it is easy to find objects without having to create everything from scratch, the thriving and supportive community and last but not least the ability to use the full range of Game Engine tools and programming capabilities for free.

The main components/windows of Unity Game Engine are: Hierarchy, Project, Console, Scene, Game, Inspector and, most importantly, Scripting mechanisms. A more detailed description of each component follows.

3.1.1. Hierarchy

The Hierarchy window (see [Figure \[3.1\]](#)) contains a list of every GameObject in the current Scene. Some of these are direct instances of Asset files (like 3D models), and others are instances of Prefabs, which are custom objects that make up most of the game. As objects are added in and removed from the Scene, they appear and disappear from the Hierarchy as well.



Figure [3.1]: *Hierarchy window.*

By default, objects are listed in the Hierarchy window in the order they are created. Re-ordering of objects can be done easily by dragging them up or down, or by making them “child” or “parent” objects. For instance, in [Figure \[3.1\]](#) the Object 1 is the parent object and Object 2, Object 3 are children of it.

Chapter 3 – Utilized Software

3.1.2. Project

In this window (see Figure [3.2]), the user can access and manage the assets that belong to his project. It consists of two panels. The left panel of the browser shows the folder structure of the project as a hierarchical list. When a folder is selected from the list by clicking, its contents will be shown in the panel to the right panel. The user can click the small triangle to expand or collapse the folder, displaying any nested folders it contains.

The individual assets are shown in the right panel as icons that indicate their type (script, material, sub-folder, etc.). The icons can be resized using the slider at the bottom of the panel; they will be replaced by a hierarchical list view if the slider is moved to the extreme left. The space to the left of the slider shows the currently selected item, including a full path to the item if a search is being performed.

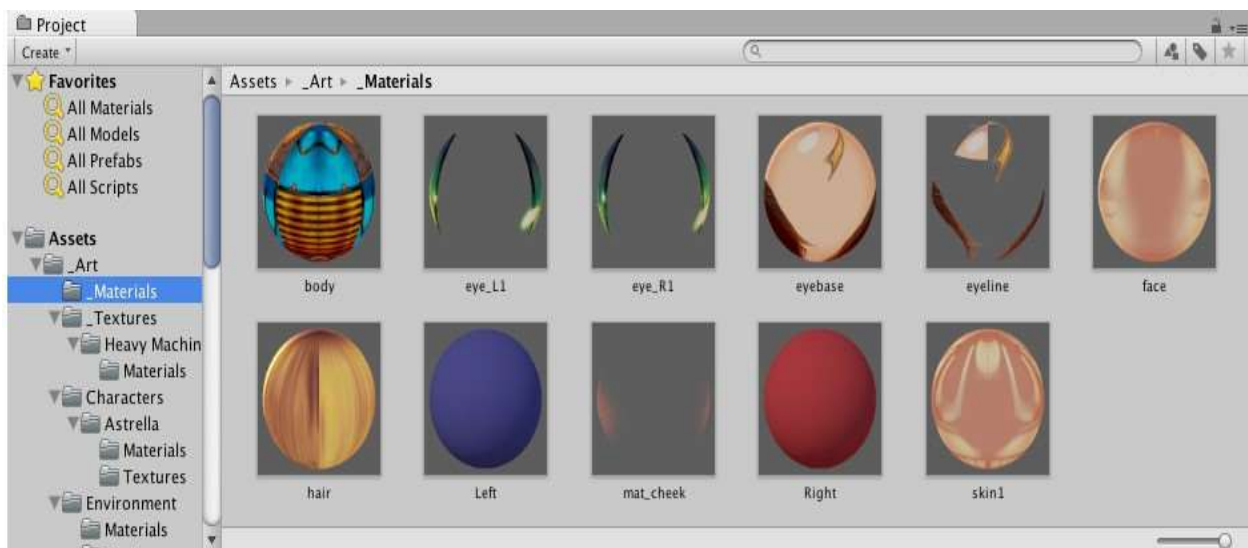


Figure [3.2]: *Project window.*

Just above the panel is a “breadcrumb trail” that shows the path to the folder currently being viewed. The separate elements of the trail can be clicked for easy navigation around the folder hierarchy. When searching, this bar changes to show the area being searched (the root Assets folder, the selected folder or the Asset Store) along with a count of free and paid assets available in the store, separated by a slash.

Chapter 3 – Utilized Software

There is an option in the General section of Unity's Preferences window to disable the display of Asset Store hit counts if they are not required.



Figure [3.3]: *Asset Store search.*

The Project Browser's search can also be applied to assets available from the Unity Asset Store. If the user selects the option of Asset Store from the menu in the breadcrumb bar ([see Figure \[3.3\]](#)), all free and paid items from the store that match user's query will be displayed. If they select an item from the list, its details will be displayed in the inspector along with the option to purchase and/or download it. Some asset types have previews available in this section so the user can, for example, rotate a 3D model before buying.

3.1.3. Console

The Console Window ([see Figure \[3.4\]](#)) shows errors, warnings and other messages generated by Unity. To aid with debugging, the user can also show his own messages in the Console using the implemented functions of Unity (Debug.Log, Debug.LogWarning and Debug.LogError).

Chapter 3 – Utilized Software



Figure [3.4]: Console Window.

The toolbar of the console window has a number of options that affect how messages are displayed. The Clear button removes any messages generated from user's code but retains compiler errors. The user can also arrange for the console to be cleared automatically whenever he runs the game by enabling the Clear on Play option. There is also the opportunity to change the way messages are shown and updated in the console. The Collapse option shows only the first instance of an error message that keeps recurring. This is very useful for runtime errors, such as null references, that are sometimes generated identically on each frame update. The Error Pause option will cause playback to be paused whenever `Debug.LogError` is called from a script. Finally, there are two options for viewing additional information about errors. The Open Player Log and Open Editor Log items on the console tab menu access Unity's log files which record details that may not be shown in the console.

3.1.4. Scene

The Scene window ([see Figure \[3.5\]](#)) is the interactive view into the world that the user is creating. Scene View can be used to select and position scenery, characters, cameras, lights, and all other types of Game Objects. Being able to Select, manipulate and modify objects in the Scene View are some of the first skills somebody will need to begin his first steps in Unity.

Chapter 3 – Utilized Software

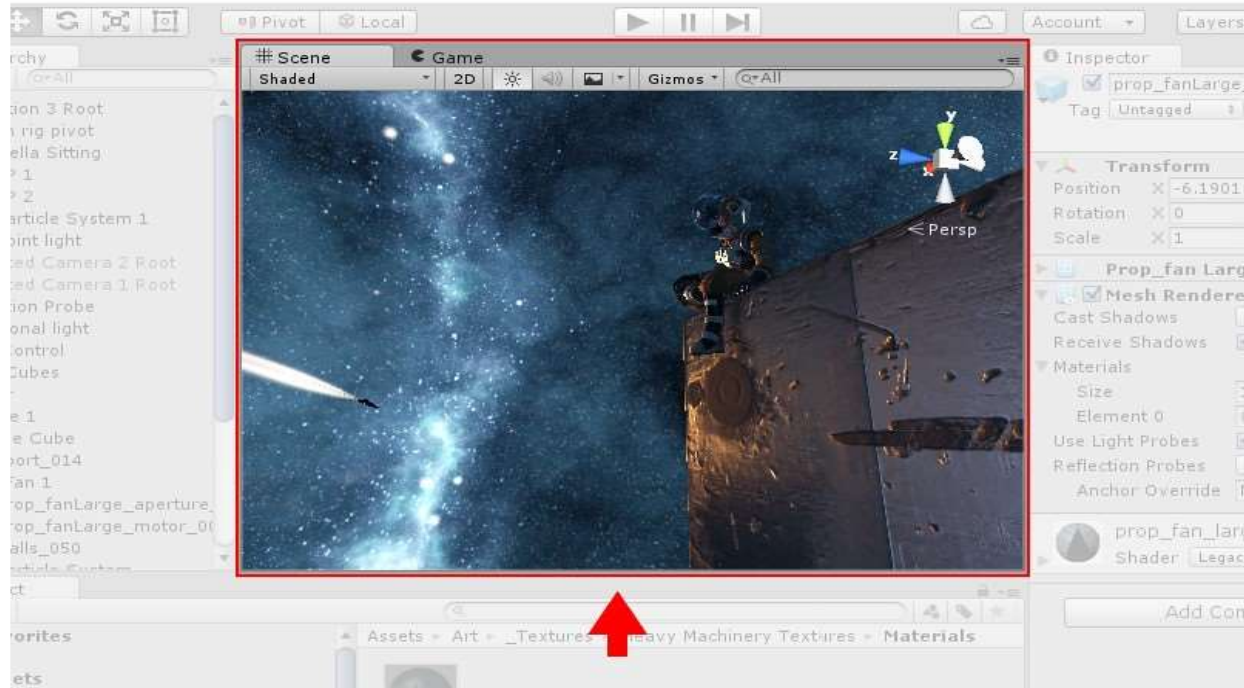


Figure [3.5]: *Scene window.*

The Scene Gizmo ([see Figure \[3.6\]](#)) is in the upper-right corner of the Scene View. This displays the Scene View Camera's current orientation, and allows the user to quickly modify the viewing angle and projection mode.

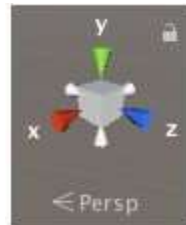


Figure [3.6]: *Scene Gizmo.*

In order to Move, Rotate, Scale, or Transform individual GameObjects, the user can use the four Transform tools in the toolbar ([see Figure \[3.7\]](#)). Each has a corresponding Gizmo that appears around the selected GameObject in the Scene view. To alter the Transform component of the GameObject, the user can use the

Chapter 3 – Utilized Software

mouse to manipulate any Gizmo axis, or type values directly into the number fields of the Transform component in the Inspector.

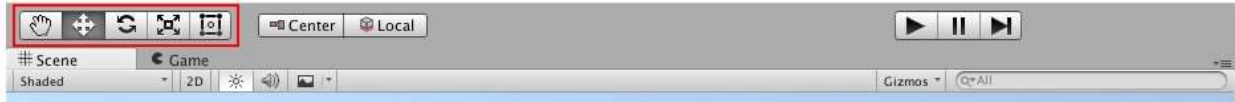


Figure [3.7]: *Transform tools.*

The Scene view control bar (see [Figure \[3.8\]](#)) provides the user with the opportunity to select between various options for viewing the Scene and also control whether lighting and audio are enabled. These controls only affect the Scene view during development and have no effect on the built game.



Figure [3.8]: *Control Bar.*

3.1.5. Game

The Game window (see [Figure \[3.9\]](#)) is rendered from the Camera in user's game. It is representative of the final, published game. It is required for the user to use one or more Cameras to control what the player actually sees when they are playing the game.

Chapter 3 – Utilized Software



Figure [3.9]: *Game window.*

3.1.6. Inspector

Projects in the Unity Editor are made up of multiple GameObjects that contain scripts, sounds, meshes, and other graphical elements such as lights. The Inspector window (sometimes referred to as “the Inspector”) displays detailed information about the currently selected GameObject (see [Figure \[3.10\]](#)), including all attached components and their properties, and allows the user to modify the functionality of GameObjects in the Scene.

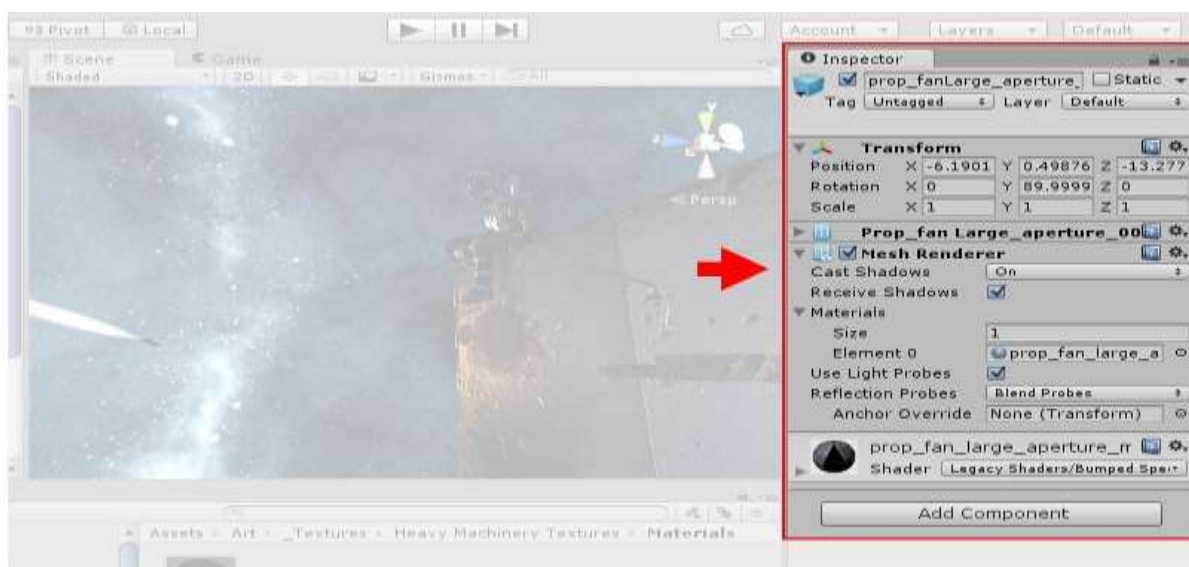


Figure [3.10]: *Inspector window.*

Chapter 3 – Utilized Software

The user can use the Inspector to view and edit the properties and settings of almost everything in the Unity editor, including physical game items such as GameObjects, assets, and materials, as well as in-editor settings and preferences. When a GameObject is selected in either the Hierarchy or Scene view, the Inspector shows the properties of all components and materials of that GameObject. Actually, the Inspector can be used to edit the settings of these components and materials.

3.1.7. Scripting

Scripting is an essential part of Unity as it defines the entire behavior of the game or application ([see Figure \[3.11\]](#)). Even the simplest game needs a script to respond to user's input. Scripts can be used for several reasons such as: to create graphical effects, to control physical behavior of objects or characters, to trigger effects upon specified conditions or even implement a custom AI system for characters in the game.



Figure [3.11]: C# Script.

Chapter 3 – Utilized Software

The behavior of GameObjects is controlled by the Components that are attached to them. Although Unity's built-in Components can be very versatile, the programmer will soon find he needs to go beyond what they can provide to implement his own gameplay features. Unity allows the user to create his own Components using scripts. These allow him to trigger game events, modify Component properties over time and respond to user input in any way he could probably want to.

Unity supports two programming languages natively, the **C#**, an object oriented programming language similar to Java or C++ and the **UnityScript**, a language designed specifically for use with Unity and modelled after JavaScript. The scripts can be written and edited in **MonoDevelop**, which is an integrated development environment (IDE) within Unity or in any other IDE like **Visual Studio**. An IDE combines a text editor with additional features for debugging, auto-complete and other project management tasks.

A script makes its connection with the internal workings of Unity by implementing a class which derives from the built-in class called **MonoBehavior**. The reader can think of a class as a kind of blueprint for creating a new Component type that can be attached to GameObjects. Each time the programmer attaches a script component to a GameObject, it creates a new instance of the object defined by the blueprint. The name of the class is taken from the name that the programmer supplied when the file was created. The class name and file name must be the same to enable the script component to be attached to a GameObject.

Thus, components can be accessed or modified by script at any time to achieve desired behavior and functionality. When a script is created, there are two functions automatically declared in it, the **Start** function and the **Update** function. The **Update** function is the right place to write code that will handle the frame update for the GameObject. This might include movement, triggering actions and responding to user input, basically anything that needs to be handled over time during gameplay. To enable the **Update** function to do its work, it is often useful to be able to set up variables, read preferences and make connections with other GameObjects before any game action takes place. The **Start** function will be called by Unity when a script is enabled and will be called exactly once in its lifetime. The **Start** function is the ideal place where initialization occurs. It is used to initialize an object's position, state and properties or load other scripts and GameObjects for later use.

Chapter 3 – Utilized Software

A script in Unity is not like the traditional idea of a program where the code runs continuously in a loop until it completes its task. Instead, Unity passes control to a script intermittently by calling certain functions that are declared within it. Once a function has finished executing, control is passed back to Unity. These functions are known as event functions since they are activated by Unity in response to events that occur during gameplay. Unity uses a naming scheme to identify which function to call for a particular event. For instance, we have already mentioned the Update function (called before a frame update occurs) and the Start function (called just before the object's first frame update). Many more event functions are available in Unity; the following are some of the most common and important events ([see Figure \[3.12\]](#)).

- **Regular Update Events:** These events can make changes to position, state and behavior of objects in the game just before each frame is rendered. The Update function is the main place for this kind of code in Unity. Update is called before the frame is rendered and also before animations are calculated. For physics update, like adding force to a GameObject, the best option is to place the code in the FixedUpdate function which updates more frequently than the Update function. Sometimes the best place to write code is the LateUpdate function in order to be able to make additional changes at a point after the Update and FixedUpdate functions have been called for all objects in the scene and after all animations have been calculated.
- **Initialization Events:** It is often useful to be able to call initialization code in advance of any updates that occur during gameplay. The Start function is called before the first frame or physics update on an object. The Awake function is called for each object in the scene at the time when the scene loads. Note that although the various objects' Start and Awake functions are called in arbitrary order, all the Awakes will have finished before the first Start is called. This means that code in a Start function can make use of other initializations previously carried out in the awake phase.
- **GUI Events:** Unity has a system for rendering GUI controls over the main action in the scene and responding to clicks on these controls. This code is handled somewhat differently from the normal frame update and so it should be placed in the OnGUI function, which will be called periodically. For instance, a set of OnMouseXXX event functions (e.g., OnMouseOver, OnMouseDown) is available to allow a script to react to user actions with the mouse. For example, if the mouse button is pressed while the pointer is over

Chapter 3 – Utilized Software

a particular object then an OnMouseDown function in that object's script will be called if it exists.

- **Physic Events:** The physics engine will report collisions against an object by calling event functions on that object's script. The OnCollisionEnter, OnCollisionStay and OnCollisionExit functions will be called as contact is made, held and broken. The corresponding OnTriggerEnter, OnTriggerStay and OnTriggerExit functions will be called when the object's collider is configured as a Trigger (i.e., a collider that simply detects when something enters it rather than reacting physically). These functions may be called several times in succession if more than one contact is detected during the physics update and so a parameter is passed to the function giving details of the collision (position, identity of the incoming object, etc.).

```
void Update() {  
    float distance = speed * Time.deltaTime * Input.GetAxis("Horizontal");  
    transform.Translate(Vector3.right * distance);  
}  
  
void OnGUI() {  
    GUI.Label(labelRect, "Game Over");  
}  
  
void OnCollisionEnter(otherObj: Collision) {  
    if (otherObj.tag == "Arrow") {  
        ApplyDamage(10);  
    }  
}
```

Figure [3.12]: *Update - GUI – Physical Events.*

Except for the functions that Unity provides, the developer can create his/her own functions in order to control or determine the behavior of a GameObject, change the properties of a component or altering the overall state of the application. In order for these custom functions to be executed, they have to be called inside a Unity event function, like the Update. The most commonly used functions were presented briefly above, as well as the concept of how they are used. The basic notion of the Unity scripting is that the scripts are components that can control the GameObject. Each component property corresponds to a script variable and the scripts can access not only the components of the GameObjects they are attached to, but also other GameObjects.

Chapter 3 – Utilized Software

3.2. Modeling

For the purposes of this project, two open suite 3D modeling tools have been used in order to create or modify some game objects. These tools are Blender and Sketch Up. The 3D modeling programs Blender and Sketch Up both allow three dimensional objects to be created, manipulated, and displayed. How these applications perform these tasks, along with costs associated with purchasing these tools, and best uses of these programs vary drastically. By knowing some of the strengths and weakness of these applications, the readers of this thesis may see why it was required to use both of them.

3.2.1. Blender

Blender is a free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. Blender is well suited to individuals and small studios who benefit from its unified pipeline and responsive development process. Examples from many Blender-based projects are available on the Internet. Blender is cross-platform and runs equally well on Linux, Windows, and Macintosh computers. Its interface uses OpenGL to provide a consistent experience.

The Blender 3D modeling program consists of five editor windows that are displayed as identified [in Figure \[3.13\]](#). Each of these windows contain numerous options, tabs, buttons, and pull-down menus. Initially all of this seems overwhelming. However, as the users get used to Blender, they find this graphic user interface providing an efficient approach for creating and animating 3D models.

Chapter 3 – Utilized Software

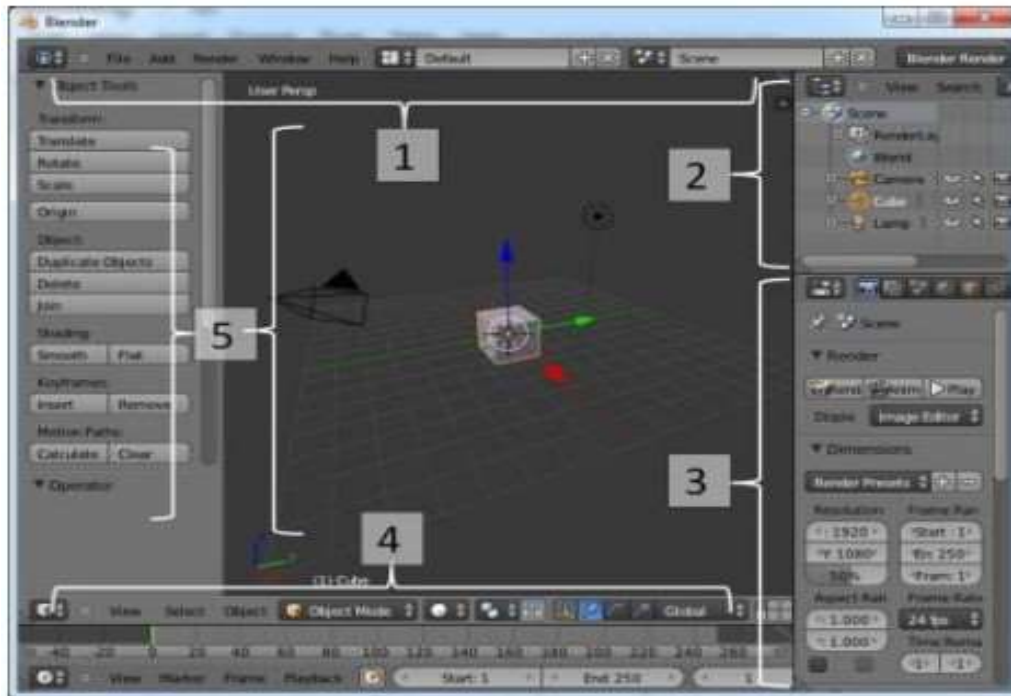


Figure [3.13]: *Blender software.*

Below is a brief description of the five editor windows.

1. Information editor - contains pull down menus, information on the scene, and objects selected.
2. Outline editor - contains an outline of the various elements in the scene.
3. Properties editor - controls the scene, rendering, lighting, and texturing.
4. Timeline editor - contains playback controls, a running timeline, and displays the position of the playback head.
5. Viewport editor window - is where the user can create, select, manipulate, transform, and view 3D objects.

3.2.2. SketchUp

SketchUp, formerly Google SketchUp, is a 3D modeling computer program for a wide range of drawing applications such as architectural, interior design, landscape architecture, civil and mechanical engineering, film and video game design. It is available as a freeware version, SketchUp Make, and a paid version with

Chapter 3 – Utilized Software

additional functionality, SketchUp Pro. Its major advantage is that the learning curve of Google SketchUp is very small and most users can learn how to operate the program within hours of first using. Google SketchUp is open source, meaning that it is a platform is open for programmers to develop plugins for this software. With the open source comes an immense amount of plugins available for the software.

When a user first opens SketchUp 3D modeling program, he will find a very simple interface is initially displayed as shown [in Figure \[3.14\]](#). Many additional interface elements can be displayed, but are not initially visible when the application is first loaded. Examples of other windows that can be displayed include: the scene window, the layer window, the styles window, the components window, the materials window, etc. Below is a brief description of the elements that are displayed when the application is first opened.

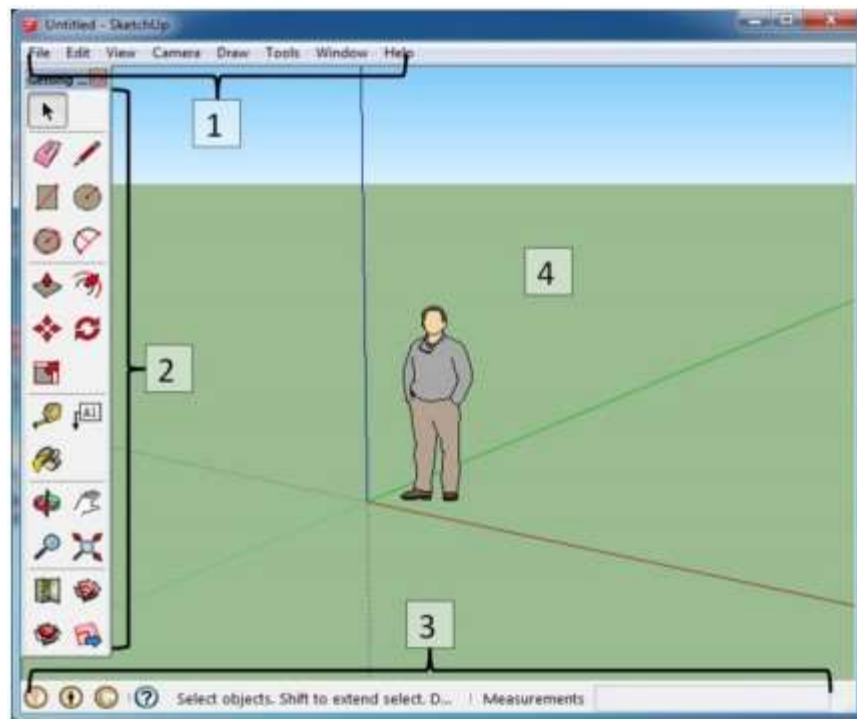


Figure [3.14]: *SketchUp software.*

1. Menu bar – contains a long list of options, commands, and settings that allow the user to perform numerous tasks.
2. Getting started toolbar – contains buttons that the user can use to active commands and tools.

Chapter 3 – Utilized Software

3. Status bar – contains contextual information and dimensions that the user can use while modeling.
4. Modeling window – is where a user can create, select, manipulate, transform, and view 3D objects.

3.3. MySQL Database

For the purposes of this project, a MySQL Database has been developed from scratch using the tools: PhpMyAdmin and XAMPP. These tools have been selected for their ease of use, the free software that they provide and the large community support that both of them offer. A brief general description of these tools follow.

3.3.1. PhpMyAdmin

PhpMyAdmin is a free software tool written in PHP, intended to handle the administration of MySQL over the Web. PhpMyAdmin supports a wide range of operations on MySQL. Frequently used operations (managing databases, tables, columns, relations, indexes, users, permissions, etc.) can be performed via the user interface, while the user still has the ability to directly execute any SQL statement.

Features provided by the program include:

- Intuitive web interface
- Support for most MySQL features, such as:
 - Browse and drop databases, tables, views, fields and indexes
 - Create, copy, drop, rename and alter databases, tables, fields and indexes
 - Maintenance server, databases and tables, with proposals on server configuration
 - Execute, edit and bookmark any SQL-statement, even batch-queries
 - Manage MySQL user accounts and privileges
 - Manage stored procedures and triggers
- Import data from CSV and SQL

Chapter 3 – Utilized Software

- Export data to various formats: CSV, SQL, XML, PDF, ISO/IEC 26300 - OpenDocument Text and Spreadsheet, Word, LATEX and others
- Administering multiple servers
- Creating graphics of the database layout in various formats
- Creating complex queries using Query-by-example (QBE)
- Searching globally in a database or a subset of it
- Transforming stored data into any format using a set of predefined functions, like displaying BLOB-data as image or download-link

3.3.2. XAMPP

XAMPP is an open-source web server package that works on various platforms. It is actually an acronym with X meaning “cross” platform, A for Apache HTTP server, M for MySQL, P for PHP, and P for Perl. XAMPP was designed to help webpage developers, programmers, and designers check and review their work using their computers even without connection to the web or internet. So, basically XAMPP may be used to stand as pages for the internet even without connection to it. It can also be used to create and configure with databases written in MySQL and/or SQLite. And since XAMPP is designed as a cross-platform server package, it is available for a variety of operating systems and platforms like Microsoft Windows, Mac OS X, Linux, and Solaris.

In order to use XAMPP, only one zip, exe or tar file is needed. Users just need to download this file and run the application. There is also not much configuration and tinkering to be done in terms of settings and its components. The XAMPP package is also updated on a regular basis to synchronize with the updates made on the different platforms involved in the package like Apache, PHP, Perl, and MySQL.

Aside from being cross-platform, XAMPP is also a freeware. This means users on different operating systems can download this server package free of charge.

Chapter 3 – Utilized Software

3.4. Web Development

In order to project the stored data to the experimenter and allow him to search for any subject, it was obligatory to develop a web interface. For the purposes of this interface, HTML has been used as well as the JavaScript language in combination with CSS and PHP language. These technologies are explained below.

3.4.1. HTML

HTML (Hypertext Markup Language) is the set of markup symbols or codes inserted in a file intended for display on a World Wide Web browser page. The markup tells the Web browser how to display a Web page's words and images for the user. Each individual markup code is referred to as an element (but many people also refer to it as a tag). Some elements come in pairs that indicate when some display effect is to begin and when it is to end.

HTML is a formal Recommendation by the World Wide Web Consortium (W3C) and is generally adhered to by the major browsers. The current version of HTML is HTML 5.1.

3.4.2. CSS

A cascading style sheet (CSS) is a Web page derived from multiple sources with a defined order of precedence where the definitions of any style element conflict. The Cascading Style Sheet, level 1 (CSS1) recommendation from the World Wide Web Consortium (W3C), which is implemented in the latest versions of the Netscape and Microsoft Web browsers, specifies the possible style sheets or statements that may determine how a given element is presented in a Web page.

CSS gives more control over the appearance of a Web page to the page creator than to the browser designer or the viewer. In general, the Web page creator's style sheet takes precedence, but it's recommended that browsers provide ways for the viewer to override the style attributes in some respects. Since it's likely that different

Chapter 3 – Utilized Software

browsers will select to implement CSS1 somewhat differently, the Web page creator must test the page with different browsers.

3.4.3. JavaScript

JavaScript is a programming language commonly used in web development. It was originally developed by Netscape as a means to add dynamic and interactive elements to websites. While JavaScript is influenced by Java, the syntax is more similar to C and is based on ECMAScript, a scripting language developed by Sun Microsystems.

JavaScript is a client-side scripting language, which means the source code is processed by the client's web browser rather than on the web server. This means JavaScript functions can run after a webpage has loaded without communicating with the server. For example, a JavaScript function may check a web form before it is submitted to make sure all the required fields have been filled out. The JavaScript code can produce an error message before any information is actually transmitted to the server.

Like server-side scripting languages, such as PHP and ASP, JavaScript code can be inserted anywhere within the HTML of a webpage. However, only the output of server-side code is displayed in the HTML, while JavaScript code remains fully visible in the source of the webpage. It can also be referenced in a separate .JS file, which may also can be viewed in a browser.

3.4.4. PHP

Stands for "Hypertext Preprocessor." (It is a recursive acronym). PHP is an HTML-embedded Web scripting language. This means PHP code can be inserted into the HTML of a Web page. When a PHP page is accessed, the PHP code is read or "parsed" by the server the page resides on. The output from the PHP functions on the page are typically returned as HTML code, which can be read by the browser. Because the PHP code is transformed into HTML before the page is loaded, users

Chapter 3 – Utilized Software

cannot view the PHP code on a page. This make PHP pages secure enough to access databases and other secure information.

A lot of the syntax of PHP is borrowed from other languages such as C, Java and Perl. However, PHP has a number of unique features and specific functions as well. The goal of the language is to allow Web developers to write dynamically generated pages quickly and easily. PHP is also great for creating database-driven Web sites.

4. Chapter 4 – Implementation

After having described every essential theory, as well as the software that has been used to develop the application and generally the background of this thesis, it's about time to explain in detail the implementation of this project.

The goal of this project was to develop a realistic 3D car driving simulator to be used for neuroscientific experimentation, in collaboration with the Brighton and Sussex Medical School. User's pulse rate was being monitored with a pulse oximeter fully synchronized with an ECG, while driving the virtual car for about 15 minutes. A photorealistic driving simulator has been implemented, displayed on an Oculus Head Mounted Display, controlled by an actual wheel interface and pedals. The software is programmed in Unity and communicates a realistic experience of a driving a car on an endless road in nature. A steering wheel as well as controlling pedals have been integrated in the application and have been calibrated in terms of the efficiency and sensitivity in order to achieve a simulation environment which is as close as possible to realistic driving.

A highly accurate method of monitoring the heart rate of individuals is put forward while users are driving. An oximeter attached to the earlobe was integrated in the application and was synchronized at 95% accuracy with an Electrocardiogram attached to the chest. Cardiac timing detection provides input to the driving simulator, which in turn, guides the occurrences of threat-inducing events while a user is immersed in a synthetic environment. These events appear at distinct points of the heart rate, either at cardiac systole or at cardiac diastole. It is important to determine if the reactions of the users are induced faster at cardiac systole when the

Chapter 4 – Implementation

baroreceptors are fired, or at cardiac diastole when the baroreceptors are quiet. These occurrences of threat-inducing events consist of: living obstacles such as horses, human-beings, and non-living obstacles, such as barrels. For each user several parameters are saved, for instance: their reaction times for each obstacle, their success in avoiding the obstacle or not, their action to avoid the obstacle etc. After the last phase of the experiments, the gathered data will be investigated and the appropriate conclusions will be drawn.

4.1. UI implementation

The experiments will be conducted by neuroscientists, so the development of a simple and understandable User Interface (UI) to set up every experiment was necessary. Taking into consideration several guidelines of HCI (Human Computer Interaction) such as the semantics of the colors, the ideal positioning of the interface elements as well as the ability to control the application either by pausing or exiting from it, was a significant part of developing an exceptional UI. According to HCI guidelines, the users of an Interface should be informed about the stage in which they are and not confront unexpected results.

The first image below ([see Figure \[4.1\]](#)) shows the first screen that the experimenter will face when the application is launched. The first thing we see in the following image, is that a forest has been selected as a background. Except for the visual appeal of the image, the forest has been selected because it's a similar environment with the one that the users will be "transferred to" in the experiment. The position of the Exit button is different from the Settings - Start Experiment buttons in order to avoid confusion. Also, the Exit button has as a red background color, because it represents "danger" according to HCI guidelines. The color of the Start Experiment button is green, because it represents a positive idea in the HCI ecosystem. The color of the Settings button is grey, because it represents something of lesser importance, as there is no need for the experimenter to click this button, unless a parameter of the experiment should be changed.

In the next image ([see Figure \[4.2\]](#)) we can see the menu that comes up if the experimenter clicks on the Settings button. It is easy to recognize the same buttons from the previous screen considering they have the same layout. It is very important

Chapter 4 – Implementation

to be consistent in positioning and coloring between different screens according to HCI User Interface guidelines. The Settings button has taken on a darker shade because the experimenter should be informed that this button was clicked and a new button has been highlighted using a blue color to provide the experimenter with the ability to check the Database.



Figure [4.1]: *Home Screen.*

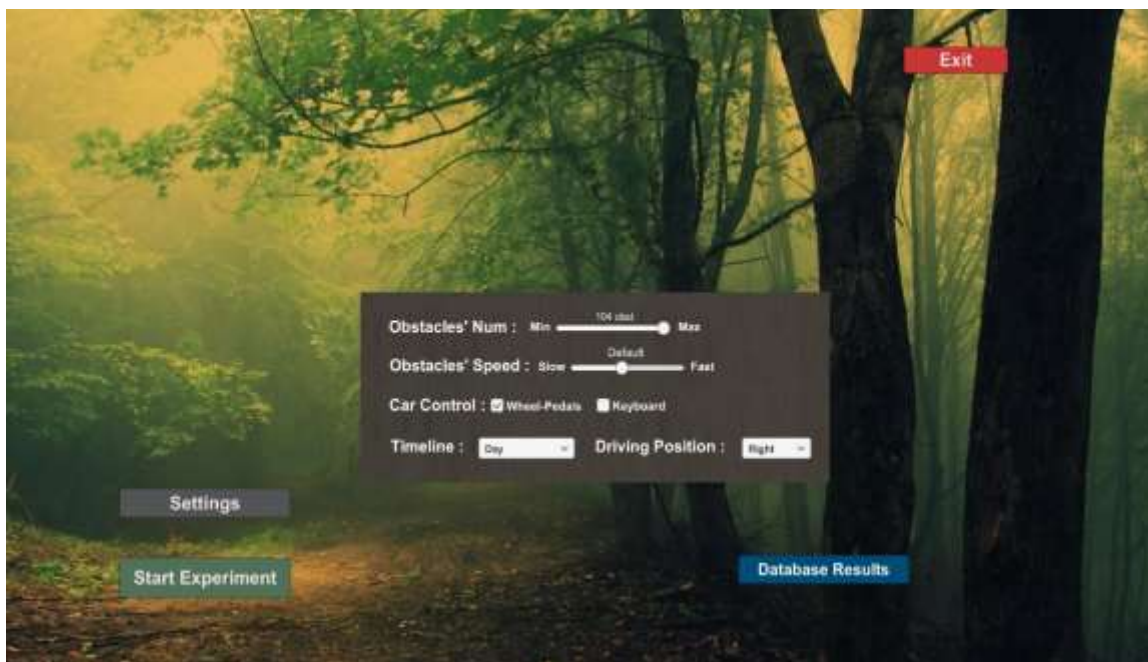


Figure [4.2]: *Settings Screen.*

Chapter 4 – Implementation

In the [Figure \[4.3\]](#) we can see the Settings Window in which the experimenter is able to adjust the settings of the experiment. The aim of this menu is to provide the experimenter with the opportunity to intuitively change any parameter of the experiment. There is not save button, as the changes that the experiments makes are automatically stored. The adjustable parameters are the following:

- Number of the obstacles - if the total obstacles amount changes, the length of the experiment changes as well.
- The speed of the obstacles - although a lot of research and testing for setting the optimal obstacles' speed has been conducted, the option for changing the speed still remains available.
- The option for controlling the car with a steering wheel or a keyboard.
- The timeline in which the virtual environment is going to be presented – available timelines are the day, the afternoon and the night; each timeline comes with different lighting options.
- Driving position of the car, because the subjects should feel comfortable in driving the car, as they are used to.



Figure [4.3]: *Settings window.*

Below, ([see Figure \[4.4\]](#)) we can see the form about the users' personal information. Their username, age and gender. It's highly important to keep the user informed about the errors that may occur in the process of filling the form, so in the yellow message we can see the message that the users will confront if they try to

Chapter 4 – Implementation

sign up with an already used username. The color of this message has been selected according the suggested colors for warnings of HCI.

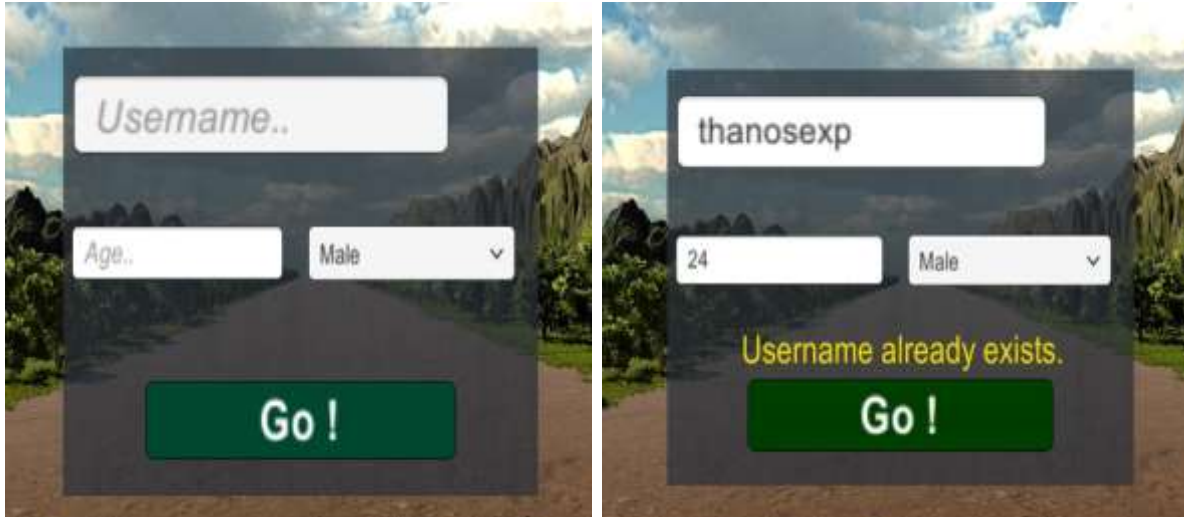


Figure [4.4]: *Sign Up window.*

An opportunity for pausing the experiment whenever the experimenter needs to is provided. The screen that appears when the application is Paused is the following ([see Figure \[4.5\]](#)).

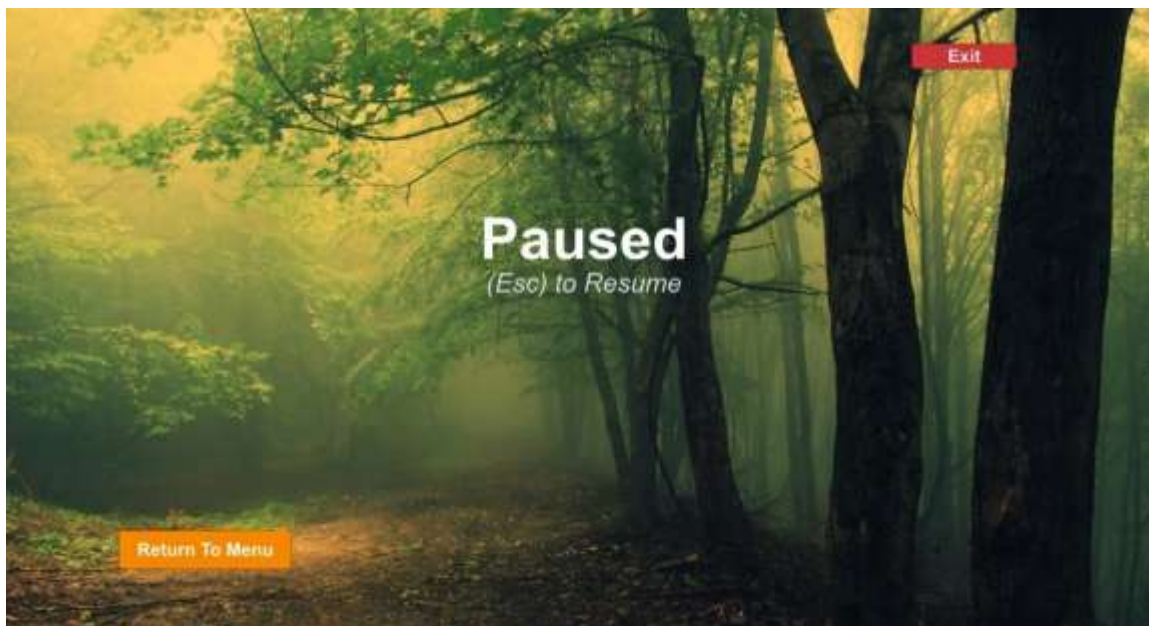


Figure [4.5]: *Pause window.*

Chapter 4 – Implementation

When the user faces the total number of obstacles that the experimenter has set, the experiment is finished. The image below ([see Figure \[4.6\]](#)), will appear on the screen of the experimenter.

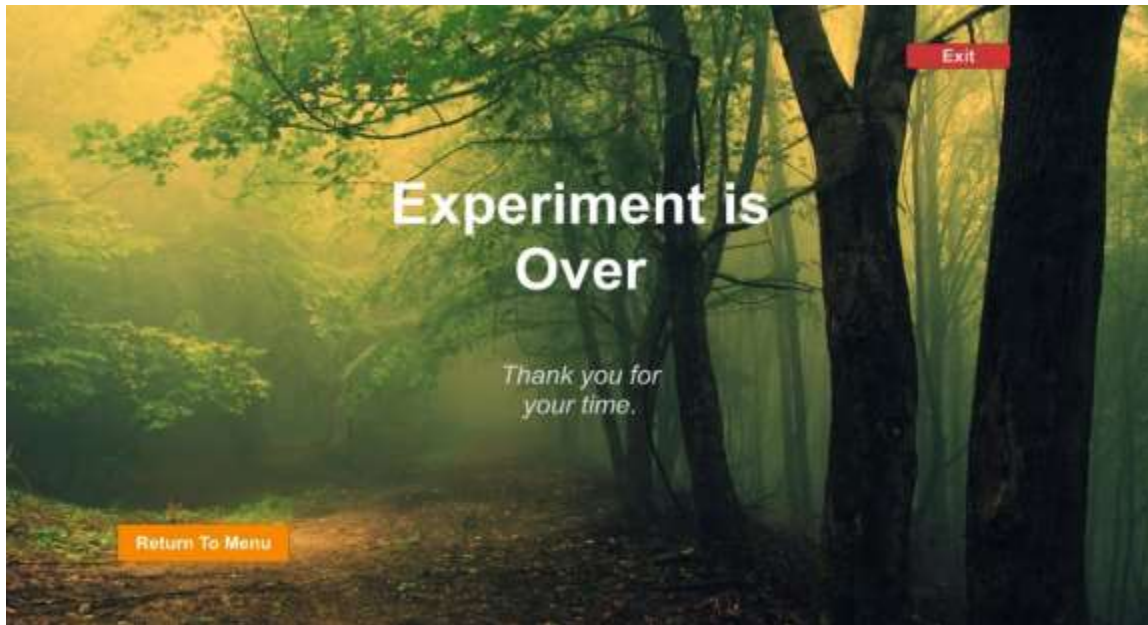


Figure [4.6]: *Experiment finished window.*

4.1.1. UI Scripting

After the development of 2D graphics in Unity Game Engine, a lot of scripting was required in order to make these UI scenes be functional. Five different scenes consist of the Homepage, the Settings page, the Pause page, the Game finished page and the main 3D scene of the experiment which also includes the sign in window. All these different pages had to be linked and controlled via scripting code using C# language.

Scripting code has been developed in order to switch from every UI scene to another by pressing the required button or key. In addition, code has also been written in order to read the data from the settings window in real time when the experimenter changes any option.

Chapter 4 – Implementation

4.2. 3D Virtual Scene

The 3D Virtual Scene of this project consists of the car, several obstacles and the environment. The car can either be for right driving position drivers (e.g. Great Britain drivers) or for left driving position drivers (e.g. most countries' drivers, such as: Greek, German etc.). The experiment takes place in a natural environment surrounded by mountains, trees and grass. The obstacles consist of 6 different categories and 5 different objects. The opportunity to select between a full experiment of 104 obstacles and a smaller experiment is given to the experimenter. During the experiment, the category and the kind of the object is selected randomly, but a logic has been implemented to secure that each category and kind of obstacles appears a specific amount of times.

In order to create, render, animate these objects a lot of steps are required. These steps will be further explained below.

4.2.1. Car modeling

For the purposes of the car modeling, different parts of different models merged together to create the final model. Most of the parts were adjusted in order to meet the needs of this project. Firstly, the car model of BMW X6 was selected. The color of the car is achieved by adding different materials in each part of it as well as different shaders wherever it was required. Also, spot lights added in the front side of the car to achieve the desirable lighting of a real car. Below ([see Figure \[4.7\]](#)) we can see the outside part of the car.

Chapter 4 – Implementation

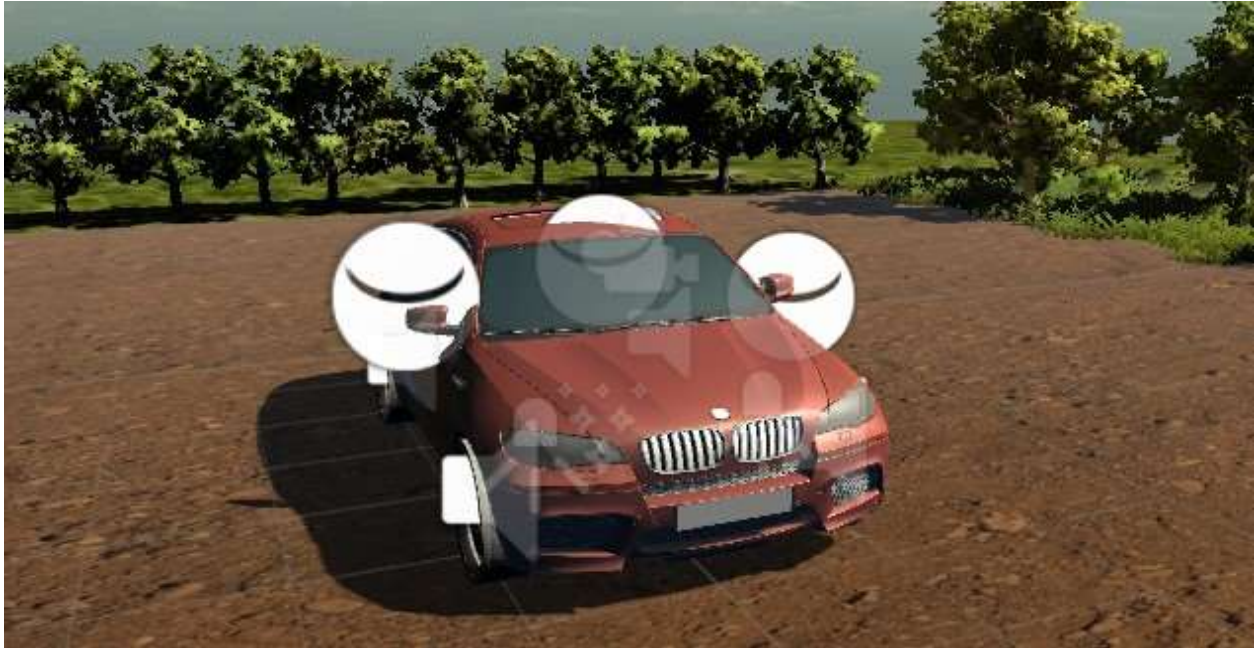


Figure [4.7]: *Outside car model.*

All the windows of the car are rendered in transparent mode so that the user will be able to see through them ([see Figure \[4.8\]](#)).



Figure [4.8]: *Car's interior - Transparent windows.*

Chapter 4 – Implementation

The whole interior part of the car was added in a 3d modeling software (in our case SketchUp) in order to adjust it to the project's needs. Afterwards, it had to be mirrored so we could cover both the left and the right car driving position ([see Figure \[4.9a\], Figure \[4.9b\]](#)).

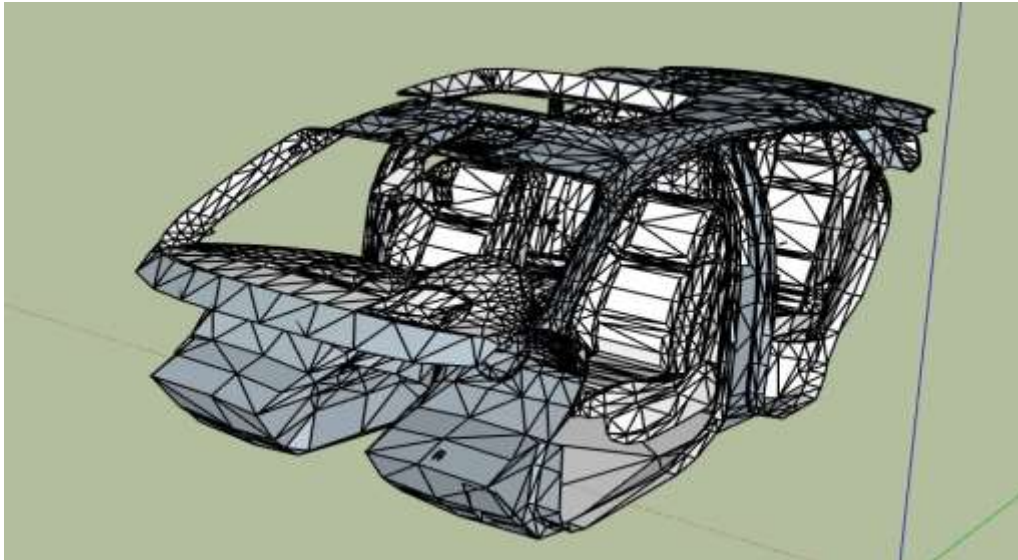


Figure [4.9a]: *Interior part of the car.*

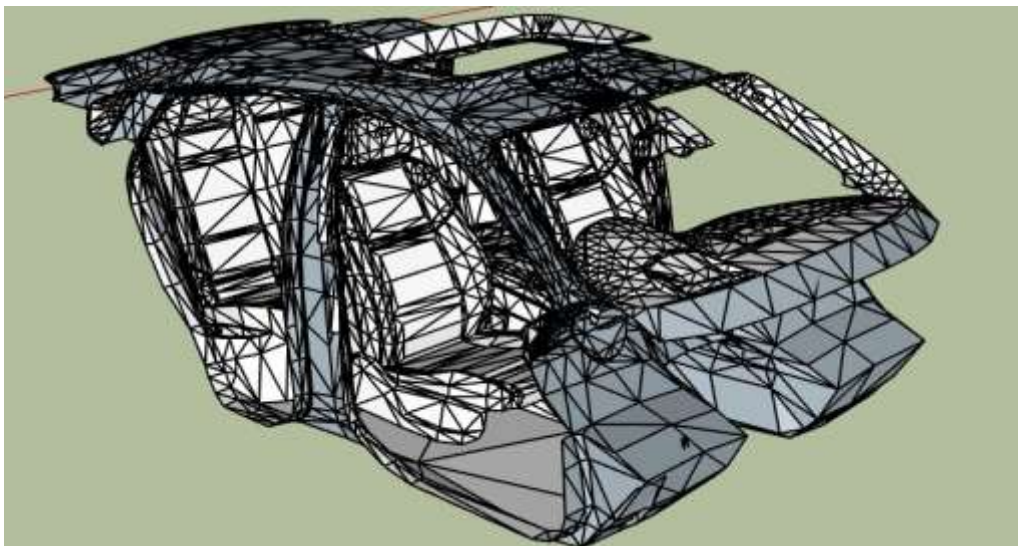


Figure [4.9b]: *Mirrored Interior part.*

The steering wheel had to be cut off and remodeled so that it could be animated for steering the car ([see Figure \[4.10\]](#)).

Chapter 4 – Implementation



Figure [4.10]: *Steering wheel.*

Also, the mirrors of the car modeled from scratch and rendered to reflect the proper background of the car. There were a couple of ways to create mirrors in Unity, but in our case, we decided to reflect the background as it is in reality. So, the subject can see different things by looking the mirror from different angles or heights ([see Figure \[4.11\]](#)).

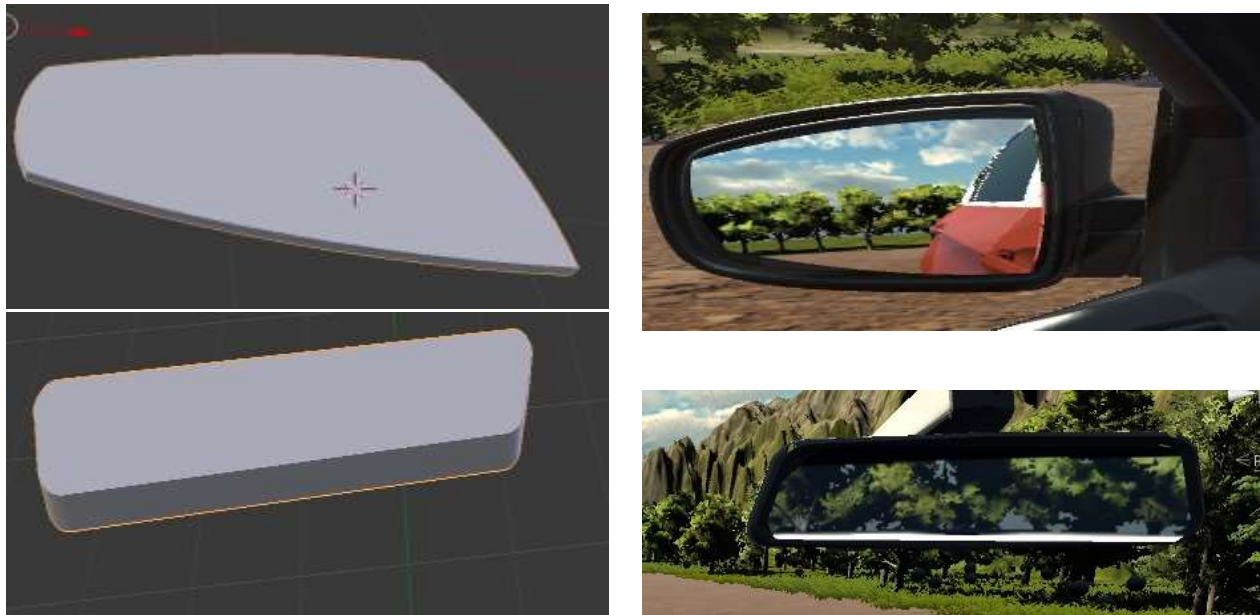


Figure [4.11]: *Models in sketch Up – Models in Unity.*

Chapter 4 – Implementation

In order to display the car's speed, a car dashboard image used and Photoshoped to achieve the desirable result. Afterwards, a needle was created from scratch in SketchUp software to show the actual speed during the driving. Below there are some pictures showing the process ([see Figure \[4.12\]](#), [Figure \[4.13\]](#)).



Figure [4.12]: *Photoshoped Car dashboard.*



Figure [4.13]: *Car needle.*

4.2.2. Environment modeling

The environment where the user is driving the car, consists of an endless road and two terrains on the side of the road as well as a terrain behind the car. In order to achieve the highest performance with the minimum computer latency, a specific optimization technique is used to reduce the graphics memory usage. This technique is applied by drawing only the part of the road that is in the visual field of the user

Chapter 4 – Implementation

and also improving the graphics that are near the user while worsening the graphics which are far from the user.

To achieve the first goal, only 300meters of road and terrain are being drawn by Unity based on the car's position. This means that anytime only a part of the whole map is drawn in order to optimize the CPU and GPU usage. To achieve the second goal, there have been created 2 different models for each side terrain. The first one which is nearer to the car has higher fidelity than the second one which is far from the car. When the car is moving, the lower fidelity terrain is being replaced with the higher one and the map that is behind the car is being deleted.

The images below ([see Figure \[4.14\]](#)), show the terrains (on the side of the road) that have high fidelity since they are closer to the car. All the terrains were designed from scratch. On this point, we cannot fail to mention that for the drawing of the high fidelity terrains, a Unity script tool ([see Figure \[4.15\]](#)) has been used which has a lot of settings for different brushes, adds noise on the ground and provides the option for adding different textures on the terrain floor. In addition, trees and grass were added only closer to the road as they require a lot of graphic memory to be rendered. Finally, after a lot of testing, the optimal parameters were selected to combine high performance with good graphics.

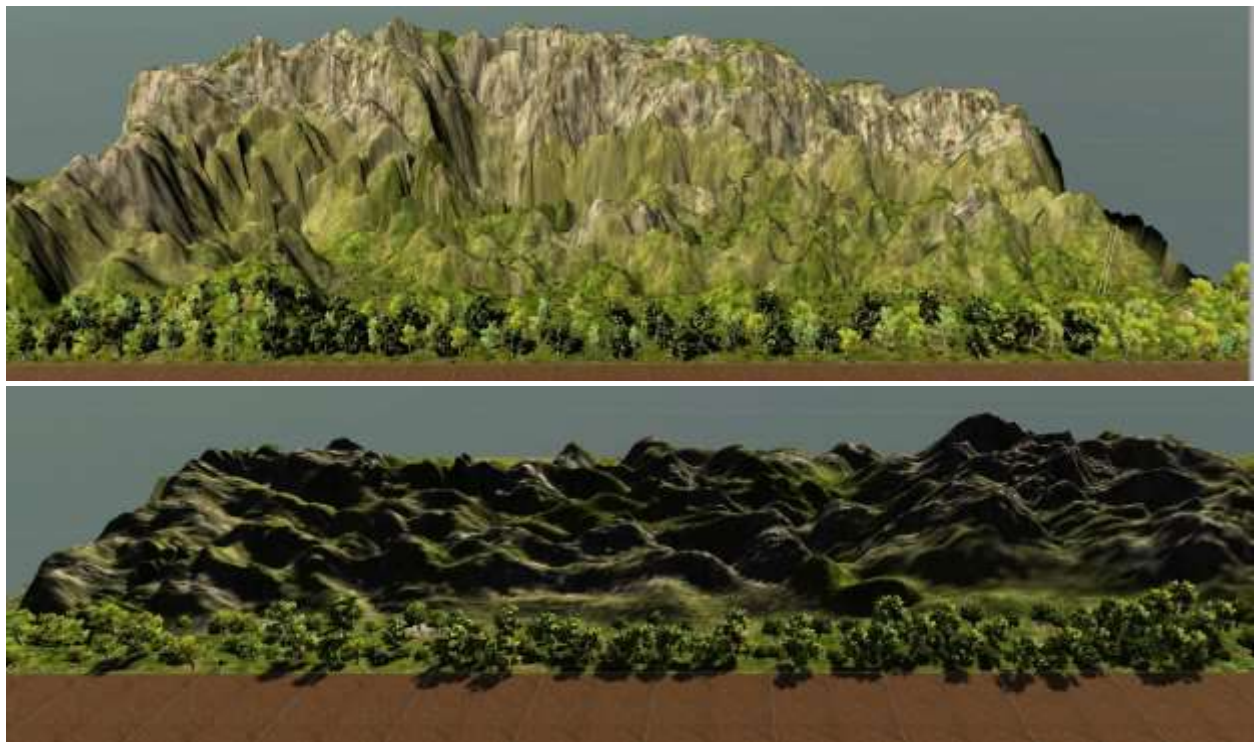


Figure [4.14]: *High Fidelity terrains.*

Chapter 4 – Implementation

```
//main brush params
public float brushSize = 10;
public float brushFallof = 0.6f;
public float brushSpacing = 0.15f;
public int downscale = 1;
public float blur = 0.1f;

public bool isErosion;
public bool isNoise { get{return !isErosion;} set{isErosion=!value;} }

//noise brush
public int noise_seed = 12345;
public float noise_amount = 20f;
public float noise_size = 200f;
public float noise_detail = 0.55f;
public float noise_uplift = 0.6f;
public float noise_ruffle = 1f;

//erosion brush
public int erosion_iterations = 3;
public float erosion_durability = 0.9f;
public int erosion_fluidityIterations = 3;
public float erosion_amount = 1f;
public float sediment_amount = 0.8f;
public float wind_amount = 0.75f;
public float erosion_smooth = 0.15f;

//painting
public SplatPreset foreground = new SplatPreset() { opacity=1 };
public SplatPreset background = new SplatPreset() { opacity=1 };
```

Figure [4.15]: *Terrain tool.*

The lower fidelity terrains ([see Figure \[4.16\]](#)), have been drawn completely by hand with the lowest possible resolution since they are far from the car and they are slightly visible. Fewer trees have been added and no grass at all, since it wouldn't be visible to the user from such a long distance. As it is explained above, the lower fidelity terrains are being replaced by the higher ones when the car approaches them. Then the low fidelity terrains are being redrawn in front of the higher ones again.

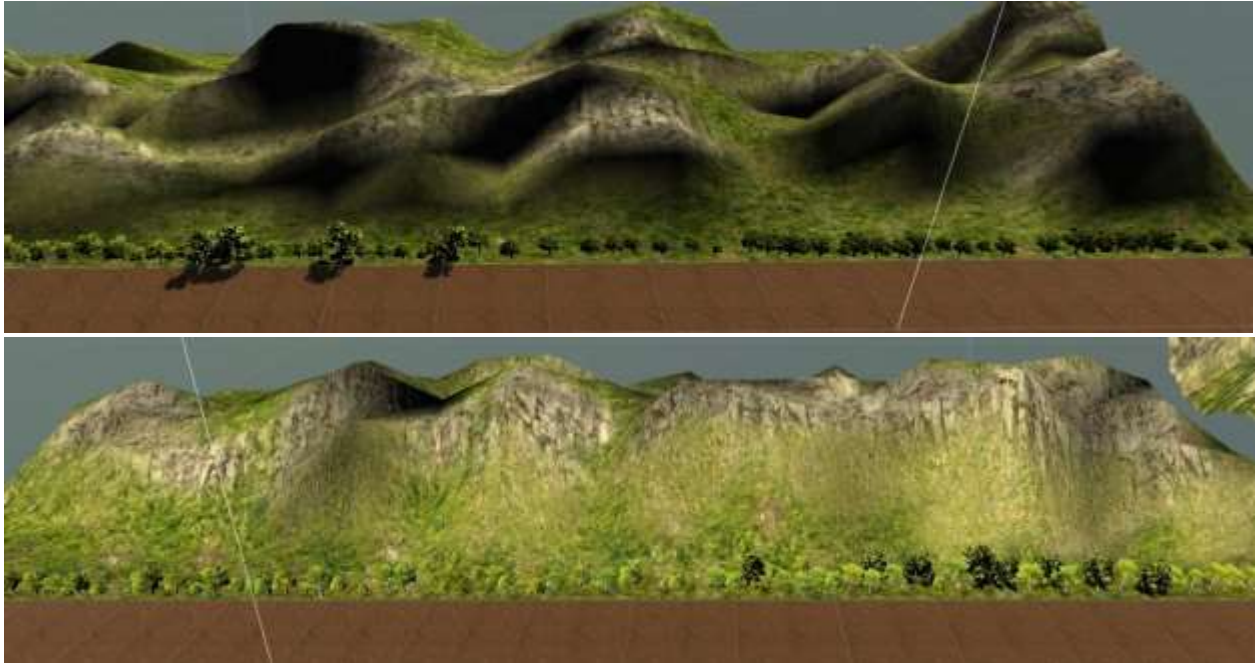


Figure [4.16]: *Low Fidelity terrains.*

4.2.3. Obstacles modeling

While the user is driving a car in an endless road, some obstacles appear and move towards him from different directions. Based on the pulse rate of the subject, obstacles appear either on diastole or at systole. In that way, when the data analysis will occur, we will be able to differentiate the obstacles that spawned at systole from those who spawned at diastole based on each category. Six different classes of obstacles are required for this project. These classes are presented in the following table ([see Figure \[4.17\]](#)).

Position:	Near	Near	Far	Far	Left	Right
Speed:	Fast	Slow	Fast	Slow	Standard	Standard

Figure [4.17]: *Obstacles Categories.*

For the first four categories (Near Fast - Near Slow – Far Fast – Far Slow), it was required to use a living and a non-living obstacle which would be randomly positioned in front of the car. So, the positioning of the object is being selected

Chapter 4 – Implementation

randomly and then each obstacle starts to move towards the car. These obstacles were programmed to move with either a faster or a slower speed. An important factor for choosing the right speed for the obstacles was to make sure that the collision would happen in less than a second for the near fast obstacle and in slightly more than a second for the far slow obstacle, if the user wouldn't steer, brake or accelerate. Also, we had to make sure that the obstacle is avoidable, to be more specific, the subject should be able to avoid the obstacles otherwise there wouldn't be any point in trying to do so. The obstacles that were selected were a horse ([see Figure \[4.18\]](#)) and two barrels ([see Figure \[4.19\]](#)).



Figure [4.18]: *Horse obstacle.*

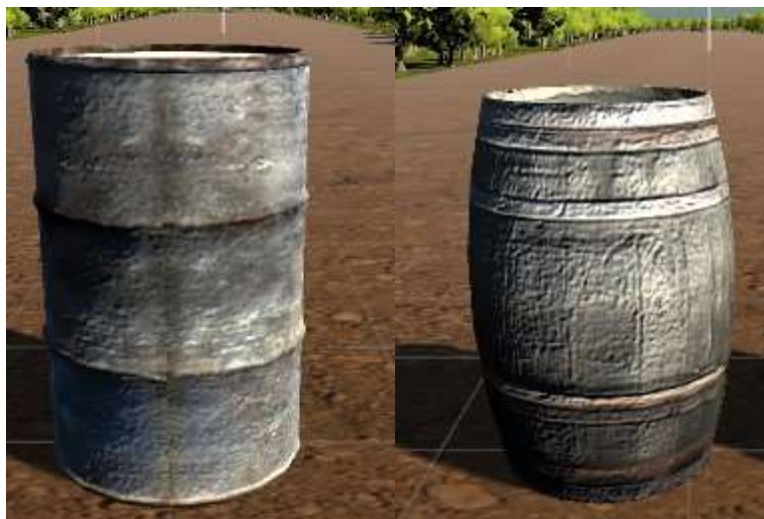


Figure [4.19]: *Barrels obstacles.*

Chapter 4 – Implementation

The obstacles that were positioned to the left and the right side of the road are an adult and a child. After these obstacles appear, they start moving to cross the road. These obstacles had to be positioned in a very specific distance from the car since we had to secure the collision if the subject wouldn't react. Every time an adult or a child appears in the side of the road, if the subject does not react by steering, accelerating or braking we had to be sure that he would collide with these obstacles. In order to secure the collision of the car with the moving obstacles crossing the road, we incorporated the speed of the car in the parameter which determines the distance from the car that these obstacles appear. This means that every time an adult or a child appears we look up for the speed of the car to position the obstacle in the right place. Below, we can see the models of the adult and the child ([see Figure \[4.20\]](#)).



Figure [4.20]: *Child – Adult obstacles.*

4.2.4. Obstacles animation

In order to make obstacles look more realistic, it was required to animate them. As it is explained in a previous chapter, animation is creating a sequence of images rapidly to make an illusion for the eyes. To begin with, in order to animate barrels, we created an animator in Unity which rotates in y-axis the barrel in loop mode. Below we can see the animator ([see Figure \[4.21\]](#)). The variables Rotation.x,

Chapter 4 – Implementation

Rotation.z remain the same during the time of animation and only the variable Rotation.y changes.

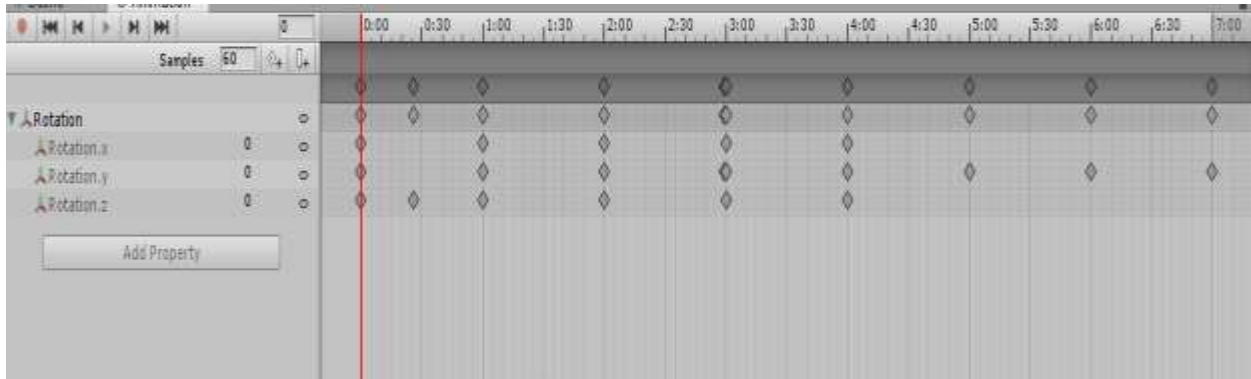


Figure [4.21]: *Barrel animator.*

Another object that had to be animated was the horse. The animation of the horse was a bit more complex because there were a lot of variables that had to be set up. Fortunately, Unity Game Engine offers a lot of animated game objects for free in the Asset Store. The steps for animating the horse are the following ([see Figure \[4.22\]](#)).

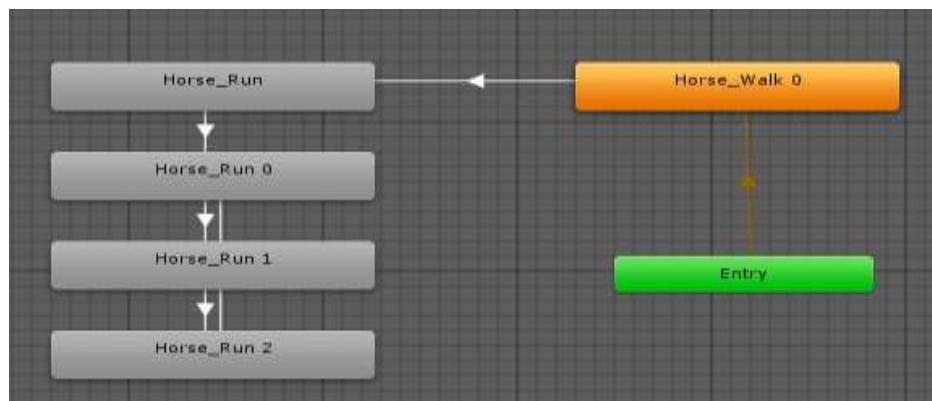


Figure [4.22]: *Steps for the Horse animation.*

We can clearly see that when the horse starts to be animated, it begins to walk and then it enters in loop where it constantly runs till we make it stop. Just to have an idea about the complexity of animation, below we present the variables that have

Chapter 4 – Implementation

to change in order to make a realistic animation ([see Figure \[4.23\]](#)). This complexity comes from the number of different parts of the horse that have to change while the horse is moving, such as the legs, the head, the tail etc.

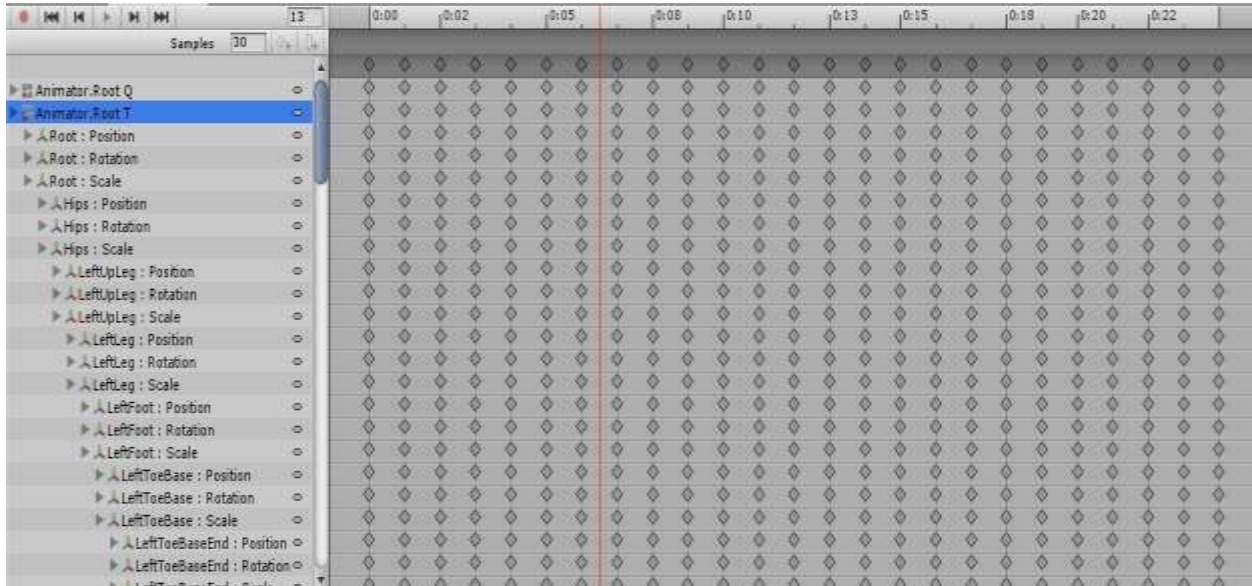


Figure [4.23]: *Horse animation.*

Below, we can see some images that show the animation of the horse ([see Figure \[4.24\]](#)).



Figure [4.24]: *Horse running.*

Furthermore, the obstacle of an adult was also required to be animated. While the adult was moving and crossing the road, his body was required to seem like a

Chapter 4 – Implementation

real walking human-being. In that way, the user that would do the experiment would feel more threatened and the image of a human would be more realistic.

In order to animate the adult model, the Animator below was used ([see Figure \[4.25\]](#)). In this image we can see the process of the adult's animation which is just a state of constantly walking. That means that when the adult appears in the virtual scene, he starts moving until we make him disappear from the scene. Because of the complexity in the animation of the human-being, an animator from Unity asset store has been used.

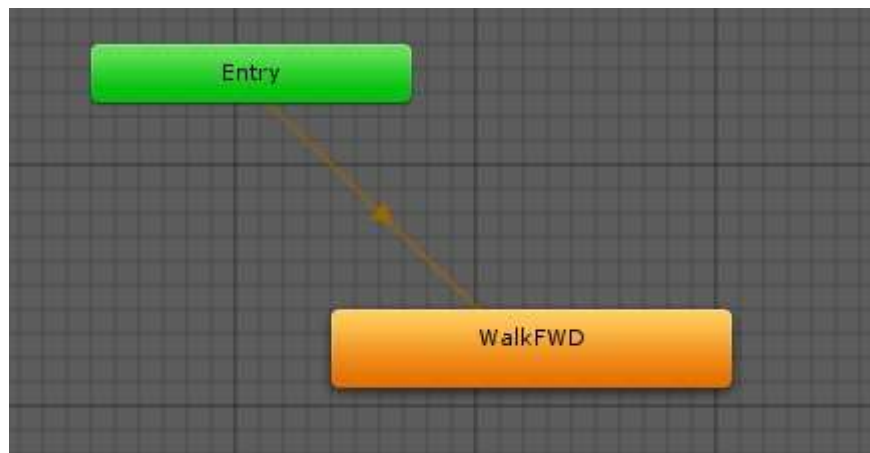


Figure [4.25]: *Human animation.*

The walking process of the adult can be seen in the following pictures, so the reader will have a more detailed idea about the scene that the users of the experiment are facing ([see Figure \[4.26\]](#)).



Figure [4.26]: *Adult walking.*

Chapter 4 – Implementation

Finally, the last object that was needed to be animated was the kid. In order to animate the model of the kid the same strategy with the adult has been followed. When the kid appears in the screen, he starts moving till we make him disappear. He is walking crossing the road, and his body is being animated ([see Figure \[4.27\]](#)).



Figure [4.27]: *Kid walking.*

4.2.5. Main Screen Scripting

From the beginning of the experiment, when the 3D scene of the application is launched, everything is being handled by programmed scripts. The car is being spawned automatically via scripting code. Also, the sky and the lighting of the scene is selected via scripting code based on the corresponding choice of the menu in the timeline option of the settings page. The terrains on the side of the road as well as the road that the car is being driven are being drawn automatically based on the car's position. Event handlers are also handling the case of pausing the application; whenever the experimenter presses the Esc button.

In addition, as we have already mentioned when the car is being spawned in the 3D virtual world, only 300 meters of the road and terrains are drawn. The first half of the terrains is presented in detail with high fidelity and the other half is presented with low fidelity for minimum computer latency. There is a handler for the map regarding the car's position, which checks in real time the position of the car and if it has reached the middle of the distance of the road (150m of 300m) then it removes the previous terrain and road and it draws the next 300m.

Chapter 4 – Implementation

Regarding the obstacles, there are a lot of scripts which handle every option for them, from the appearance to the animation. 22 different counters were needed to cover the variety of all the kinds of obstacles: with different distances from the car, different speeds and for both cardiac cycle distinct points (systole – diastole). Based on the requirements of neuroscientists, an obstacle appears every 5 to 7 seconds randomly either at systole or at diastole. We had to secure that each obstacle would appear a specific amount of times, so although in the beginning of the experiment the selection of the obstacles is random, when an obstacle's appearances reaches the maximum value, then automatically via scripting code we randomly select to spawn one from the remaining obstacles that has not reached the maximum number of appearances. Also via scripting code are selected: the timing that the animation of the obstacles starts and stops as well as the exact position of them which is randomly selected between a defined number range.

Moreover, the speed of the car, as well as the animation of the speedometers were programmed in order to be accurate based on the car's actual speed. Firstly, we presented the speed of the car in a digital form in the center of the car's speed indicator. Then, an object was used as a point for rotation in the center of the car's needle and the needle rotated around this object based on the car's speed. Afterwards, in order to animate the needle for the car revs a similar procedure was followed but instead of rotating the needle based on the car's speed it was rotated based on the car's revs.

4.3. Car Controller

Unity 5.0 or higher, comes along with a standard car controller that can be assigned into a car and with few adjustments on the code provides the opportunity to control the car. In the following image we can see the options that can be adjusted in the car controller ([see Figure \[4.28\]](#)).

Chapter 4 – Implementation

```
[SerializeField] private CarDriveType m_CarDriveType = CarDriveType.FourWheelDrive;
[SerializeField] private WheelCollider[] m_WheelColliders = new WheelCollider[4];
[SerializeField] private GameObject[] m_WheelMeshes = new GameObject[4];
[SerializeField] private WheelEffects[] m_WheelEffects = new WheelEffects[4];
[SerializeField] private Vector3 m_CentreOfMassOffset;
[SerializeField] private float m_MaximumSteerAngle;
[Range(0, 1)] [SerializeField] private float m_SteerHelper;
[Range(0, 1)] [SerializeField] private float m_TractionControl;
[SerializeField] private float m_FullTorqueOverAllWheels;
[SerializeField] private float m_ReverseTorque;
[SerializeField] private float m_MaxHandbrakeTorque;
[SerializeField] private float m_Downforce = 100f;
[SerializeField] private SpeedType m_SpeedType;
[SerializeField] private float m_Topspeed = 20;
[SerializeField] private static int NoOfGears = 5;
[SerializeField] private float m_RevRangeBoundary = 1f;
[SerializeField] private float m_SlipLimit;
[SerializeField] private float m_BrakeTorque;
```

Figure [4.28]: *Car controller.*

Unity game engine supports a lot of input devices such as: keyboard, joystick, steering wheel, gamepad etc. These devices are used to have better control of the game objects in the scene. In our case, the experimenter can either select to control the car by using a keyboard or a steering wheel – pedal from the starting menu.

4.3.1. Keyboard

In order to make the game playable through keyboard and mouse, Unity has already implemented the adjustments that were needed by default. In the following image we can see the code that checks the menu options to enable the keyboard or the steering wheel controller ([see Figure \[4.29\]](#)).

Unity has some default settings for the input controllers that can be changed either programmatically or from the User Interface menu. The only thing that was required to be adjusted is the use of mouse, in virtual reality. The virtual world consists of 3 dimensions, so the use of mouse is a bit confusing. In order to overcome this issue, it was required to turn off the virtual-reality for the Menu images of the application, because only there the use of mouse was required.

Chapter 4 – Implementation

```
public void wheel_pedals()
{
    if (Keyboard.isOn)//Values for keyboard
    {
        flag_keyboard = true;

        flag_wheel = false;
        wheel.isOn = false;
    }
    if (Wheel.isOn) //Pedals
    {
        flag_wheel = true;

        flag_keyboard = false;
        Keyboard.isOn = false;
    }
}
```

Figure [4.29]: *Controller selection.*

4.3.2. Steering Wheel - Pedals

For the purposes of this project the following product has been bought and used to control the car: Logitech Steering Wheel G29 ([see Figure \[4.30\]](#)). This product has been selected for its great quality, compatibility and ease of use. Logitech used leather and anodized aluminum materials for the steering wheel and cold rolled steel, brushed stainless steel as well as polyoxymethylene thermoplastic for the pedals.



Figure [4.30]: *Logitech G29 product.*

Chapter 4 – Implementation

Moreover, in order to be able to control the car through the steering wheel in Unity, only adjustments for the pedals was needed to be made since the steering wheel was already functional from the default settings of Unity. Regarding the pedals, the right axis had to be found from the Input manager of Unity and assigned to a variable in order to be visible from the Game Engine ([see Figure \[4.31\]](#)). The sensitivity of the pedals was adjusted as well as the function of each one of them.

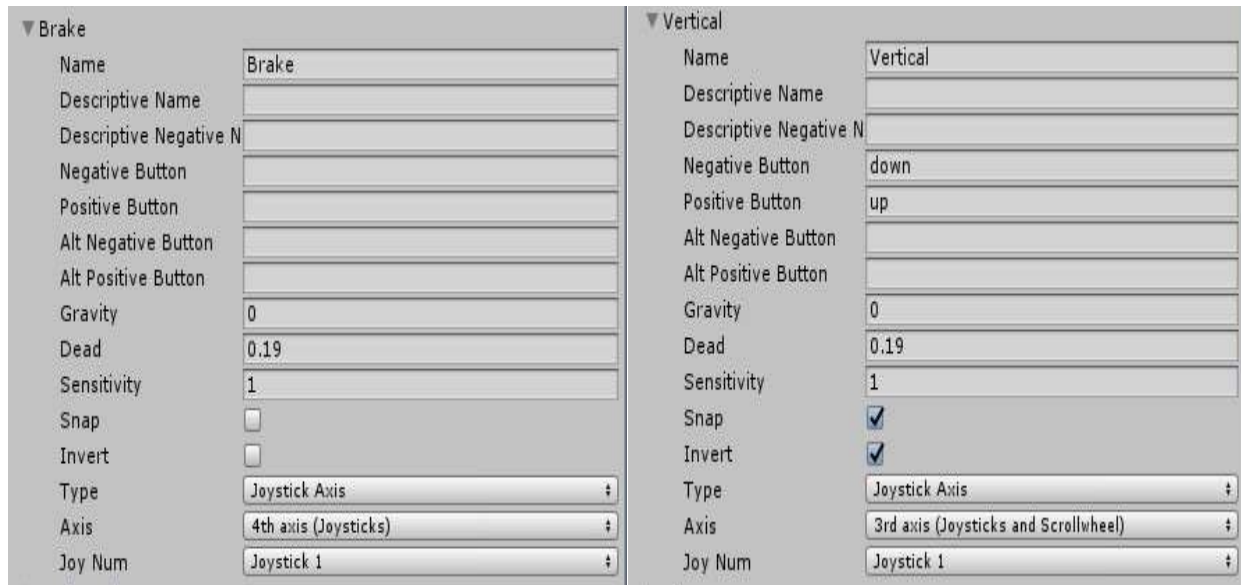


Figure [4.31]: *Unity Input Manager.*

4.4. Audio

“Sound design is what truly convinces the mind that is in a place; in other words, hearing is believing.”- The Art of Game Design, Jesse Schell. Based on the previous quote it is perfectly clear that a game without audio would be totally incomplete, it would lose the connection between the player and the environment. So we should all agree that the audio is valuable and necessary part of a game.

In real life, sounds are emitted by objects and heard by listeners. The way a sound is perceived depends on a number of factors. A listener can tell roughly which direction a sound is coming from and may also get some sense of its distance from its loudness and quality. A fast-moving sound source (like a falling bomb or a passing police car) will change in pitch as it moves as a result of the Doppler Effect.

Chapter 4 – Implementation

Also, the surroundings will affect the way sound is reflected, so a voice inside a cave will have an echo but the same voice in the open air will not.

To simulate the effects of position, Unity requires sounds to originate from Audio Sources attached to objects. The sounds emitted are then picked up by an Audio Listener attached to another object, most often the main camera. Unity can then simulate the effects of a source's distance and position from the listener object and play them to the user accordingly. The relative speed of the source and listener objects can also be used to simulate the Doppler Effect for added realism.

For the purposes of this project, the audio listener is attached in the main camera (inside the car) and several sounds are coming from objects such as: the car, the horse, the barrel and the human-beings.

These sounds have been downloaded from several repositories of the Internet and edited to fit in our needs. The format of the audio files is mp3 and the duration of them is really short.

4.4.1. Audio Scripting

In the application we have integrated some sound effects in order to make the environment more realistic and connect the user with the 3d environment. Each of these sounds are attached to specific game objects of the screen, for instance: to the car, the horse, the barrel, and the human beings.

To simulate the **car's audio**, a programmed script has been attached in the car that consists of a four-channel engine sound. One channel is for the simulation of the low acceleration sound, another is for the high acceleration sound, another for the low deceleration sound and one for the high deceleration sound based on the acceleration of the car. In addition, the script of the car controller checks also if the wheels of the car are sliding and if they do, another sound for sliding is being played.

To simulate the **horse** walking, a sound clip has been attached in the horse that is being played from the programmed scripts each time a horse appears in the environment. The same thing happens for the rolling **barrel**, a clip that simulates the metal sound of a rolling barrel is being played by a script each time a barrel appears in the environment.

Chapter 4 – Implementation

Finally, when the users collide with an obstacle, a short and realistic sound has been selected to be played from the script for half a second to make the user feel more frightened and closer to reality.

4.5. Oximeter integration

An accurate and reliable way to measure the heart rate of individuals is the pulse oximeters. Pulse oximeters are being widely used for non-invasive, simultaneous assessment of haemoglobin oxygen saturation. They are often used for estimating heart rate at rest and during exercise. In their most common (transmissive) application mode, a sensor device is placed on a thin part of the user's body, usually a fingertip or earlobe, or in the case of an infant, across a foot.

Finger clip pulse oximetry sensors may not perform well if the digit is poorly perfused or there is excessive hand movement, so in our case an ear probe has been selected instead of the finger. Probes are carefully designed, so that they can shine light through the finger or the ear and detect it on the other side of the body. They measure the changing absorbance at the wavelength, allowing it to determine the absorbance due to the pulsing arterial blood alone, excluding venous blood, skin, bone, muscle, fat, and (in most cases) nail polish.

For the project's needs, a NONIN XPOD pulse oximeter was used combined with an ear Clip Sensor 8000Q2. The ear clip sensor was required since the users will use their hands to control the steering wheel.

4.5.1. Connection with Unity

In order to connect the NONIN pulse oximeter with Unity, the basic idea of scripting mechanisms first implemented from a software engineer in the Brighton and Sussex Medical School in the context of another project. Afterwards the scripts were updated and modified to be functional in this project.

Three main scripts were implemented in order to connect the oximeter with Unity and provide feedback in the rest of our scripts about the prediction of the user's cardiac systole and diastole. Firstly, a function (HeartBeatOxi()) is searching for

Chapter 4 – Implementation

connected oximeters in the computer with specific characteristics, such as: a name of the port, the maximum bits that the port can transfer per second (9600b/s in our case), the data bits that the oximeter is transferring (8 in our case) etc. Once the connection is done with the oximeter, it is checked if all the received packets from it are correct based on the NONIN specification for the oximeter. For instance, the 7th bit should always be 1 and the sum of the first 4 bits should be a multiple of 256. If the received packets are correct, a function for detecting heart beats is called. This function is looking for heartbeats and once a heartbeat is detected, another function from another script is called (HeartBeat()).

In this function there is a counter which measures the time in milliseconds between each heart beat and it creates a list of the previous 5 heartbeats. Based on these heartbeats it predicts the next one based on the average value of them. Other mechanisms for prediction and a more detailed description is following in the next section. Then the third function is called (Feedback()) passing the value of the next heart beat prediction.

This function aims in counting the time from the millisecond that is called till the heart beat to occur. It is important to mention that we had to include the delay of the system in this measurement which was 350ms. Two calculations are made at this point, one for the systole that is the exact number that the previous function transferred to this function minus the delay of the system and one for diastole that is the number of systole minus 300ms. Afterwards, when the system predicts diastole or systole if it's also as a coincidence time to spawn an obstacle then it is being spawned, otherwise we proceed in calculating the next heart beat prediction.

4.5.2. Cardiac Beat prediction

The main goal of this project was to intensify if there are differences in the reaction times of users related to their cardiac rate. This means that the obstacles had to appear in specific times based on the cardiac rate of each subject. After connecting the pulse oximeter with Unity, we were able to determine when the heart is beating and measure the exact time between each heartbeat. This gave us the opportunity to apply a method in order to predict the next heartbeat.

There are a lot of methods that can be applied in order to predict the next heart beat based on a specific sample of heartbeat periods. One of the probably most

Chapter 4 – Implementation

obvious methods is to predict the next heartbeat based only on the previous one. This method has proved not to be the best one since there is a chance that the subject has a false heart beat or not a stable cardiac rate and something like this would affect our prediction. Also, in cases that the subject has faster heart rate from the usual the computer wouldn't have much time to predict the next beat. For instance, if someone is having 550ms heart rate period it means that the computer would have to make the calculations in about 250ms (for diastole) minus the system delay (350ms) which is impossible.

Another method could be to find the median of a sample of heart beat periods. The median is the value separating the higher half of a data sample, from the lower half. In simple terms, it may be thought of as the "middle" value of a data set. For example, in the data set {1, 3, 3, 6, 7, 8, 9}, the median is 6, the fourth number in the sample. This method is good enough for our case, and it was one of our two options we were considering.

The last method that was our final choice, was to find the average of a specific sample. The most important factor for this method was to set properly the size of the sample that is being considered every time. If it was too large then the prediction wouldn't be accurate, and if it was too small then we wouldn't have considered all the information. Finally, the size of the list of the samples was decided to be 5. In other words, the prediction of the next heart beat each time is considered as the average value of the previous 5 beats.

4.5.3. Synchronization with ECG

In order the experiments to lead to meaningful results, we had to be sure that the data of the oximeter was 95% accurate. That means that no more than 50ms variation should occur between the heartbeat predictions and the actual heartbeat. So as to prove this, we attached the wires of an Electrocardiogram to a user while at the same time his cardiac rate was being monitored by a pulse oximeter.

The first step was to secure that the oximeter provided correct raw data comparing to the ECG. The way to test this, was to create an analog signal from Unity program and monitor it at the same computer that was monitoring the ECG data. Unfortunately, we saw that there was a big gap between these two signals (350ms) and we figured out that the problem occurred because of the latency of

Chapter 4 – Implementation

Unity code. The solution to this problem was to measure and then add the amount of the system delay to the oximeter measurement. Afterwards, several trials of testing led to the significant amount of 95% accuracy on the heartbeat prediction comparing with the ECG data.

4.6. Events storage

As events are considered the obstacles that appear in front of the users and the several actions that they could probably do to avoid them. In the default experiment the amount of events is 104. For every event that occurs, a number of parameters is saved in order to process them and result to conclusions.

These events are being saved in lists of the following variables:

- **Obstacle Name:** The actual name of the obstacle that the user confronted (e.g. barrel, horse).
- **Obstacle Distance:** The distance between the car and the obstacle (e.g. far, near).
- **Obstacle Speed:** The obstacle's moving speed (e.g. fast, slow).
- **Reaction Time:** Reaction time for either brake, swerve or accelerate of the user.
- **Brake Data:** The rate of braking or accelerating (positive or negative value of this parameter).
- **Swerve Data:** The rate of steering the car.
- **Speed of the Car:** The actual speed of the car about 1 second before the obstacle appears.
- **Heartbeat Distance:** The timing variance between the obstacle's spawning and the heartbeat (e.g. 0ms is the variance for systole and -300ms is the variance for diastole).
- **R wave Distance:** The timing variance between the obstacle's spawning and the R wave (e.g. 0ms is the variance for diastole and 300ms is the variance for systole).
- **Collision Flag:** The information about the collision of the car with the obstacle
- **Expected Collision:** The information about the time of expected collision. Time for expected collision is considered as the time that the car would collide

Chapter 4 – Implementation

with the obstacle if the user wouldn't have reacted. To find the time for expected collision, the following math equation is used:

$$U_1 * t + U_2 * t = X \quad (1)$$

Since the car and the obstacle are moving towards each other, we can use the equation [\(1\)](#). The variable U_1 represents the speed of the car, the variable U_2 represents the speed of the obstacle, the variable X represents the distance between the car and the obstacle, and the variable t represents the expected time for collision.

- **Real Collision:** The time until the car reaches the obstacle without considering if they collided or not. If the car does not collide with the obstacle, as a real collision time is considered the time until the car reaches the line in x-axis of the obstacle.

4.7. Database implementation

For the purposes of this project it was required to save the data of every event. These events had to be categorized by several information as well as for each user. It was also required to access them through a web interface in order to provide the opportunity to the experimenter to see the results organized.

Databases store information in electronic records that may be searched, retrieved and organized in countless ways. Having stored the information in a database, instead of on paper or in spreadsheets, not only saves user's time and preserves vital information, it allows also to see patterns in the operations that are visible in no other way.

In this project, as it is described in Chapter 3, the phpMyAdmin has been used in combination with XAMPP software tools. The XAMPP tool used as it offers an Apache web server which processes and delivers web content to a computer and it's the most popular web server online. It also offers a MySQL database server in order to connect with the phpMyAdmin software tool that's handling the MySQL database over the web.

The structure of our database consists of two tables ([see Figure \[4.32\]](#)), the table of users that keeps some personal information about the users who take part in

Chapter 4 – Implementation

the experiments and the table of reaction times which keeps all the information needed for the events.

Table	Action	Rows	Type	Collation	Size	Overhead
reactiontimes	Browse Structure Search Insert Empty Drop	1,029	InnoDB	latin1_swedish_ci	176 KiB	-
subjects	Browse Structure Search Insert Empty Drop	10	InnoDB	latin1_swedish_ci	32 KiB	-
2 tables	Sum	1,039	InnoDB	latin1_swedish_ci	208 KiB	0 B

Figure [4.32]: Database tables.

The personal information (see [Figure \[4.33\]](#)) that's being stored about the users is an auto incremented unique ID, a unique username, the age of each subject as well as their gender. It is important to mention that the ID and the username of each subject cannot be the same for two different users.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	ID	int(11)			No	None	AUTO_INCREMENT
2	Username	varchar(20)			No	None	
3	Age	int(2)			No	None	
4	Gender	varchar(6)			No	None	

Figure [4.33]: Database personal information table.

The user is stored in the database using a PHP script with SQL code (see [Figure \[4.34\]](#)).

```
$sql = "INSERT INTO subjects(Username, Age, Gender )  
VALUES ('" . $Username . "', '" . $Age . "', '" . $Gender . "')" ;
```

Figure [4.34]: Insert personal information in Database.

Finally, all the events that have been saved in the type of lists in Unity will be stored in the database after each experiment is over (see [Figure \[4.35\]](#)). Firstly, the

Chapter 4 – Implementation

id number of the previous table is copied so that we can link the personal information of the users with their events data. Then another unique ID for each event is created as well as several parameters relative with the events are being stored, such as the obstacle's name, the obstacle's distance, the obstacle's speed, as well as user's reaction time, breaking data, swerving data, the speed of the car, the systole distance, the R wave distance, the collision flag, the expected collision time and the real collision time.




#	Name	Type	Collation	Attributes	Null	Default	Extra
1	ID 	int(11)			No	None	AUTO_INCREMENT
2	id_num  	int(11)			No	None	
3	Obstacle	varchar(20)			Yes	NULL	
4	Obst_Distance	varchar(20)			Yes	NULL	
5	Obst_Speed	varchar(20)			Yes	NULL	
6	ReacTime	double			Yes	NULL	
7	Breaking_Data	double			Yes	NULL	
8	Swerving_Data	double			Yes	NULL	
9	Speed_Before_Obst	double			Yes	NULL	
10	HeartBeat_Distance	double			Yes	NULL	
11	R_Wave_Dist	double			Yes	NULL	
12	Colision_Flag	varchar(20)			Yes	NULL	
13	Expected_col	double			Yes	NULL	
14	Real_col	double			Yes	NULL	

Figure [4.35]: Database events' table.

In order to transfer the values from Unity to database, an SQL query has been written in PHP code ([see Figure \[4.36\]](#)).

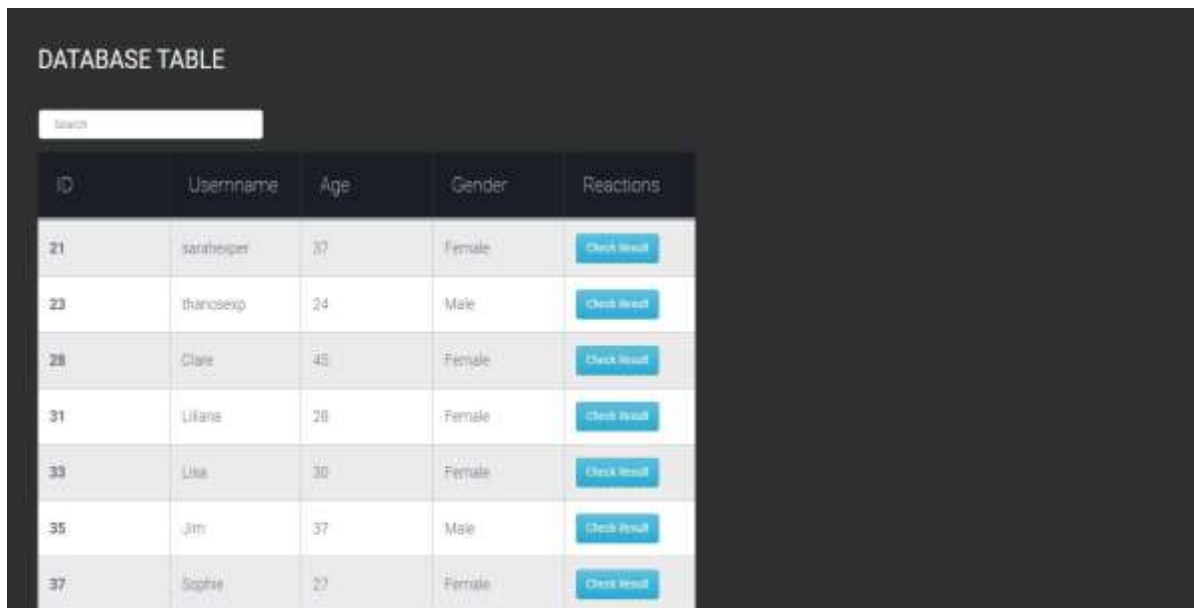
Chapter 4 – Implementation

```
$sql2 = "INSERT INTO reactiontimes(id_num,Obstacle,Obst_Distance,Obst_Speed,
ReactTime,Breaking_Data,Swerving_Data,Speed_Before_Obst,HeartBeat_Distance,
R_Wave_Dist,Colision_Flag,Expected_col,Real_col)
VALUES (
'" . $row[ID] . "',
'" . $Obstacle . "',
'" . $Obst_Distance . "',
'" . $Obst_Speed . "',
'" . $ReactTime . "',
'" . $Breaking_Data . "',
'" . $Swerving_Data . "',
'" . $Speed_Before_Obst . "',
'" . $HeartBeat_Distance . "',
'" . $R_Wave_Dist . "',
'" . $Colision_Flag . "',
'" . $Expected_col . "',
'" . $Real_col . "')";
```

Figure [4.36]: Insert events in Database.

4.8. Webpage implementation

The experimenter can access the web interface that has been developed, by choosing the right option from the menu. If he does so, he will be transferred to the web page that shows all the users who have done the experiment ([see Figure \[4.37\]](#)).

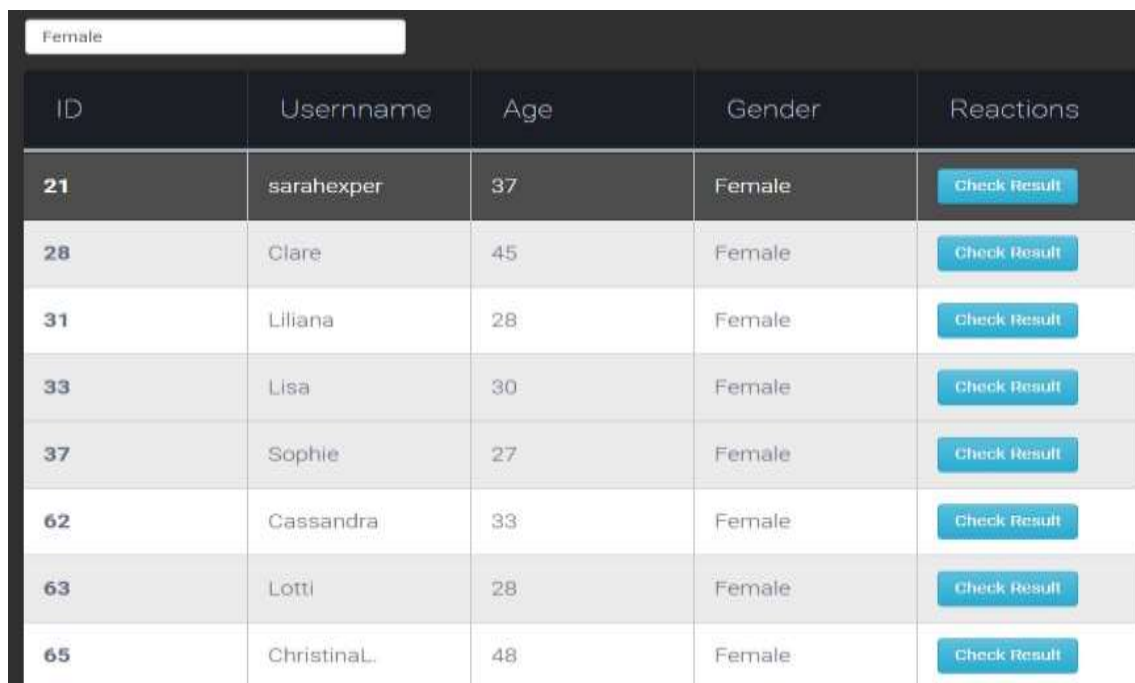


ID	Username	Age	Gender	Reactions
21	sarahesper	37	Female	<button>Check Result</button>
23	thanoexp	24	Male	<button>Check Result</button>
28	Clare	45	Female	<button>Check Result</button>
31	Liliane	28	Female	<button>Check Result</button>
33	Lisa	30	Female	<button>Check Result</button>
35	Jim	37	Male	<button>Check Result</button>
37	Sophie	22	Female	<button>Check Result</button>

Figure [4.37]: Web Interface.

Chapter 4 – Implementation

In the image below, we can see the field where the experimenter can search for a specific person, or a category ([see Figure \[4.38\]](#)). The ability to see the personal information of each user is provided, and a button that opens two more windows in order to show: the events that have been stored for the selected person ([see Figure \[4.39\]](#)), and some statistics for this person based on specific categories ([see Figure \[4.40\]](#)). These statistics are split up on the following categories: diastole, systole, diastole for near and slow obstacles, systole for near and slow obstacles, diastole for far and slow obstacles, systole for far and slow obstacles, diastole for near and fast obstacles, systole for near and fast obstacles, diastole for far and fast obstacles, systole for far and fast obstacles, diastole for side obstacles and systole for side obstacles. For each one of these categories, we show the reaction time of the user, the number of collisions with obstacles and how many times the subject avoided the obstacles.



ID	Username	Age	Gender	Reactions
21	sarahexper	37	Female	Check Result
28	Clare	45	Female	Check Result
31	Liliana	28	Female	Check Result
33	Lisa	30	Female	Check Result
37	Sophie	27	Female	Check Result
62	Cassandra	33	Female	Check Result
63	Lotti	28	Female	Check Result
65	ChristinaL.	48	Female	Check Result

Figure [4.38]: Searching for females.

Chapter 4 – Implementation

Gender	Reactions	Obstacle Name	Obstacle Distance	Obstacle Speed	Time For Response
Female	Check Result	Adult	Right	Normal	0.6168365
Female	Check Result	Barrel	Near	Fast	0.3222198
Female	Check Result	Horse	Near	Slow	0.5676994
Female	Check Result	Barrel	Near	Fast	0.4697266
Female	Check Result	Child	Left	Normal	0.03141785
Female	Check Result	Horse	Near	Fast	0.5088501
Female	Check Result	Child	Left	Normal	0.228302
Female	Check Result	Horse	Near	Slow	0.2152252
Female	Check Result	Barrel	Near	Slow	0.5593872
		Barrel	Far	Fast	0.4532623
		Barrel	Far	Fast	0.4871674
		Barrel	Near	Slow	0.5166168

Figure [4.39]: Events for a specific person.

Diastole Reaction	Systole Reaction	Diastole Near Slow	Systole Near S
0.418983456	0.4327133275	0.3499860675	0.454595
7 collided	9 collided	1 collided	3 collided
13 not collided	11 not collided	3 not collided	1 not coll

Figure [4.40]: Statistics for specific categories of events.

When the experimenter selects to see another user in the Web Interface, an AJAX request executes in order to refresh only a specific part of the page ([see Figure \[4.41\]](#)).

Chapter 4 – Implementation

```
<script>
function myFunction(idd)
{
    if (window.XMLHttpRequest) {
        xmlhttp=new XMLHttpRequest();
    }
    else{
        xmlhttp=new ActiveXObject("Microsoft.XMLHttpRequest");
    }
    xmlhttp.onreadystatechange=function(){
        if (this.readyState==4 && this.status ==200){
            document.getElementById("display").innerHTML = this.responseText;
        }
    };
    xmlhttp.open("GET","getuser.php?q="+idd,true);
    xmlhttp.send();
}
</script>
```

Figure [4.41]: *AJAX request for getting the user from database.*

In order to calculate all the statistic that we provided in the web interface, numerous (35) SQL queries have been written ([see Figure \[4.42\]](#), [Figure \[4.43\]](#) and [Figure \[4.44\]](#)).

```
$sql35="SELECT count(*) cnt24 FROM reactiontimes WHERE id_num= '". $q. "'
AND Collision_Flag='NOT Collided' AND HeartBeat_Distance='-300'
AND (Obst_Distance='Right' OR Obst_Distance='Left')";
$result35 = mysqli_query($con,$sql35);
```

Figure [4.42]: *Example of SQL query that calculates the number of avoidance for the diastole of a specific person.*

```
$sql11="SELECT AVG(ReactTime) averagel2 FROM reactiontimes WHERE id_num= '". $q. "'
AND ReactTime<100 AND HeartBeat_Distance='-300'
AND (Obst_Distance='Right' OR Obst_Distance='Left')";
$result11 = mysqli_query($con,$sql11);
```

Figure [4.43]: *Example of SQL query that calculates the average reaction time for the diastole of a specific person.*

Chapter 4 – Implementation

```
$sql9="SELECT AVG(ReactTime) averagel0 FROM reactiontimes WHERE id_num= ' ".$q."'  
AND ReactTime<100 AND HeartBeat_Distance='0'  
AND Obst_Distance='Far' AND Obst_Speed='Fast';  
$result9 = mysqli_query($con,$sql9);
```

Figure [4.44]: *Example of SQL query that calculates the average reaction time for the systole of a specific person.*

5. Chapter 5 - Experiments

5.1. Experimental Procedure

The general goal of this project was to create a realistic 3D car driving simulator to be used for neuroscientific experimentation, in collaboration with the Brighton and Sussex Medical School. This project actually builds on earlier work at the Brighton and Sussex Medical School which shows that although many studies suggest that perceptual processing, particularly to painful stimuli, tend to be inhibited at cardiac systole, fear detection and processing is selectively enhanced at systole (S. Garfinkel & H. Critchley, 2016). Because of this conflict the goal of the experiments to be conducted is to investigate whether users react faster at cardiac systole expressing, in this way, enhanced fear processing or whether they react slower at cardiac systole based on their inhibited processing. This work also builds upon previous research at the Brighton and Sussex medical School looking at responses to ballistic stimuli (H. Prins et al., 2015) that are time-locked to distinct parts of the cardiac cycle and also demonstrated that at cardiac systole people tend to have faster reactions.

In order to test the application with a variety of users, several steps followed to conduct a successful experiment. The experimental procedure consists of the following phases: consent of the users, filling of a demographics form, filling of questionnaires, heart beat detection tests and finally the driving.

Chapter 5 - Experiments

5.1.1. Consent of the Users

Voluntary informed consent is a prerequisite for a subject's participation in research. It is a process, in which the subject has an understanding of the research and its risks. Informed consent is essential before enrolling a participant and ongoing once enrolled. Obtaining consent involves informing the subject about his or her rights, the purpose of the study, the procedures to be undergone, and the potential risks and benefits of participation. Users in the study must participate willingly.

The Belmont Report (<http://ohsr.od.nih.gov/guidelines/belmont.html>) and the Nuremberg Code (<http://www.cirp.org/library/ethics/nuremberg/>) both address voluntary informed consent as a requirement for the ethical conduct of human users' research. Informed Consent is the process through which researchers respect individual autonomy, the fundamental ethical principle. An autonomous individual is one who is capable of deliberation and personal choice. The principle of autonomy implies that responsibility must be given to the individual to make the decision to participate. Informed Consent means that users are well informed about the study, the potential risks and benefits of their participation and that it is research, not therapy, in which they will participate.

5.1.2. Demographics form

The following demographics form ([see Figure \[5.1\]](#)) is going to be provided to the users:

Chapter 5 - Experiments

Demographic Information

Subject ID: _____

1. Age _____ years
2. Gender _____ male _____ female
3. Height _____ cm
4. Weight _____ kg
5. What is the highest level of education you have completed?
 - a. GCSC or similar
 - b. A-Levels or similar
 - c. Attended university or business college but did not receive a degree
 - d. Received Undergraduate degree
 - e. Post-graduate degree
6. Do you smoke?
 - a. no (never have)
 - b. occasionally
 - c. not anymore, stopped _____ months ago
 - d. yes, average number of cigarettes a day: _____
7. How much alcohol do you drink on average?
_____ servings (drinks) per typical week
8. Is there any strenuous physical activity on your current job?
 - a. Yes
 - b. No
9. Do you engage in regular physical exercise for recreation off the job?
 - a. Yes
 - b. No

If yes, how often do you exercise on average per week? _____
and what type of exercise do you do most of the time? _____
10. Have you done any exercise before the experiment today?
 - a. Yes
 - b. No
11. Do you have any experience with meditation or Buddhist practises such as Yoga or Tai Chi or Mindfulness?
 - a. Yes
 - b. No

If yes, what have you done and for how long? _____

12. Do you consider yourself
 - a. Right handed
 - b. Left Handed
 - c. Ambidextrous

Chapter 5 - Experiments

13. Are you currently any taking medications or drugs?
a. Yes b. No

If yes, what is the drug and how long have you been taking it?

14. Do you currently experience any symptoms of a cold?
a. Yes b. No

15. Do you currently experience a headache?
a. Yes b. No

16. When was your last meal before the experiment? _____ hours ago

17. Do you suffer (or have you ever suffered) from any of the following:

___ Asthma

___ Heart Disease

___ Gastric distress or digestive problems (Heartburn, Dyspepsia, Irritable Bowel Syndrome)

___ Eating Disorder

___ Rheumatologic Disorders or Arthritis

___ Psychiatric Disorders

___ Neurological Disease (Migraine headaches, Trauma, Epilepsy,...)

___ Diabetes

___ Endocrine problems (e.g., thyroid, adrenal, or gonadal hormone dysfunction)

___ Diseases of the respiratory system

___ Stroke or Heart Attack

___ (Partial) Blindness

___ Cancer

18. Did you consume any caffeinated drinks before the arrival to the experiment today?
a. yes b. no

If yes, how many drinks did you have today? _____

19. Did you use any medication today?

a. yes b. no

If yes, what type of medication did you take? _____

Figure [5.1]: *Demographics form.*

Chapter 5 - Experiments

5.1.3. Questionnaires

In order to determine the state of anxiety of the users, the following questionnaire ([see Figure \[5.2\]](#)) will be given to them and they will have to circle the appropriate number to the right of the statement to indicate how they feel at that specific moment.

	NOT AT ALL	SOMEWHAT	MODERATELY SO	VERY MUCH SO
1. I feel calm	1	2	3	4
2. I feel secure	1	2	3	4
3. I am tense	1	2	3	4
4. I feel strained	1	2	3	4
5. I feel at ease	1	2	3	4
6. I feel upset	1	2	3	4
7. I am presently worrying over possible misfortunes	1	2	3	4
8. I feel satisfied	1	2	3	4
9. I feel frightened	1	2	3	4
10. I feel comfortable	1	2	3	4
11. I feel self-confident	1	2	3	4
12. I feel nervous	1	2	3	4
13. I am jittery	1	2	3	4
14. I feel indecisive	1	2	3	4
15. I am relaxed	1	2	3	4
16. I feel content	1	2	3	4
17. I am worried	1	2	3	4

Chapter 5 - Experiments

18. I feel confused	1	2	3	4
19. I feel steady	1	2	3	4
20. I feel pleasant	1	2	3	4

	ALMOST NEVER	SOMETIMES	OFTEN	ALMOST ALWAYS
21. I feel pleasant	1	2	3	4
22. I feel nervous and restless	1	2	3	4
23. I feel satisfied with myself	1	2	3	4
24. I wish I could be as happy as others seem to be	1	2	3	4
25. I feel like a failure	1	2	3	4
26. I feel rested	1	2	3	4
27. I am “calm, cool and collected”	1	2	3	4
28. I feel that difficulties are piling up so that I cannot overcome them	1	2	3	4
29. I worry too much over something that doesn’t really matter	1	2	3	4
30. I am happy	1	2	3	4
31. I have disturbing thoughts	1	2	3	4
32. I lack self-confidence	1	2	3	4

Chapter 5 - Experiments

33. I feel secure	1	2	3	4
34. I make decisions easily	1	2	3	4
35. I feel inadequate	1	2	3	4
36. I am content	1	2	3	4
37. some unimportant thought runs through my mind and bothers me	1	2	3	4
38. I take disappointments so keenly that I can't put them out of my mind	1	2	3	4
39. I am a steady person	1	2	3	4
40. I get in a state of tension or turmoil as I think over my recent concerns and interests	1	2	3	4

Figure [5.2]: *State of Anxiety Questionnaire.*

Another questionnaire, probably the most important for this project, the Baratt (see [Figure \[5.4\]](#)), will be given to people to determine how impulsive they are; it has three main factors, looking impulsivity in the attentional domain, motor domain and non-planning domain as explained in [Figure \[5.3\]](#).

2nd Order Factors	1st Order Factors	# of items	Items contributing to each subscale
Attentional	Attention	5	5, 9*, 11, 20*, 28
	Cognitive Instability	3	6, 24, 26
Motor	Motor	7	2, 3, 4, 17, 19, 22, 25
	Perseverance	4	16, 21, 23, 30*
Nonplanning	Self-Control	6	1*, 7*, 8*, 12*, 13*, 14
	Cognitive Complexity	5	10*, 15*, 18, 27, 29*
			*reverse scored items

Figure [5.3]: *Table for Baratt Questionnaire explanation.*

Chapter 5 - Experiments

DIRECTIONS: People differ in the ways they act and think in different situations. This is a test to measure some of the ways in which you act and think. Read each statement and put an X on the appropriate circle on the right side of this page. Do not spend too much time on any statement. Answer quickly and honestly.				
① Rarely/Never	② Occasionally	③ Often	④ Almost Always/Always	
1 I plan tasks carefully.	①	②	③	④
2 I do things without thinking.	①	②	③	④
3 I make-up my mind quickly.	①	②	③	④
4 I am happy-go-lucky.	①	②	③	④
5 I don't "pay attention."	①	②	③	④
6 I have "racing" thoughts.	①	②	③	④
7 I plan trips well ahead of time.	①	②	③	④
8 I am self controlled.	①	②	③	④
9 I concentrate easily.	①	②	③	④
10 I save regularly.	①	②	③	④
11 I "squirm" at plays or lectures.	①	②	③	④
12 I am a careful thinker.	①	②	③	④
13 I plan for job security.	①	②	③	④
14 I say things without thinking.	①	②	③	④
15 I like to think about complex problems.	①	②	③	④
16 I change jobs.	①	②	③	④
17 I act "on impulse."	①	②	③	④
18 I get easily bored when solving thought problems.	①	②	③	④
19 I act on the spur of the moment.	①	②	③	④
20 I am a steady thinker.	①	②	③	④
21 I change residences.	①	②	③	④
22 I buy things on impulse.	①	②	③	④
23 I can only think about one thing at a time.	①	②	③	④
24 I change hobbies.	①	②	③	④
25 I spend or charge more than I earn.	①	②	③	④
26 I often have extraneous thoughts when thinking.	①	②	③	④
27 I am more interested in the present than the future.	①	②	③	④
28 I am restless at the theater or lectures.	①	②	③	④
29 I like puzzles.	①	②	③	④
30 I am future oriented.	①	②	③	④

Figure [5.4]: Baratt Questionnaire.

Chapter 5 - Experiments

In addition the Alexithymia questionnaire of Toronto ([see Figure \[5.5\]](#)) which monitors how good people are in understanding their own emotions might also be provided to the users.

Indicate how much you agree or disagree with each of the following statements. Just tick the appropriate box. Use the middle box ('I neither agree or disagree') only if you are really unable to assess your behaviour.	I strongly disagree	I quite disagree	I neither agree nor disagree	I quite agree	I strongly agree
1- I am often confused about what emotion I am feeling					
2- It is difficult for me to find the right words for my feelings					
3- I have physical sensations that even doctors don't understand					
4- I am able to describe my feelings easily					
5- I prefer to analyze problems rather than just describe them					
6- When I am upset, I don't know if I am sad, frightened, or angry					
7- I am often puzzled by sensations in my body					
8- I prefer to just let things happen rather than to understand why they turned out that way					
9- I have feelings that I can't quite identify					
10- Being in touch with emotions is essential					
11- I find it hard to describe how I feel about people					
12- People tell me to describe my feelings more					
13- I don't know what's going on inside me					
14- I often don't know why I am angry					
15- I prefer talking to people about their daily activities rather than their feelings					
16- I prefer to watch « light » entertainment shows rather than psychological dramas					
17- It is difficult for me to reveal my innermost feelings, even to close friends					
18- I can feel close to someone, even in moments of silence					
19- I find examination of my feelings useful in solving personal problems					
20- Looking for hidden meanings in movies or plays distracts from their enjoyment					

Figure [5.5]: Alexithymia Questionnaire.

Chapter 5 - Experiments

Finally, the Buss Scale questionnaire, which measures people's anger and hostility might also be added ([see Figure \[5.6\]](#)).

Using the 5 point scale shown below, indicate how uncharacteristic or characteristic each of the following statements is in describing you. Place your rating in the box to the right of the statement.

- 1 = extremely uncharacteristic of me
- 2 = somewhat uncharacteristic of me
- 3 = neither uncharacteristic nor characteristic of me
- 4 = somewhat characteristic of me
- 5 = extremely characteristic of me

1. Some of my friends think I am a hothead
2. If I have to resort to violence to protect my rights, I will.
3. When people are especially nice to me, I wonder what they want.
4. I tell my friends openly when I disagree with them.
5. I have become so mad that I have broken things.
6. I can't help getting into arguments when people disagree with me.
7. I wonder why sometimes I feel so bitter about things.
8. Once in a while, I can't control the urge to strike another person.
9. I am an even-tempered person.
10. I am suspicious of overly friendly strangers.
11. I have threatened people I know.
12. I flare up quickly but get over it quickly.
13. Given enough provocation, I may hit another person.
14. When people annoy me, I may tell them what I think of them.
15. I am sometimes eaten up with jealousy.

Chapter 5 - Experiments

16. I can think of no good reason for ever hitting a person.
17. At times I feel I have gotten a raw deal out of life.
18. I have trouble controlling my temper.
19. When frustrated, I let my irritation show.
20. I sometimes feel that people are laughing at me behind my back.
21. I often find myself disagreeing with people.
22. If somebody hits me, I hit back.
23. I sometimes feel like a powder keg ready to explode.
24. Other people always seem to get the breaks.
25. There are people who pushed me so far that we came to blows.
26. I know that "friends" talk about me behind my back.
27. My friends say that I'm somewhat argumentative.
28. Sometimes I fly off the handle for no good reason.
29. I get into fights a little more than the average person.

Figure [5.6]: *Buss Scale Questionnaire.*

5.1.4. Heart Detection Tests

In order to secure that our experiment is going to provide some useful results we have to test the interoception of the users. Interoception refers to the sensing of internal bodily changes. Also it interacts with cognition and emotion, making measurement of individual differences in interoceptive ability broadly relevant to neuropsychology. In other words, interoception is the body-to-brain axis of sensation concerning the state of the internal body and its visceral organs. (S. Garfinkel et al., 2014)

Studies have shown that those who are better able to detect their internal bodily signals (i.e. who have higher interoceptive accuracy) tend to be more emotional. Users are tested using the Schandry (1981) method as well as the Katkin

Chapter 5 - Experiments

(1983) method. Based on Schandry, emotional experience is coupled to perception of bodily processes. From this it is deduced that individuals who show good perception of heart activity tend to exhibit higher levels of a momentarily experienced emotion. In order to test the interoception of the users with Schandry method, people will be instructed to count silently their heart beats, only by concentrating on their body but not by taking their pulse, during a signaled time interval and the reported number of beats will then be compared to the actual number of beats as extracted from the ECG (see [Figure \[5.7\]](#)). The greater the overlap between people's perceived and actual heartbeats, the better their interoceptive accuracy.

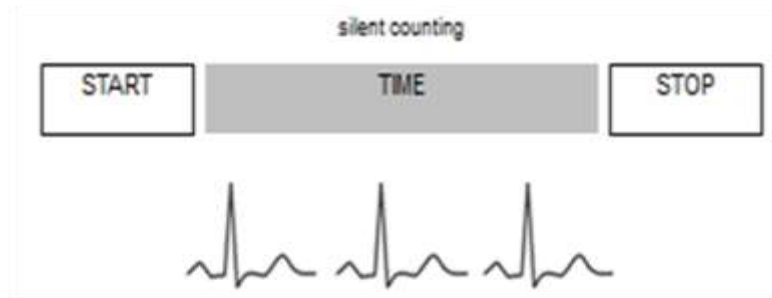


Figure [5.7]: *Schandry heart beat Perception.*

The second measure of interoception is the heartbeat detection task (Katkin et al., 1983, Whitehead et al., 1977) (see [Figure \[5.8\]](#)). Based on this method, people will receive auditory tones that will be triggered by their heartbeats and played either on the heartbeat or slightly time shifted ($\sim 500\text{ms}$). The participant's task will be to judge when the tones are synchronous with their heartbeat. Again, higher accuracy in this task indicates better interoceptive accuracy.

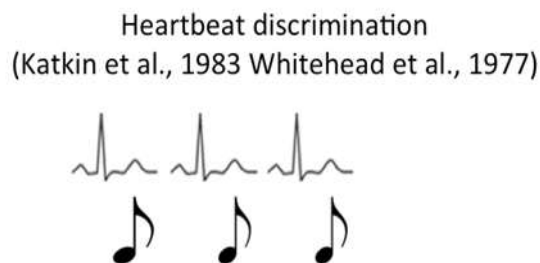


Figure [5.8]: *Katkin et al., 1983, Whitehead et al., 1977.*

Chapter 5 - Experiments

5.1.5. Driving

After the consent of the users, the demographics form, the questionnaires and the heart beat detection tests, the experimenter sets the experiment for each user. The height of their seat should be convenient for them and they wear the Oculus Rift as well as the oximeter with the help of the experimenter. The aim for the users is to be focused to driving a VR driving simulator and avoid obstacles as fast as possible. During the experiment they should avoid talking, unless they have a problem.

5.2. Results

A small amount of pilot experiments has been conducted in Brighton and Sussex Medical School as a preliminary stage of our main experimental phase. Reaction times of users were categorized based on their cardiac phase and the type of the object. Seven people were participated in the experiment with the following reaction times ([see Figure \[5.9\]](#)).

Diastole Reaction	Systole Reaction	Diastole Near Slow	Systole Near Slow	Diastole Far Slow	Systole Far Slow	Diastole Near Fast	Systole Near Fast	Diastole Far Fast	Systole Far Fast
0.41898	0.43271	0.3499	0.45459	0.57419	0.59857	0.3287	0.4115	0.2897	0.4618
0.45485	0.41522	0.5852	0.61979	0.31103	0.22615	0.4056	0.3452	0.4779	0.3447
0.52548	0.46816	0.6268	0.31273	0.48824	0.46933	0.4305	0.3797	0.4309	0.6433
0.34525	0.39095	0.3953	0.32676	0.27895	0.34778	0.2146	0.4255	0.2153	0.4358
0.48019	0.41393	0.3650	0.46560	0.49495	0.36811	0.4449	0.3399	0.5271	0.4379
0.63259	0.46315	0.4252	0.46188	0.42996	0.42645	0.3847	0.4370	0.4845	0.5402
0.385	0.33959	0.2622	0.35436	0.37344	0.18199	0.6007	0.3853	0.2107	0.2985

Figure [5.9]: Reaction Times Categorized

Afterwards we plotted the average reaction times for systole and diastole based on each category of the obstacles. We can clearly see that the nearer obstacles

Chapter 5 - Experiments

have a slight gap between systole and diastole. This probably happens because it is harder to avoid these obstacles so their cardiac rate didn't affect their reaction time a lot. Fortunately, in both categories (slow and fast) people reacted faster in systole as some of the previous studies had indicated. About the far obstacles, we see a controversial phenomenon. For the slow obstacles, people reacted faster in systole but when they faced the fast obstacles they reacted much faster in diastole ([see Figure \[5.10\]](#)).

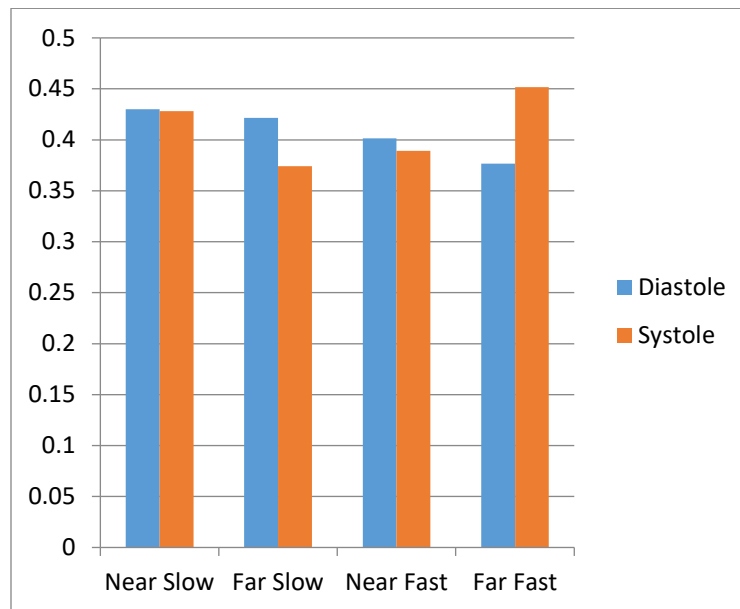


Figure [5.10]: *Average Reaction Times Categorized.*

In the following table ([see Figure \[5.11\]](#)), we can see the differences in reaction times of each subject for diastole and systole. In this way, we can see if there is a significant variance between their reaction times during their cardiac cycle. In the Near Slow category, the third subject has the biggest variance of systole and diastole reaction times and it is very important to mention that this specific subject had very high interoception which means she were more aware of her body stage. Unfortunately, the other users do not have enough variance between their systole and diastole reaction time so they cannot be taken into consideration. A larger number of experiments is required in order to conclude in more accurate results.

Chapter 5 - Experiments

<i>Near Slow</i>	<i>Far Slow</i>	<i>Near Fast</i>	<i>Far Fast</i>
-0.104609508	-0.024381625	-0.082815218	-0.172129618
-0.034582125	0.084886558	0.06035807	0.133148215
0.31409526	0.018910207	0.05088448	-0.212392397
0.068544004	-0.068823419	-0.210957324	-0.220475931
-0.100541818	0.1268436	0.105029293	0.089156669
-0.036673754	0.003512957	-0.052280013	-0.055681018
-0.092107571	0.191447863	0.215335875	-0.087837975

Figure [5.11]: *Variance of diastole and systole per subject.*

In the following plot (see [Figure \[5.12\]](#)), we can see the previous analysis in a more understandable way. We can see the differences of systole and diastole of all users based on the 4 categories of obstacles (near slow- far slow- near fast- far fast). In this design is clearer that the near slow obstacle has the highest peak of variance between systole and diastole.

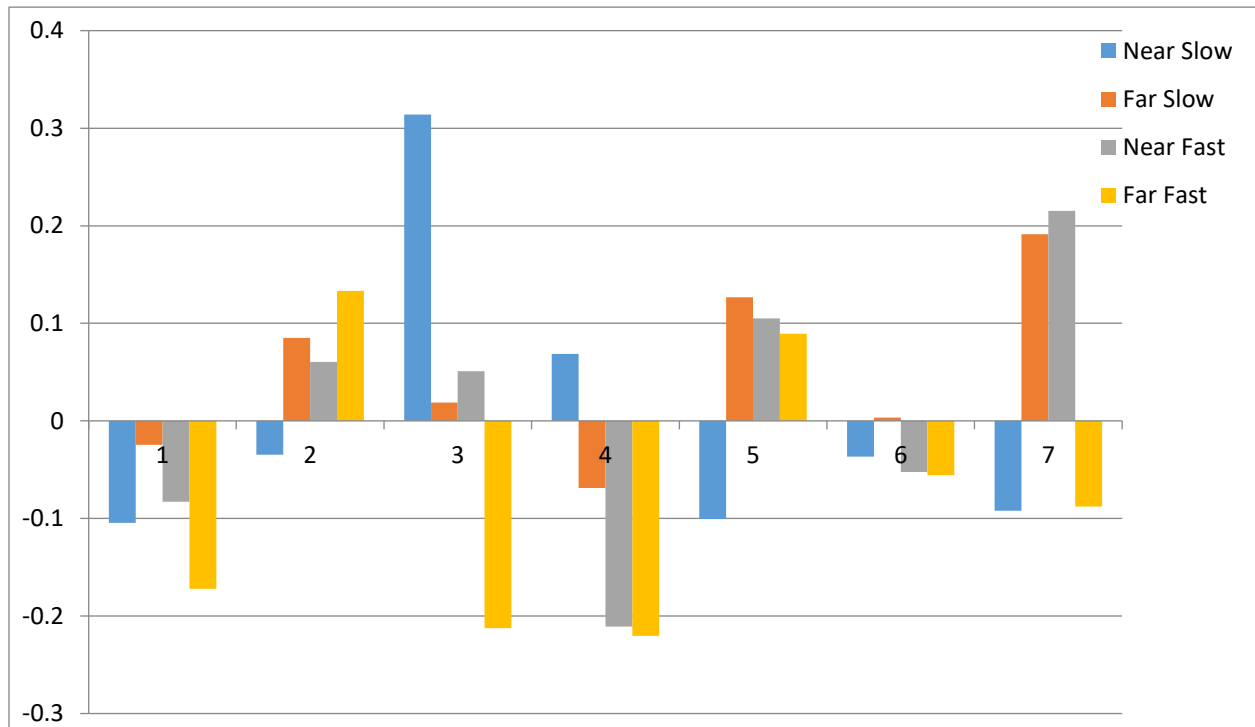


Figure [5.12]: *Variance of diastole and systole per subject.*

6. Chapter 6 - Conclusions and Future Work

This thesis is put forward the development of an interactive 3d driving simulator system to be projected on an Oculus Rift. The application was developed in order to investigate how signals from the heart alter driving responses to stimuli timed at distinct phases of the cardiac cycle while an individual is driving a car. A steering wheel as well as a pulse oximeter were integrated in the application to control the car while the cardiac rate of each subject is being monitored.

Although this work focused on the technical implementation of the system, the goal is to use this system to explore whether the users had significant variance in their reaction times between systole and diastole. Furthermore, for the first time a high fidelity application was proposed to measure the differences in people's responses between systole and diastole while the users are performing an everyday routine such as driving

It was a challenge to develop a high fidelity interactive 3D driving simulator which dynamically and accurately displays obstacles based on cardiac timing, satisfying experimental as well as technical demands. Previous attempts usually employed simpler tasks, for instance a 3D video game of lower graphics and just the use of one button without the capability of storing the user reactions in a database. Using VR in this project has the advantage of involving participants in interactive animated environments which more realistically reflect social and emotional

Chapter 6 - Conclusions and Future Work

situations. VR technology has already been successfully applied to neuroscientific research and the number of research groups that is using VR stimuli is growing day by day.

6.1. Main Contributions

The effectiveness of a Virtual Environments for experiments has often been linked to the sense of presence reported by the users. Presence is defined as the subjective experience of being in one place or environment, even when one is physically situated in another. It is argued that VEs that generate a higher feeling of presence would result in transfer equivalent to real-world situations, which would be extremely useful for conducting experiments in VE such as driving simulators.

This project aimed at developing an application which would be computationally cheap while generating a strong feeling of presence to the user. It is mandatory to have a computationally cheap application because experiments should occur in real time with no delays at all at a low cost computer. If we had sacrificed the level of detail of our application then, potentially, people would feel less comfortable driving the virtual car and the feeling of presence would be much lower. For the first time, a high fidelity VR application has been developed in three dimensions that not only enables the neuroscientists to get significant results from it, but also provides the user with a car driving game which is fun to play.

The main phase of this experiment is about to start. It is hoped that significant variance will be observed between reaction times on systole and diastole of the users. On one hand, we expect to see that people will react faster on diastole based on studies which support that perceptual processing, particularly of painful stimuli is inhibited at cardiac systole. Slower responses at systole are linked to cortical inhibition engendered by increased arterial baroreceptor activity (McIntyre et al. 2008). There is an inhibition of the processing of affectively salient painful stimuli at systole when arterial baroreceptors are firing. This is observed as a modulation of the nociceptive flexion reflex (Edwards et al. 2001) and attenuated pain-evoked potentials (Edwards et al. 2008).

On the other hand, there is also evidence that arterial baroreceptor activity during systole will facilitate the processing of other classes of affective stimuli. Fearful faces are detected more easily and perceived to be more intense when

Chapter 6 - Conclusions and Future Work

presented at systole compared to late diastole (S. Garfinkel et al. 2014). To a lesser extent there is also an enhancement of disgust processing (Gray et al. 2012). Thus, there appears to be a differential effect of arterial baroreceptor activity during systole, and thus physiological arousal, on the processing of signals of threat (implying a potential future physically aversive experience) and the processing of pain (an actual physically aversive experience) (S. Garfinkel and H. Critchley 2016).

It is also expected that people with greater interoceptive accuracy will have greater variance between their systole and diastole reaction times. This highlights the influence of brief interoceptive signal from the heart on behavioral and subjective responses to dynamic physical threats to the body and the modulation of these effects by specific attributes of the stimuli and by individual differences in interoceptive accuracy of the observer.

6.2. Future Work

The implementation of this project as well as the experiments described in detail in Chapter 4 and Chapter 5 were formally designed. However, certain improvements could be accomplished by the following actions:

- The experiments were designed in order to be run in a cheap computation and accurate environment but without sacrificing the fidelity. However, if the specifications of the computer were higher in relation to computational power, the fidelity and the graphics of the application would be much better and the feeling of presence could be much stronger. It would be interesting to investigate the differences of reaction times and cardiac timing between a driving environment of greater level of detail and the level of detail of the existing application, therefore, establishing a neuroscientific fidelity metric for VR.
- It would be very useful to check the differences in reaction times at systole and diastole in other everyday activities, such as walking, sailing, swimming etc. This innovative idea would help neuroscientists to investigate the effects of cardiac timing of events and how reactions differ when occurring during diverse cardiac phases.

References

- [1] S. N. Garfinkel and H. D. Critchley. Threat and the Body: How the Heart Supports Fear Processing. *Trends in Cognitive Sciences*, January 2016.
- [2] S. N. Garfinkel, L. Minati, M. A. Gray, A. K. Seth, R. J. Dolan, H. Critchley. Fear from the Heart: Sensitivity to Fear Stimuli Depends on Individual Heartbeats. *The Journal of Neuroscience*, 34(19): 6573-6582, 7 May 2014.
- [3] H. Prins, S. Garfinkel, K. Suzuki, A. Seth, H. Critchley 2015. The influence of interoceptive signals on detection of 3D ballistic stimuli. 19th annual meeting of the ASSC, July 7/10/2015 Paris, France.
- [4] H. Wu, M. Rubinstein, E. Shih, J. Guttag, F. Durand, W. T. Freeman. Eulerian Video Magnification for Revealing Subtle Changes in the World. *ACM Transactions on Graphics*, Volume 31, Number 4 (Proc. SIGGRAPH), 2012.
- [5] C. J. Bohil, B. Alicea, F. A. Biocca. Virtual reality in neuroscience research and therapy. *Nature Reviews Neuroscience*, 12(12):752-62, November 2011.
- [6] S. A. Aseeri. Virtual Reality Interaction Using Mobile Devices. *IEEE Symposium on 3D User Interfaces (3DUI)* 16-17 March 2013.
- [7] M. Mori. The Uncanny Valley: The Original Essay. *Energy*, vol. 7, no. 4, 1970, pp. 33–35.

References

- [8] O. D. Troyer, F. Kleinermann, B. Pellens, W. Bille. Conceptual modeling for virtual reality. In *Tutorials, Posters, Panels, and Industrial Contributions of the 26th international Conference on Conceptual Modeling (ER 07)*, Publ. CRPIT, Auckland, New Zealand, 2007
- [9] D. Perani, F. Fazio, N.A. Borghese, M. Tettamanti, S. Ferrari, J. Decety, M.C. Gilardi. Different brain correlates for watching Real and Virtual Hand Actions. *Neuroimage* 14(3):749-58, 2001 September.
- [10] R. McDonnell¹, S. Jorg, J. McHugh, F. Newell, C. O'Sullivan¹. Evaluating the emotional content of human motions on real and virtual characters. *Proceedings of the 5th Symposium on Applied Perception in Graphics and Visualization, APGV 2008*, Los Angeles, California, USA, August 9-10, 2008
- [11] P. Gamito, J. Oliveira, D. Morais, P. Rosa, T. Saraiva. Review of AR/VR in the Neuroscience Domain. Under CC BY 3.0 license. The Author(s), December 9 2011.
- [12] L. Pramme, M. F. Larra, H. Schanchinger, C. Frings. Cardiac Cycle time effects on selection efficiency in vision. *Society for Psychophysiological Research*, Volume 53, Issue 11, Pages 1609–1759, November 2016.
- [13] S. N. Garfinkel, A. K. Seth^b, A. B. Barrett^b, K. Suzuki, H. D. Critchley. Knowing your own Heart. *Biological Psychology* Volume 104, Pages 65–74, January 2015.
- [14] R. Schandry. Heart Beat Perception and Emotional Experience. *Psychophysiology*, 1981.
- [15] E. Katkin, S. Reed, C. Deroo. A Methodological Analysis of 3 Techniques for the Assessment of Individual-Differences in Heartbeat Detection. *Psychophysiology* 20:452-452, 1983.
- [16] J. M. Loomis, J. J. Blascovich, A. C. Beall. Immersive virtual environment technology as a basic research tool in psychology. *Behav. Res. Methods Instrum. Comput.* 31, 557–564, 1999.
- [17] M. J. Tarr, W. H. Warren. Virtual reality in behavioral neuroscience and beyond. *Behavior Research Methods, Instruments, & Computers*, Volume 31, Issue 4, December 1999, pp 557–564.

References

- [18] A. A. Rizzo & G. J. Kim. A SWOT analysis of the field of virtual reality rehabilitation and therapy. *Presence* 14, 119–146, 2005.
- [19] C. Weidemann, M. Mollison & M. Kahana. Electrophysiological correlates of high-level perception during spatial navigation. *Psychon. Bull.Rev.* 16, 313–319, 2009.
- [20] S. Fry, N. Rohrezeit, A. Straw, M. Dickinson. TrackFly: virtual reality for a behavioral system analysis in free-flying fruit flies. *J. Neurosci. Methods* 171, 110–117, 2008.
- [21] G. Corey, M. Schneider-Corey, P. Callanan, *Issues and ethics in the helping profession* (8th ed.). Belmont, CA: Brooks & Cole, Cengage Learning, 2011.
- [22] G. Riva. Virtual Reality in Psychotherapy: Review. *Cyber psychology & behavior*. Volume 8, Number 3, 2005
- [23] Unity3D.
<http://unity3d.com/>
- [24] 3D Model Repositories.
<http://archive3d.net/>,
<http://www.3dmodelfree.com/>,
<https://www.assetstore.unity3d.com/en/#!/home>,
<http://tf3dm.com/3d-models/unity>,
<http://www.turbosquid.com/>