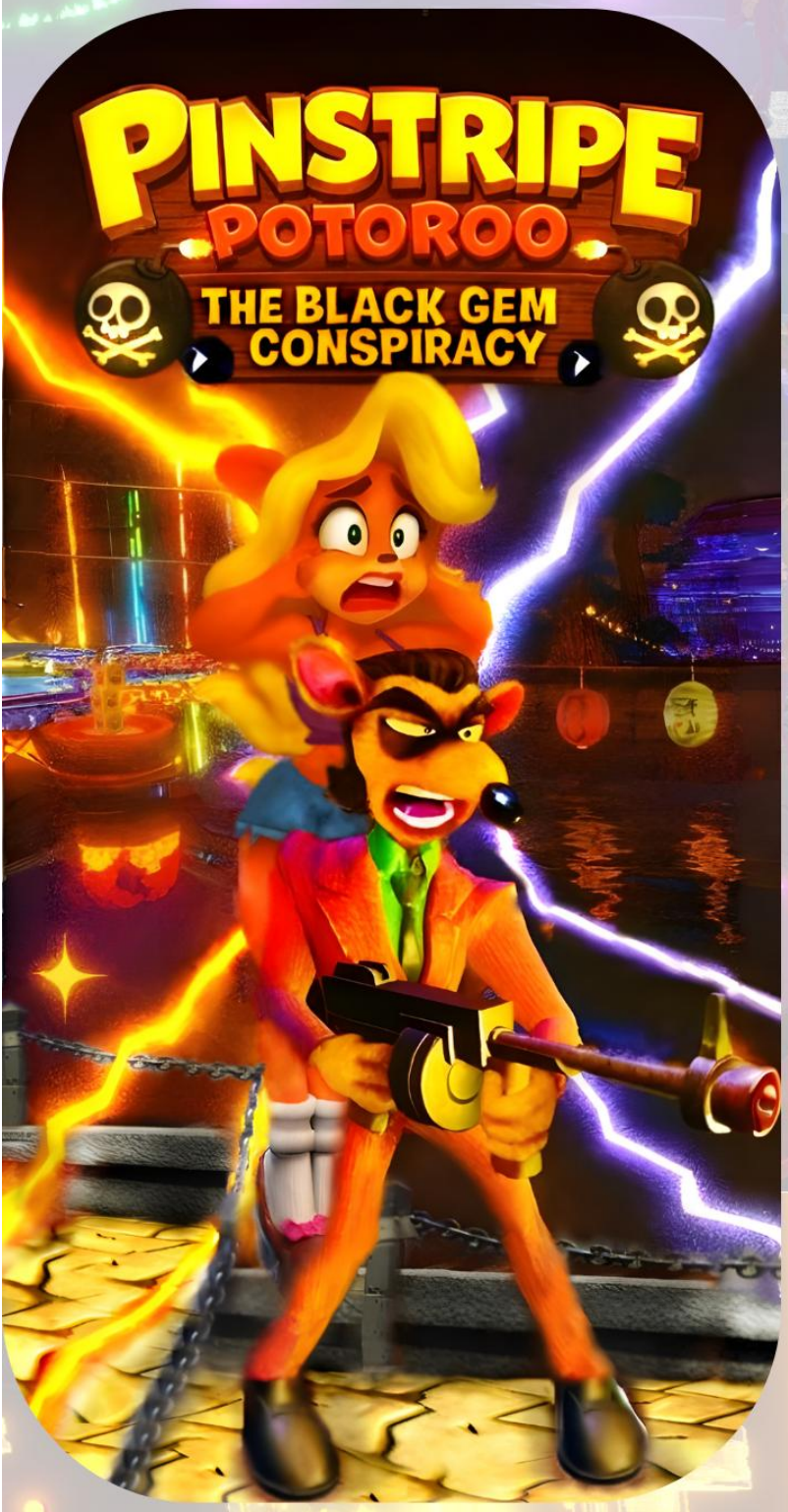
Finding the Fastest Path in a Self-Developed Game Using Optimization Techniques and Game Theory



Achilleas Missos

Examination Committee:

Ioannis Marinakis

Magdalini Marinaki

Pavlos Fafalios

TECHNICAL
UNIVERSITY OF CRETE
SCHOOL OF PRODUCTION
ENGINEERING AND MANAGEMENT

September 2025

I would like to thank my supervisors, Mr. Ioannis Marinakis and Mrs. Magdalini Marinaki, for giving me the opportunity to showcase an old game project I created when I was 13 years old, in our 9th semester logistics course bonus presentation. When I shared my project with them, their enthusiasm was evident. They fully supported the idea of making the project fully realized. During a conversation about the possibility of turning this childhood dream of mine into my diploma thesis; combining it with optimization methods and game theory techniques, it was a clear decision taking this idea to the next level. I am thankful for my family for their unconditional love and their involvement in my project. They supported me through discussing about ideas relative to my project, became my game testers and shared my happiness of every little milestone I had set and achieved in my game development with pride. My friends also were there along the way, they kept me grounded and balanced during long hours of development, preventing me from isolating myself completely as I had when I first attempted to create this game ten years ago. A special note of gratitude goes to the work I did in the summer at the Ammades beach bar, which allowed me to purchase a new laptop to proceed with developing this game. What was only a dream years ago has now become a reality, and this project would not have been possible without this opportunity that has been given to me.

Abstract

This thesis investigates how optimization techniques and game theory can be applied within the design of a self-developed 3D platformer/third-person shooter action-adventure game. After outlining the background of related games, genres, and academic literature, the development process of the game was described in detail, from asset creation and workflows to Unity-based implementation. The core contribution of this study lies in the application of game-theoretic principles to level design, focusing on the analysis of player objectives and resource allocation. A case study illustrated how concepts like payoff structures can guide design choices. The evaluation showed that while the models provide useful insights, they remain theoretical tools rather than validated predictions. Nevertheless, this project contributes to the discussion of how academic principles from game theory and optimization can inform creative development and provides a foundation for future research and empirical testing.

Contents

1. Introduction 1

1.1 Overview of video games 1
1.2 Introduction to Game Theory 2
1.3 Structure of Thesis 2

2. Background and Related Work 4

2.1 Inspiration and Related Games 4
2.1.1 Genre Analysis and Classification 5
2.1.2 Core Design Influences 6
2.2 Motivation 10
2.3 Game Engines 14
2.4 Academic Research and Similar Projects 16
2.5 Development Context 17

3. Workflow, Tools and Implementations 20

3.1 Overview of the Development Pipeline 20
3.2 External Development Environments and Techniques 25
3.2.1 Video and Image Processing Tools 25
3.2.2 Audio Editing and Sound Design 29
3.2.3 Animation and 3D Modeling 32
3.2.4 Programming Environment 35
3.3 Unity Systems and In-Engine Tools 38
3.3.1 Animation and Motion Systems 38
3.3.2 Rendering and Visual Systems 46
3.3.3 UI and Feedback Systems 52

4. Game Design 54

4.1 Overview of Gameplay Pillars 54
4.2 Progression and Level Structure 55

4.2.1 Main Collectibles 55
4.2.3 Level Types and Special Challenges 57
4.2.4 Narrative Progression and Hub World Design 60
4.3 Resource Management 62
4.3.1 Wumpa Coins 62
4.3.2 Mojo Power 63
4.3.3 Ammo 64
4.3.4 Uka Uka Invincibility 65
4.4 Core Combat Mechanics 65
4.4.1 Aim Mode 65
4.4.2 No-Aim Mode 67

5. Game Theory Application 68

5.1 Introduction and Problem Definition 68
5.2 Multiple Objectives and Nash Equilibria 70
5.3 Resource Management Payoff Structures 76

6. Conclusion Remarks and Limitations 83

7. Future Work 84

References 87

Appendix A – Unity Terminology 89

1. Introduction

1.1 Overview of video games

If the reader is unfamiliar with games, one can imagine games as interactive digital systems. To further explain, players learn by doing, receiving immediate feedback, practicing skills in scaffolded challenges using input devices such as controllers, keyboards, or motion sensors. Playing video games provides both entertainment appeal and at the same time educational potential [1-4].

However, it is important to note that perceptions of video games vary. While many recognize their potential benefits, from a non-gamer perspective by focusing only on surface aspects, games could be perceived as harmful and a waste of time. A lot of people fail to understand the underlying benefits. Some of the benefits of gaming are the opportunities provided for differentiated learning [5].

Video games come in a wide range of categories, so learning specific skills depends on the genre of the game being played. Some games don't require much player input serving as mindless entertainment. Others may also have limited gameplay but through cutscenes they deliver engaging stories and meaningful lessons equivalent to movies or series. The platformers genre specifically, which is related to my game, allows people to progress at their own pace and receive immediate feedback. The genres related to my project will be further discussed in detail in the next chapter. Gamers benefit from gaming in their everyday life since video games, when used responsibly, can provide emotional benefits such as excitement and fulfillment.

In children, gaming can have particularly strong effects. A study published in JAMA Network Open (2022) found that children who played action games for three or more hours daily had better impulse control, working memory, and heightened brain activity in areas related to attention and planning [6]. Making this study relative to my game specifically, having to memorize pathways, enemy attack patterns and accurate timing when jumping from platform to platform improve pattern recognition, strategic thinking, decision-making under pressure and develop problem-solving skills.

Figure 1: Some of the most popular gaming platforms.

1.2 Introduction to Game Theory

Game theory is the branch of mathematics concerned with strategic interactions between decision-makers, known as players, where the outcome of a player's choice of action depends critically on the actions of other players. Decision making processes are complicated, but game theory provides tools for analyzing situations of conflict, competition or cooperation. Moreover, game theory has been applied to contexts in war, business, and biology.

Osborne (2004, p. 11) states "the only way to understand game theory's worth is by either seeing it in action or even better setting it yourself in action." According to Martin J. Osborne in *Introduction to Game Theory (2004)*, the foundation of game theory is built on key concepts such as players, strategies, payoffs, and equilibria. A strategy represents a plan of action chosen by a player; a payoff is the outcome or reward resulting from chosen actions. A Nash equilibrium occurs when all players make optimal decisions given the strategies of others, with no incentive to deviate [10]. In Chapter 5, these principles will be applied to player strategies such as speedrunning vs. completion and resource allocation choices.

1.3 Structure of Thesis

This thesis carries a dual identity. On the one hand, it is a deeply personal expansion of a childhood game demonstrating how individual passion can drive rigorous academic work. On the other hand, it is a structured research investigation into resource management, decision-making under uncertainty, and Nash equilibria within game environments. The reader not only sees the academic analysis but also appreciates the game's design as both a personal creation and a system to be optimized.

Chapter 2 provides an overview of the related games, motivation and academic research leading to the creation of this project. This chapter also documents in detail the realization of the project and explains the reason behind choosing Unity 6 as the game engine. Before moving to the next chapter, a comparison between one-person game development and professional studio game development is demonstrated.

Chapter 3 presents the procedure of the game's development. It addresses all the different tools and software used for asset, animation and gameplay systems creation. Specifically, it covers the entire pipeline from initial visual and audio asset production to fully integrated game elements in Unity. It also outlines the use of AI-assisted tools and the step-by-step process of building functional levels and gameplay systems.

Chapter 4 focuses on the gameplay and game's content. The core gameplay systems, resource management, main collectibles, level types, hub world and progression will be examined from the perspective of the player. The investigation of game design serves as a foundation for understanding how gameplay systems and player decisions interact.

Chapter 5 applies game theory to model player behavior and identify optimal strategies for completing levels in the most efficient manner. Resource allocation problems are modeled as strategic interactions, with concepts such as Nash equilibria and payoff structures used to evaluate the fastest and most efficient paths.

Chapter 6 concludes with some remarks and reflects on limitations.

Chapter 7 proposes future improvements and additions.

2. Background and Related Work

2.1 Inspiration and Related Games

The development of the project is based on well-known video game franchises. After getting involved with the aforementioned franchises and delving deep into their world, I familiarized myself with all aspects of those games. *Figure 2* depicts most of the main titles that influenced my game *Pinstripe Potoroo: The Dark Gem Conspiracy*.

Figure 2: Inspirational game titles for project development.

2.1.1 Genre Analysis and Classification

Pinstripe Potoroo: The Black Gem Conspiracy could be characterized as a 3D Platformer/Collectathon and Third Person Shooter hybrid. It's enriched with Action-Adventure elements through its Exploration (Hub World) and story-driven level progression.

Platform games often referred to as platformers, are games in which Characters and settings are seen inside view as opposed to top view, thus creating a graphical sense of "up" and "down" as is implied in "Platform" [7]. Collectathon is a sub-category of 3D Platformers focusing on item collection within enclosed levels [9]. Adventure games refer to games which usually have in common a story that drives the gameplay. "Adventure" is usually associated with a quest-like structure [7]. In a Third Person Shooter the player can see the in-game representation of himself perform various actions such as running, jumping, reloading, and firing weapons [8].

Some of the games that I have taken inspiration from are already hybrid games themselves. Except the *Crash Bandicoot N Sane Trilogy (2017)* and the original PlayStation 1 counterparts, as well as *Crash Bandicoot the Wrath of Cortex (2002)* and *Crash Bandicoot 4 It's About Time (2020)*, which are part of the Platform genre, video game titles like *Jak II (2003)* are a hybrid of Platformers, Third Person Shooters and Action-Adventure games. *Jak II* is a mission-based game that combines difficult platforming challenges with shooting and combat sections. Games like *Crash of the Titans (2007)* and *Crash Mind Over Mutant (2008)* are combining Platforming with a Beat' em Up style of gameplay which was taken into consideration while developing my game's combat sections. A Beat' em Up game is based on hand-to-hand combat between the player and many non-players characters. Games like *Crash Tag Team Racing (2005)* are considered a hybrid of Platform, Kart-Racing and Adventure games. Racing games emulate driving a car on a racetrack. [7] My video game will have some racing sections on some levels but that's not enough to consider it a kart racer.

Figure 3: Genre classification of video games influencing my project.

Figure 4: Genre classification of Pinstripe Potoroo: The Black Gem Conspiracy.

In *Figure 3* and *Figure 4* are the genre structures of video games. The main difference between *Jak II* and *Pinstripe Potoroo: The Dark Gem Conspiracy* is that in *Jak II* at the start of the game, the player has some basic attacks while guns are later acquired as an upgrade after some missions. On the contrary, in my video game the gun is part of the game from start to finish and is the most well-rounded offensive strategy in the game. Furthermore, in *Jak II* the platforming compared to the combat is not as challenging while the action happening is way more intense throughout the game as enemies appear consistently. Clearly combat is more of a focus comparing it to my game. That is the reason why I consider my game mainly a Platformer and Third Person Shooter hybrid.

2.1.2 Core Design Influences

In general, video games as mentioned earlier can be classified into a variety of genres. They are divided based on their player objectives, gameplay mechanics, 2D or 3D environments and interaction styles. Nowadays, a lot of games are hybrid versions of different genres to keep things fresh and interesting. Even though there are a lot of successful games that stick to one genre, my game is inspired by a variety of different games that share similarities while being their own games with their own identity. The following paragraph will analyze a list of games that influenced this project.

Let's start with the main inspiration for my game which is the whole Crash Bandicoot Series. I have taken inspiration from the characters, story and levels. So, I do have a list of potential levels in my game which are influenced heavily by a lot of Video Games.

Figure 5: A big portion of Crash Bandicoot characters including my main protagonists Pinstripe and Tawna.

The level design of each level is original for the most part to accompany the unique style of gameplay I created in my game. As a base though I have been inspired by a lot of different parts of existing levels. For instance, in *Figure 6* the level called Colour Agony is inspired heavily by Ant Agony, a level from the game *Crash Twinsanity (2004)* and in *Figure 7* the level called Bonsai Ruins is inspired heavily by Banzai Bonsai a level from *Crash Bandicoot the Wrath of Cortex (2002).*

Figure 6: Comparing in game footage of the Original Ant Agony level design and its adaptation in Colour Agony.

Figure 7: Comparing in game footage of the Original Banzai Bonsai level design and its adaptation in Bonsai Ruins.

Other games I have taken into consideration for level environments and level design are: Most of the Crash Bandicoot series, *GTA: San Andreas (2004)*, *The Jak and Daxter trilogy (2001-2004)*, *League of Legends (2009)*, and some series from my childhood like a Hi-5 inspired level. Hi-5 is an Australian preschool television series. In *Figure 9* some of the concepts for future levels are depicted.

Figure 8: Some of the concept art I had made for future levels inspired by other games.

For the unique style of gameplay, I have combined aspects of the titan games "combat system and mutants" specifically *Figure 9* shows clearly the integration of the stun indicator in my test level, the platforming style of the classic Crash Bandicoot games and similar main collectibles such as Gems, Crystals and Relics, the third person shooter aspect from games like the GTA series and the latter two entries in the Jak and Daxter games which also inspired the combination of platforming and shooting.

Figure 9: Early development of the Combat System Inspired by Crash: Mind Over Mutant.

The clear difference between the two is that in *Crash: Mind Over Mutant* to fill the stars the player must perform various hand-on-hand attacks since it's mainly a Beat' em Up game. While in my game's case the player must shoot the enemy mutant to fill the stars. When the stars are filled in *Crash: Mind Over Mutant* the enemy is stunned, giving players the option to mount them or finish them off. In *Pinstripe Potoroo: The Dark Gem Conspiracy* a bomb icon replaces the stars. If the player doesn't finish them off with a bomb, the mutant will get out of the dazed phase and will be able to attack the player again until the star bar is filled once more. This cycle ends only when they get bombed while stunned. In *Figure 10* even though my game is in an early state, the difference between the two systems is quite evident, demonstrating how the inspiration was integrated into the new gameplay framework.

Figure 10: A visual comparison between the original stunned mutants and the early development state of my integration.

The idea of progression is quite like *Crash Bandicoot 2: Cortex Strikes Back (1997) and Crash Bandicoot: Warped (1998)*, meaning that there are in the main hub area five levels which the player must complete (get at least the main collectible which is the Power Crystal) so that you can proceed with the boss's fight. Defeating the boss leads you to the next area and that repeats until you reach the final boss. The warp room is replaced with Crash Tag Team Racing's MotorWorld which is closer to the *Spyro the Dragon (1998)* hub worlds meaning it's more of an exploration area until you reach the next level portal area. That is totally unique and original combining very different ideas of level progression. Unfortunately, the hub world is still under development and the only art that can be shown is the combination of the game's ideas that have merged in *Figure 11*. So, the concept is to have portals inside the original spot of the race portals that areas of the MotorWorld had in Crash Tag Team Racing, with buttons numbered like in Crash Warped which if jumped on will enable portals. The portal that is open will be random shapes that are shaped from intense electricity and a moving picture of the overview of the level. Close to the electricity the player will be absorbed and taken to the level. The player needs to find switches throughout the hub world that will unlock each level.

Figure 11: Concept art of a hub world that incorporates elements from different Crash Bandicoot games.

In *Figure 11* at the top left is a part of the warp room from the remade version of *Crash Bandicoot: Warped (2017)* and bottom left is a part of the MotorWorld from *Crash Tag Team Racing*. In the far left a small screenshot is visible of how the main collectibles are shown in the remade version of *Crash Bandicoot 2: Cortex Strikes* Back *(2017)*. I will integrate it into my game where you can see an altered version of the MotorWorld where elements from all three games are implemented. The MotorWorld is filled with collectibles that are retrievable through completing each level. One collectible which is absent all together from the original Crash games is the Voodoo Dolls which are inspired by Titan games as shown in the far-right down corner a screenshot from *Crash: Mind Over Mutant*. They will also be retrieved while playing levels through different challenges.

2.2 Motivation

The initial motivation for developing *Pinstripe Potoroo: The Black Gem Conspiracy* dates back to 2015. The project was initially named *Crash Bandicoot 1.2: Pinstripe's Revenge* and was created during a period when the Crash Bandicoot franchise had entered a long hiatus with no major titles being released. The last significant entry at the time was *Crash: Mind Over Mutant (2008)*, and for several years, fans of the series were left without new content. Of course, at the time many fan-made Crash games popular within the fandom such as *Crash Bandicoot: Crystal's Wrath (2013)* were created by fans whose projects demonstrated creativity and passion. The lack of new Crash games and seeing similar projects being created by the community were enough reasons to motivate the creation of my first project.

From the start, I envisioned a game that expanded beyond the traditional Crash Bandicoot formula. I wanted to be able to control other characters of the series since there is multiple to choose from like depicted in *Figure 5*. I decided to go with Pinstripe Potoroo since he was absent from later installments of the series. I wanted to extend his lore that was once mentioned by developers and fans. When in the original *Crash Bandicoot (1996)* Crash Bandicoot defeated in a boss fight Pinstripe Potoroo, the only mention of his lore was in the *Crash Bandicoot 2: Cortex Strikes Back (1997)* manual, which explains that Tawna left Crash to be with him, excusing at the same time Tawna's absence from later installments. The reality was that publisher, Universal Interactive Studios, and Naughty Dog disagreed on her design, with Universal finding the original design of Tawna too "inappropriate". Inspired by this narrative, I had decided to expand upon it, exploring what exactly happened to Pinstripe and unfolding his story.

The initial version of the project ten years ago laid the foundation for the current iteration of the game. The early project was created with Unity version 5.6.1f1 which seemed like my only option for a game engine at the time to start experimenting with game development. I used to watch a lot of YouTube tutorials and reddit posts on how things work in the world of game designing and developing. I came across Blender, a professional, free, and open-source 3D creation suite used by artists and designers for modeling and other related actions. I also learned about programming, specifically JavaScript which is a programming language supported by the version of Unity I was using and some C++ which is still supported by the most recent version of Unity. Utilizing a very big old PC and having an extremely basic screenshot tool I used to take screenshots of level gameplays shown on YouTube editing them to my desired measures and filters, then importing them into my game and turning them into textures. I also used some Google pictures, downloading them to the PC and creating textures out of them. I utilized blender to create skeletons for my characters and created most of them from scratch like the ones seen in *Figure 13*. I did come across Pinterest and other sites that did share rigged models of existing characters, I used the *Crash Twinsanity (2004)* rigged model of Pinstripe, imported it to Blender made a skeleton for it created animations and made new textures for it when I imported it to Unity. Another imported model is Grimly from *Crash: Mind Over Mutant* which is the model in the top left corner of *Figure 12*. I created a skeleton and with it created animation. The rest of the models in *Figure 12* are designed and textured by me. The big clock model I constructed ten years ago in the bottom right area of *Figure 12* is reused with different textures, size dimensions and lighting effects in the new version of the game.

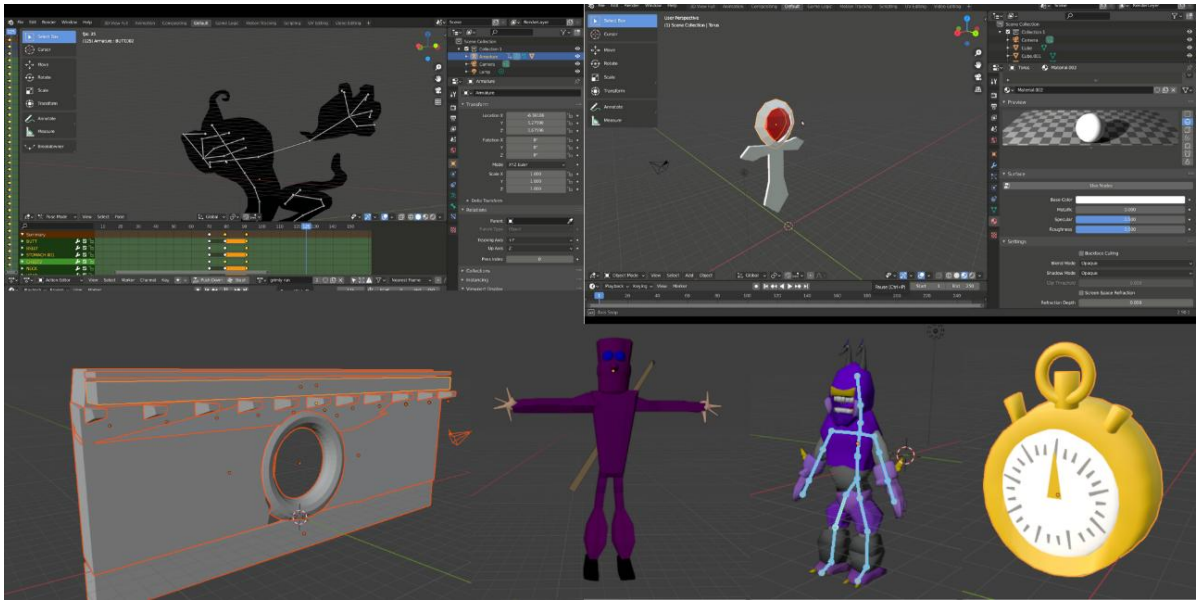


Figure 12: Development of Game Assets: Characters, Environment Elements, and Collectibles created using Blender in 2015.

As for the game's soundtrack, music was used from videos of a YouTube channel under the name MsDaBoss7 and converted them into mp3. The channel exists still to this day and is run by a female remix artist, who makes Crash Bandicoot remixes and in general remixes of music that are used in official Video Game titles. The music was tweaked a bit adjusting the pitch and speed using free software.

Generally, it took up to thirty minutes for the old PC to load a single scene in Unity. If a new animation for my models in Blender was necessary, a waiting period of roughly five minutes was required for an animation to be saved. Every single game test needed 5 minutes of loading time. In *Figure 13* and *Figure 14* there are screenshots of the same levels made in both games (to the left is my first project *Crash Bandicoot 1.2: Pinstripe's Revenge*, and to the right is *Pinstripe Potoroo: The Black Gem Conspiracy*).

Figure 13: Bonsai Ruins Crash Bandicoot 1.2: Pinstripe's Revenge (2015) Pinstripe Potoroo: The Dark Gem Conspiracy (2025).

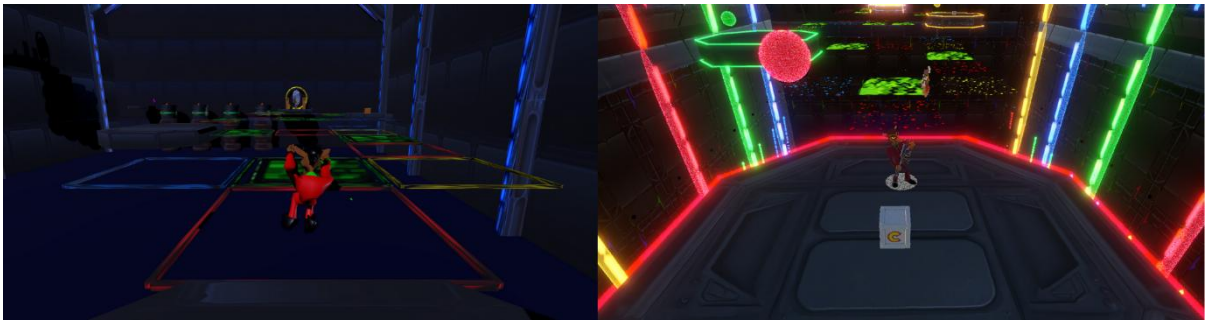


Figure 14: Colour Agony Crash Bandicoot 1.2: Pinstripe's Revenge (2015) Pinstripe Potoroo: The Dark Gem Conspiracy (2025).

The game's development ended in 2016 since I didn't have enough knowledge of complex programming, and my designing skills were limited. Another reason that I gave up on it was that I used to spend whole days developing that game, so I stopped due to other priorities like school. Moreover, it was affecting negatively my psychology because as a young boy living in a small island in Greece, there were not many people taking interest in my game since Crash Bandicoot was not so well recognized where I lived. Finally, in 2016 it was announced that Crash Bandicoot was officially back with a brand-new game, a full remake of the original trilogy, released as *Crash Bandicoot N. Sane Trilogy (2017)* giving me excitement and enough satisfaction to say goodbye to my game.

Fast forwarding about seven years later in late 2022 I started using RaveDJ, an automatic online mashup creator, to create some mashups of my favorite songs for fun. Then out of curiosity I made some mashups of Crash Bandicoot songs just to see how they would sound, and I was pleasantly surprised. I continued and made several songs that I thought could really fit my old game very well. I then created a folder containing a lot of music. That music really inspired me to create very rough new concept with Paint to recover old and create new ideas I had for levels.

I wanted to recover the old game, but its files were stored on that old PC nobody was using, and it was on a different island from where I was. There was also no motivation to continue my old project since I hadn't really been involved with game development at all during all these years. If I wanted to make my game, relearning everything was the only option.

In 2024 when I was working on several projects some of which involved a certain level of programming, with the power of AI technology I realized that I could program anything I wanted and learn everything I need to know to create my game. I did participate in a bonus presentation my university offered, specifically Mr. Ioannis Marinakis and Mrs. Magdalini Marinaki in the 9th semester logistics course. There I showcased my old project and my professors suggested turning the game into my diploma thesis, enhancing it further by employing optimization techniques and game theory.

My thought initially was to use the first version of my game as a foundation. Even though I managed to regain access to the old PC and retrieve some files, I was unsure which were necessary to successfully migrate my project to another system. As of now, after research I have identified the exact files necessary for recovery and realized the version of Unity is important to consider too. At that time, however, my option was to create a new project expanding it through the application of optimization techniques and game theory concepts.

Thanks to the major success of *Crash Bandicoot: N. Sane Trilogy (2017)* selling over 2.5 million copies in the first three months, there was a lot of discussion surrounding new Crash Bandicoot games and the current publishers of the franchise Activision were announcing new titles almost every year. After the cancellation of Crash Bandicoot 5 reportedly owing to the disappointing sales of *Crash Bandicoot 4: It's About Time (2020)* there have been no indications for upcoming entries in the franchise. This situation is highly reminiscent of the circumstances in 2011, since there were no official Crash Bandicoot title releases until 2017. Acknowledging the presence of *Crash Team Rumble (2023)*, in 2026 close to three years will have passed since the last entry, motivating me enough to consider publicly releasing my project on platforms such as YouTube, gaining attraction from fans of the franchise who need brand new Crash Bandicoot content.

This chapter is written not only to situate the project in the context of existing research and game development practices but also to highlight its personal significance. Unlike a purely technical review, it interweaves academic references with the inspirations that shaped my project over many years. By presenting both scholarly and personal influences, the chapter demonstrates how a deeply personal passion project can evolve into an academically rigorous thesis, showing the link between individual motivation and formal research contributions.

2.3 Game Engines

Game engines are software frameworks designed to facilitate the creation and development of video games by providing a suite of tools such as rendering engines, physics simulations, animation handling, audio systems and scripting interfaces, allowing game developers to focus only on gameplay design [11]. They are used not only in the game industry but also in engineering and design [12]. Some of the most popular game engines include Unity, Unreal Engine and Godat.

Particularly, Unity 6, officially released in October 2024, and Unreal Engine 5, officially released in April 2022, were examined as possible platforms for my project. While researching the best option, I came across an abundance of publicly available documentation of their mechanical aspects and capabilities.

Figure 15: Unreal Engine 5 (left) Unity 6 (right).

Unreal Engine is known to be used by developers from both industry-leading video game companies and independent or small teams. The utilization of Unreal Engine is not restricted to video game development. For instance, ESPN, a major broadcast network, hires teams that use Unreal Engine to create real-time graphics and virtual production for sports, concerts and festivals. Some animations for film and television are built with Unreal. Notably, *The Mandalorian (2019)* utilized Unreal Engine from its first season for real-time rendering of complex visual effects. An alternative application of Unreal could be the integration of CAD files. Particularly, it uses its Datasmith technology and the Unreal Studio suite to import complex CAD data. Unreal can create 3D representations and renderings making it applicable to industries like product design, architecture and manufacturing. For example, in Unreal rendering a design for a new car is possible. That 3D model of the car could be used as a digital asset for customers to explore virtually on the company's website. Unity has other potential use cases other than strict game development as well. In addition to immersive and visually compelling games, Unity's engine with the use of its plugins can potentially be used to solve all types of problems that were approached using Unreal Engine.

Unreal Engine 5 is optimized to be visually better than Unity 6, supporting high-fidelity graphics and physic-heavy environments, at the cost of file sizes being larger on average. This justifies why AAA game titles and architectural visualizations adopt Unreal. In general, Unity is considered the more user-friendly option because Unreal Engine's impressive features can be more challenging to learn. Unity has a robust community of users and support available online that can help beginners start creating 3D and virtual experiences. A big difference between Unity 6 and Unreal Engine 5 is that the Unity editor uses the C# programming language, and the Unreal Engine editor uses a combination of two programming languages C++ and Blueprints.

Taking all of this into consideration, several indicators influenced my decision to select Unity 6 over Unreal Engine 5. Photo-realistic graphics were not a priority for my game, as its visual style is mainly cartoony with only slight elements of realism. My prior experience with Unity 5.6.1lf1 was the main indicator of my decision. Unity 5.6.1lf1 supported not only C# but also JavaScript. Unfortunately, Unity discontinued support for JavaScript in 2017, making my decision of choosing JavaScript, purely for its simplicity, over C# not beneficial in the long term. Ultimately, the two main reasons for preferring Unity 6 would be the prior core Unity component knowledge and the manageable learning curve.

2.4 Academic Research and Similar Projects

While researching previous diploma theses from students at the Technical University of Crete, I came across one titled *"3D Game Development with Optimized Game Design"*, by Malkogiannis Konstantinos (2015, Department of Electrical and Computer Engineering). This game was developed using Unreal Development Kit (UDK) which was a free version of Unreal Engine 3 toolset. UDK was released on November 5, 2009, and allowed students, indie developers and hobbyists to create 3D games and visualizations before Unreal Engine 4 was released. Even though the relevancy to my project is apparent, game development has changed dramatically since then. For instance, UnrealScript, which is the programming language used in this case, is no longer supported; physically based materials, dynamic global illumination, and virtualized geometry are modern rendering techniques unavailable in UDK.

One academic project that covers the development of a 2D platformer in Unity 6 is titled "*Platformer Game from Concept to Working Prototype"*, by Anton Ivanov (2025, Engineer (University of Applied Sciences), Industrial Information Technology). It includes a chapter dedicated to the technical part of making a game, analyzing Unity 6's game development tools. Generally, it has a very good structural model and shares similarities with the next chapter.

There's a methodological framework paper titled *"The Game Between Game Theory and Gaming Simulations Design Choices"*, by Roungas Bill, Bekius Femke, Meijer Sebastiaan (2019). This article proposes a framework for formalizing, and consequently standardizing, expediting and simplifying, the modelling of gaming simulations. The proposed framework applies game concepts pertaining to game theory in the abstraction of the real system and the game design decisions. Application of the framework in three case studies reveals several advantages of incorporating game theory into game design, such as formally defining the game design elements and identifying the worst-case scenarios in the real systems, to name but two [14]. For my project, it's highly relevant to the fifth chapter, legitimizing my modeling choices.

One more relevant source bridging formal game theory with practical game design is an academic conference paper titled *"Game Theory and Video Game: A New Approach of Game Theory to Analyze and Conceive Game Systems"*, by Guardiola and Natkin (2005).

This project is a testament to my ability to rapidly acquire new skills. Through my research of various projects, comparing old projects to new, there was a realization about our world where specific knowledge quickly becomes obsolete. Therefore, adapting as fast as possible to changing conditions is what I deem most significant.

2.5 Development Context

Game development takes a lot of patience and skill due to several reasons. First, a game needs a concept, which is a blueprint for what the project will become. Secondly, building game logic requires time, programming skills and knowledge of core game development principles. Thirdly, every game needs a visual aspect, which is represented in the form of images, models and animations. Those components need to be produced by hand according to the chosen style and theme of the product. Finally, a game requires music, which must be fitting and be as active as it is subtle not to divide too much attention from gameplay [13].

Prior to proceeding to the next chapter, a short comparative analysis will be conducted between the professional studio workflow and solo development pipelines. This project was developed entirely by a single individual. By contrast, in a professional game development studio, tasks are distributed among specialized teams. *Table 1* was formulated to demonstrate in greater detail the different aspects of production.

Figure 16: Solo Developer VS Professional Studio.

Table 1: Comparison between Professional Studio Workflow and Solo Developer Workflow

Aspect	Professional Studio Workflow	Solo Developer Workflow

Team Structure	Defined roles such as designers, programmers, artists, sound engineers, writers, and QA testers are given to people in large, specialized teams. There are roles specifically designed to coordinate those teams managed by producers and project managers.	All roles are assigned to a single individual. As a result, time and energy must be carefully allocated to balance all aspects of game development. A multi-disciplinary and variety of knowledge is required.
Project Management	Documentation tools such as Agile are utilized with regular sprint reviews, milestones and dedicated scheduling tools. Formal project management methodologies are enforced.	Planning without strict deadlines. Personally, using note-taking applications particularly OneNote, text editor Notepad and my phones generic Notes app. Development is adaptive, since new inspirations or challenges arise.
Tools and Software	Licensed software suites, enterprise-level. Key examples are Autodesk Maya and 3ds Max for 3D animation and modeling and 3D painting software, Substance Painter, for texturing. Everything is optimized for workflow efficiency.	Tools are selected based on availability, learning curve and cost-efficiency. Personally, I used a combination of accessible tools and open-source software such as Unity, Blender, Meshy AI and Audacity.
Asset Creation	Departments are provided for modeling, animation, rigging, texturing and sound. Each asset passes through a review pipeline to ensure quality and style consistency.	Assets are created manually or semi-automatically by one individually. AI tools assist rigging, modeling and animation to save time.
Testing and Quality Assurance	Performance validation across multiple platforms, usability testing and bug tracking follow formal protocols and are executed by structured QA teams.	QA is achieved through playing the game by either the developer, online communities or the developer's peers. Occasional informal feedback is received to fix overlooked issues.

Version Control	Perforce and Git are Centralized Version Control Systems (CVCS) which allow many team members to track and manage changes to files over time. Stable game builds are maintained by utilizing branching strategies and automated build systems.	Minimal version control. Changes are often monitored through local backups and manual tracking. Fewer collaborators reduce the need for complex versioning systems.
Workflow Organization	Clearly defined pipeline stages: concept → pre-production → production → QA → release. Dependence is carefully managed between departments.	Flexible, nonlinear workflow. Immediate needs or new creative concepts might shift the developer's task focus. Personally, switching constantly between creating new concept art, programming, game design, music and sound effects was common.
Production Scale	Even before the stage of the concept, marketing, localization and long-term support are already built into the plan. That plan is generally not fixed in a contract, especially with longer-term support. Generally large-scale games have budgets ranging from hundreds of thousands to millions of dollars.	No marketing budget. The project's scope is defined entirely by the developer's available time and resources.
Knowledge Sharing	Knowledge sharing supports collective innovation and relevant documentation. Some instances are technical showcases presenting solutions to complex problems and sessions of collaborative coding where developers compete on programming puzzles, fostering a fun and engaging way to share and apply coding skills.	Knowledge is more personal and informal. Personally, solving problems relied purely on self-documented notes, online tutorials, and now with the rise of AI technology solutions to most problems were proposed by AI-based support tools like ChatGPT with accurate text input.

3. Workflow, Tools and Implementations

3.1 Overview of the Development Pipeline

As described in Section 2.5 the development process of this game could be characterized as non-linear and highly iterative. The asset creation process has evolved drastically, especially when comparing to the previous attempt in the older project and the earlier stages of this game. Becoming more intricate over time, the process composed primarily of a lot of experimentation with different tools and techniques to determine what worked best. These experiments have been refined into a structured workflow, representing a linearized version designed for documentation purposes. Particularly, these workflows have been organised into four main categories, allowing for a coherent and easily reproducible process for all the different aspects of the game's creation pipeline.

Figure 17 depicts an overview of the general process of asset creation. The procedure starts with either concept art, screenshots or footage from other games. Depending on the asset, if it is of simple geometry, a *GameObject* (see Appendix A) in Unity is created and an imported edited image is used as a material, e.g., the jet platforms. If it's not of simple geometry which is most of the case, I will use either Blender to create the object from the ground up, e.g., the big old clock or turn the image into a model with one click using MeshyAI, e.g., the ant enemy. Another case is finding a model from a previous game and rigging it through a different procedure, e.g., the Iron Checkpoint. When the model is created, if its AI generated, I will use Blender for cleanup, removing any unnecessary Faces and Vertices. The following step is, importing it to Unity. There, C# scripts are written using Visual Studio Code to define specific behaviors combining them with other components like Animators and Colliders. For the animators, animator controllers are created and animations within them. The animations are either developed in Unity using its in built Animation tool or imported from Blender and MeshyAI. Particle systems for visual effects are often designed for the GameObject as well as shaders and materials. Performance settings like lighting or even mesh and particle system simplifications may be required for the game to maintain an optimized performance. If the developer intends to apply the asset multiple times, the asset could be made into a *prefab* (see Appendix A) to keep track of changes. Finally, it is ready to be used within a level.

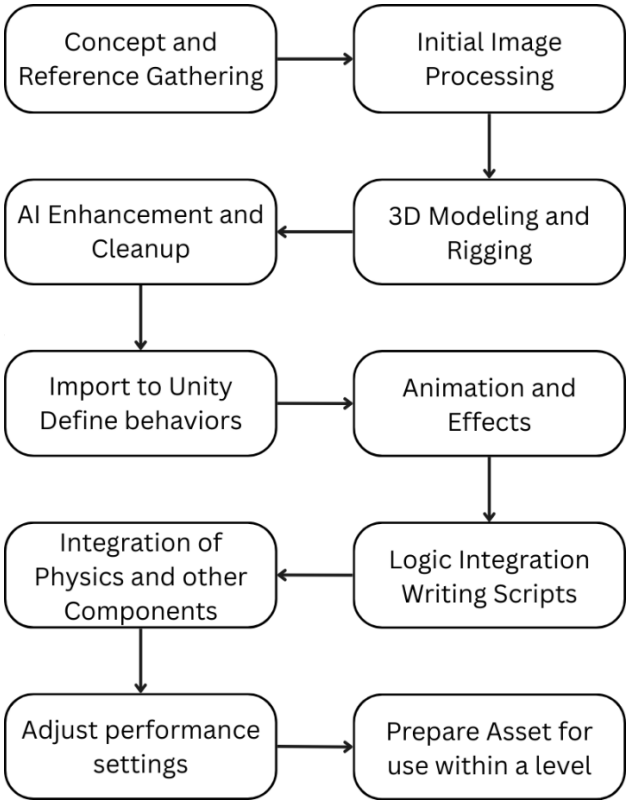


Figure 17: Flowchart of the Asset Creation Pipeline.

Figure 18 illustrates the workflow steps of User Interface (UI) 2D asset creation. First, the content that is required to create the GameObject is recorded with OBS or similar software. The content might also be an existing GIF downloaded from Google. After securing the necessary material, they are transformed into individual frames using software like ezgif. Cleaning each frame then proceeds; the frames are in image format and cleaning involves removing backgrounds using remove.bg and enhancing image quality with Img.Upscaler. The proceeding phase includes combining all frames into a single *spritesheet* (see Appendix A) using the Leshy Spritesheet Tool. The spritesheet is imported into Unity and through the *inspector* (see Appendix A) import settings are changed. The texture type should be assigned to Sprite (2D and UI) and depending on the use of the *sprite* (see Appendix A), changing the sprite mode is followed to single or multiple. Unity recognizes each frame individually if multiple is chosen and can be edited with Unity's Sprite Editor tool. Thereafter, frames are arranged into an animation with Unity's Animator, making the UI behave like a video while being performance friendly. Following this, the UI asset is ready to be assigned to menus, HUD elements or interactive UI systems.

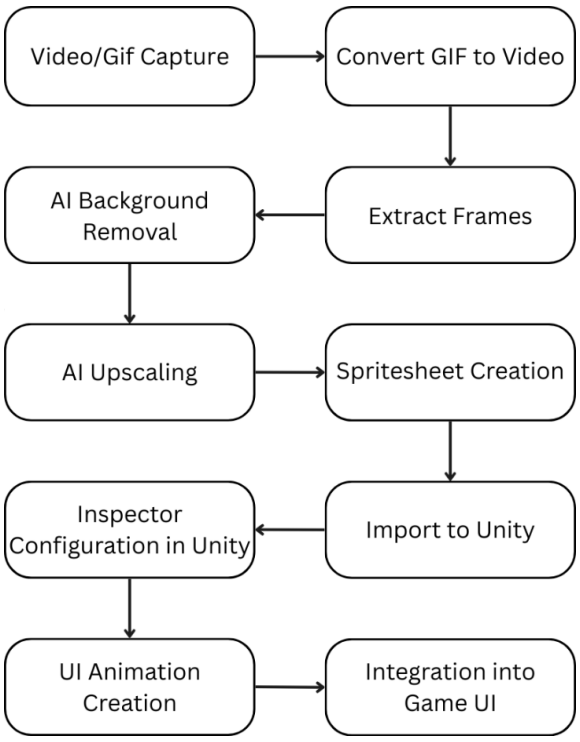


Figure 18: Flowchart of the 2D UI Asset Creation Pipeline.

Figure 19 focuses on the creation process of the player character Pinstripe, as a detailed example of how complex assets are being developed. The Pinstripe character model was something that changed throughout the development process. The final version of the model is a modified version of the *Crash Team Racing Nitro-Fueled (2019)* Pinstripe model. The game's *rigged* (see Appendix A) models were either downloaded from Steam → Source Filmmaker → Workshop or from Deviant Art where users can find MikuMikuDance Models (MMD) converted from *Crash N Sane Trilogy (2017)* by SAB64. Downloading MMD plugins or any other necessary addons, the following step was importing the model into Blender. Blender proved effective when editing the existing animations that came with the rigged model and making new ones. Exporting the model to FBX format and importing it to Unity, the primary step is to structure the character into *parent* (see Appendix A) and *child* (see Appendix A) GameObjects. For the parent object a personal decision was made to attach all physic related *components (see Appendix A)* and scripts while the child object "PinstripeVisuals" contains as the name suggests the armature, materials and animations of the character. Adding a script to link the visual representation of the characters actions and other components like an animator controller to support the Blenders animations. The animation editing is done in the inspector of Unity, specifically deciding to loop the animation or adjusting its speed. Other child GameObjects are created, e.g. GroundDetector, to accompany the actions that are written in the parent's scripts or add more visuals to the character depending on the context. The concluding step involves creating a prefab of the parent GameObject, for the purpose of future editing.

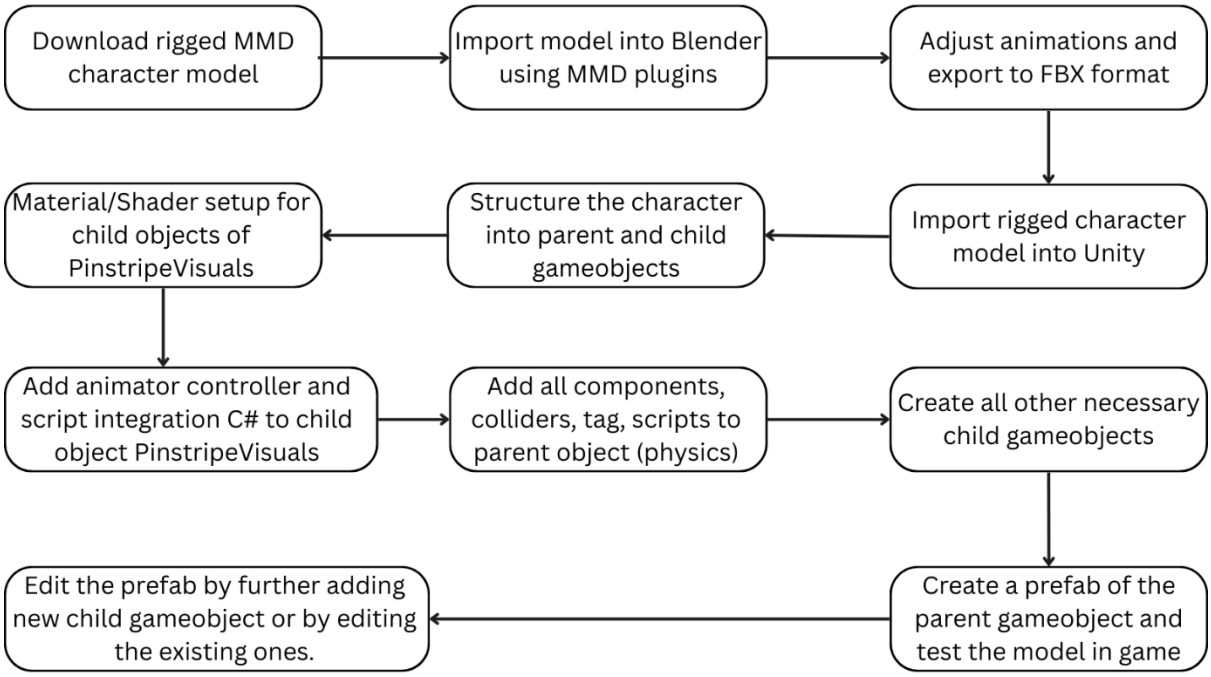


Figure 19: Flowchart of Pinstripe Character Creation.

Finally, in *Figure 20* the last workflow involving level creation is presented. More specifically, it demonstrates the assembly of the created assets into functional game levels. Before starting with the creation of the level, a level theme, its gameplay goals and narrative role need to be defined. For most of the level concepts, a combination of rough sketches, google images and text are assembled with Paint in order to illustrate important level information. Conceptualization is of great importance to pre-level creation. Since the project is being developed in Unity 6, creating a scene is the initial step. Unity 6 recognizes levels as scenes with assets being dragged from the project window into the scene window. The prototype layout is created from research, references and the rough concept art previously designed. At a later stage, using optimization principles to ensure balanced difficulty and correct resource distribution, crates, enemies and other collectibles are placed in the level. By playtesting the level, the layout of the platforms might be altered over time to match the game's physics and improve player experience. A list of bugs and improvement notes is also compiled for further improvements. Other things to consider optimizing are the lighting settings, particle effects and camera adjustments. The game runs using Universal Render Pipeline (URP) lighting settings. Those settings are adjustable and directly affect the atmosphere, style and overall look of the game. Fulfilling the criteria, a fully playable prototype is built utilizing the *build settings* (see Appendix A) and is ready for re-evaluation.

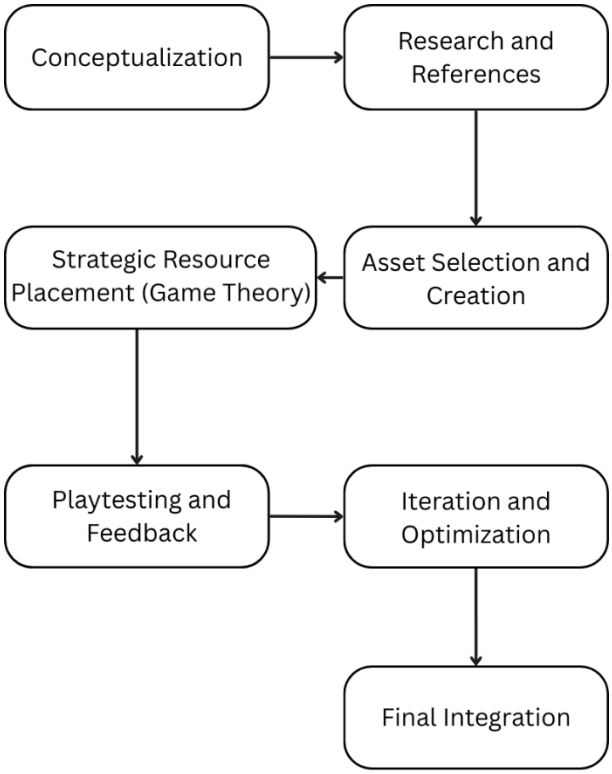


Figure 20: Flowchart of Level Creation.

Figure 21: Evolution of Colour Agony's Level Creation.

3.2 External Development Environments and Techniques

The software, systems and techniques are explored thoroughly in this section. Each category of tools, which was previously mentioned as part of the workflows, has its own dedicated subsection. Those tools are utilized to prepare, create or edit assets and logic before they are integrated into the Unity environment.

3.2.1 Video and Image Processing Tools

Tools namely OBS Studio, Xbox Game Bar, Canva, Shotcut, Ezgif, Remove.bg, Img.Upscaler, Leshy Spritesheet Tool and ChatGPT are used for raw asset extraction and preparation.

OBS Studio and Xbox Game Bar were used for screen recording gameplay footage. For instance, to create the HUD element of the crate counter, a video of it rotating was captured. The scene's view was used to get the best angle of it. Another application, this time Xbox Game Bar's video capture widget, was the recording of Crash Twinsanity's Ant Agony gameplay running on PCSX2 in the best video quality possible. PCSX2 is a PlayStation 2 emulator. The footage was then edited with Canva.

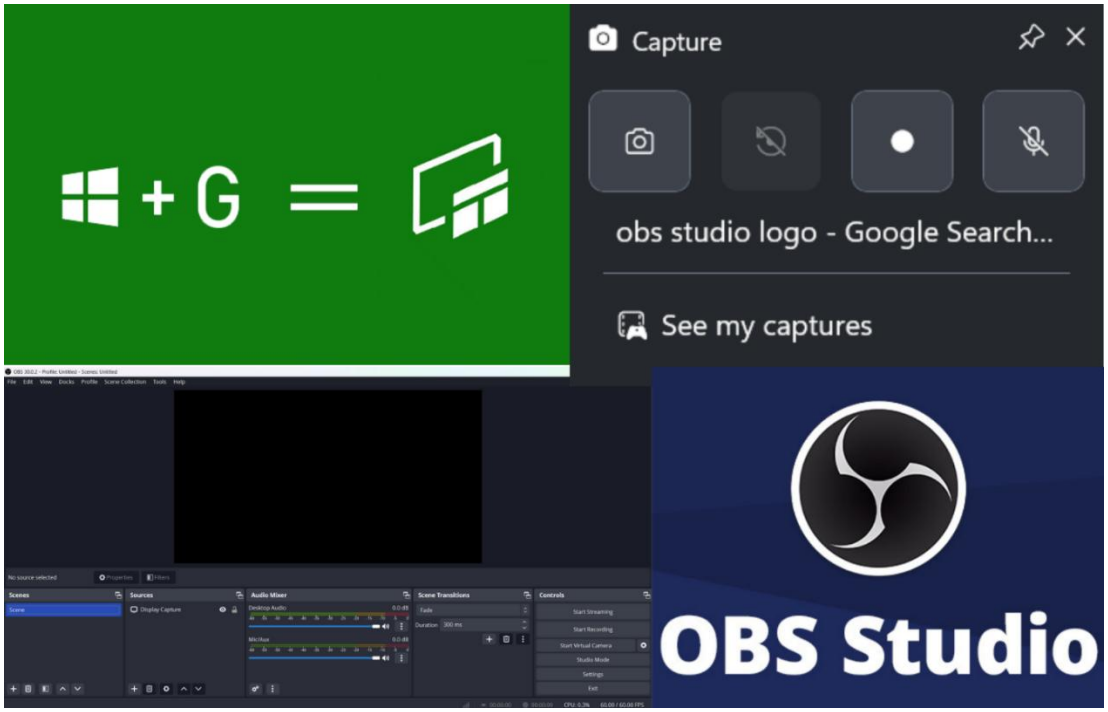


Figure 22: OBS Studio and Xbox Game Bar Interfaces and Logos.

Canva is a free to use online graphic design tool. Probably the most versatile and easy to use tool with many different features that are easy to learn. One of its applications was to resize the frame size of the videos, before editing them in Shotcut. Another application was changing the colors of the game's textures. Combining with an AI Image Upscaler clearer textures were created. The tool was utilized to create brand new textures as well, like the textures used for TNT and Life Crates.

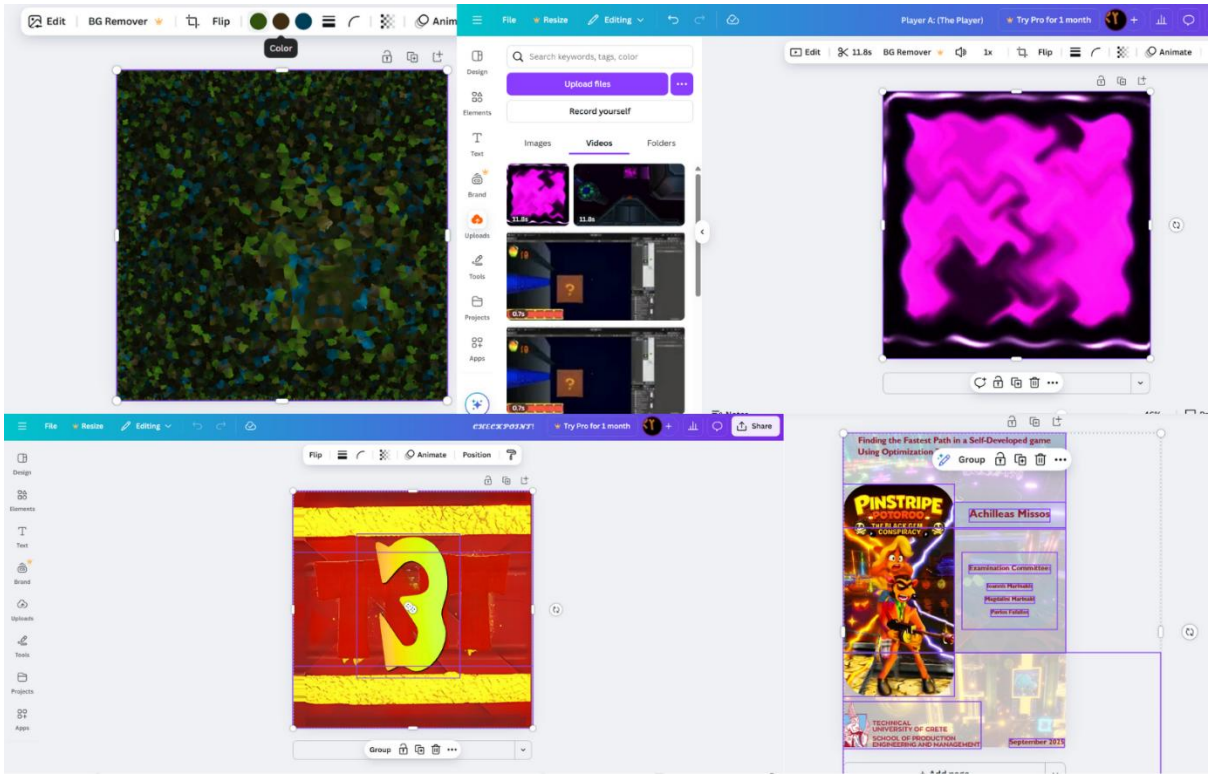


Figure 23: Some Examples of Canva Applications.

Shotcut is a video editor available for Windows. It's designed for users of all skill levels making it a perfect choice for editing video captures. A clear illustration of its utility is all the color time-based platforms, which were originally video captures edited in Canva. In Shotcut a glitch effect, other effects and color corrections were applied to the videos.

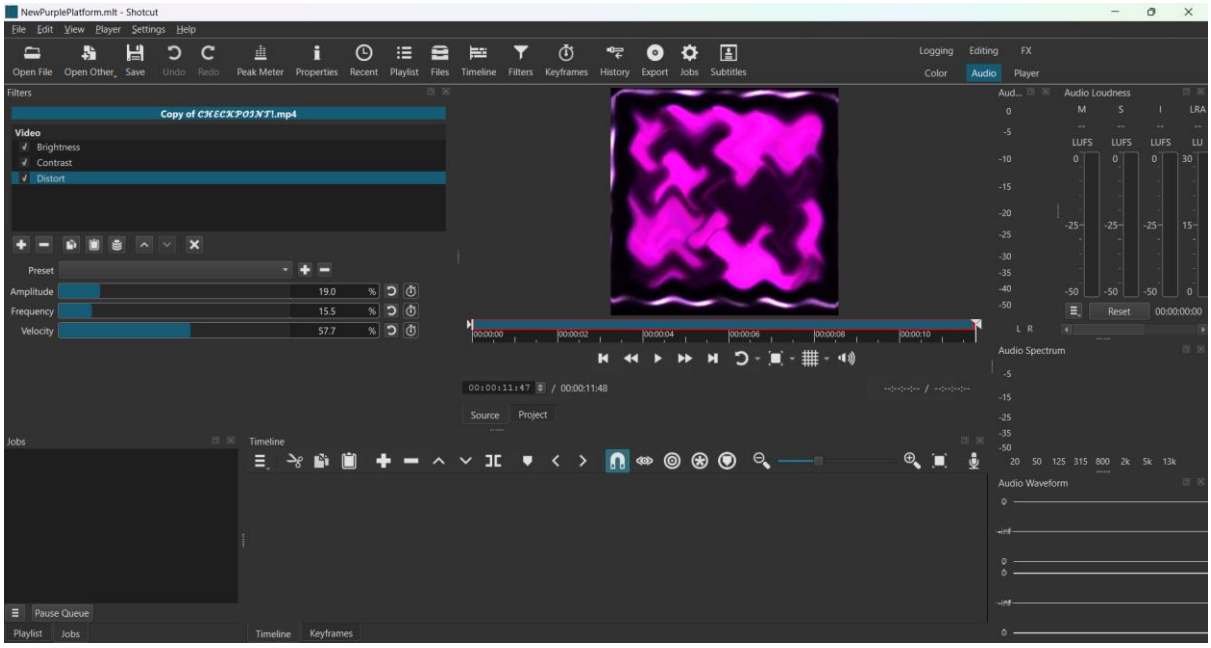


Figure 24: Shotcut Environment and Demonstration.

Ezgif is a collection of tools for creating and editing GIFs. This tool provides Video or GIFs conversion to a PNG image sequence. For example, after Shotcut the color edited captures of the time-based platforms are converted from Video to JPG. A different utilization of the tool is, downloading GIFs, e.g. Uka Uka UI flames and converting them into PNG or JPG format. It is worth noting that, prior to the conversion there are options like the selection of FPS, video duration and size.

Figure 25: Ezgif Environment and Range of Tools.

Remove.bg is an AI-powered tool which automatically removes backgrounds from images. It is a very useful tool for removing backgrounds of textures or 2D sprites. It was leveraged on so many different occasions due to its simplicity and effectiveness. In *Figure 26* on the left there are pictures including the Mojo counter and Mojo icon used in crates. All the left ones are examples of ChatGPT creations. Giving correct and purposeful prompts with other related image attachments, ChatGPT created the first version of the images. The current images are edited with Remove.bg and Img.Upscaler.

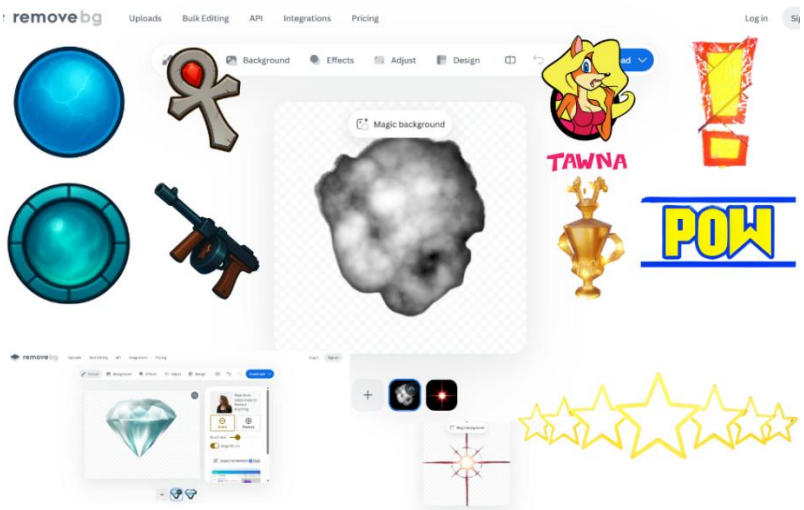


Figure 26: Some examples of Remove.bg Applications.

Img.Upscaler is an online, AI-powered image upscaling tool. It aided in fixing low-resolution or blurry captures, increasing the game's textures clarity. It was used as much as remove.bg and in most cases right after. Img.Upscaler improved the quality while keeping the background removed.

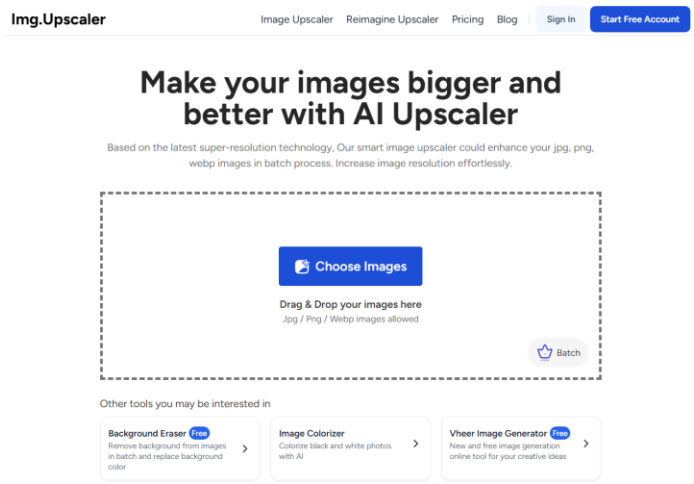


Figure 27: Img.Upscaler interface.

Leshy Spritesheet Tool is a web based, free, HTML5 application for creating, modifying and packaging sprite sheets. Before discovering it, sprite sheets were created by arranging in Paint all the images needed, in a horizontal line. It was a very tedious and limited procedure. After some research, this tool was found and from that moment forward all the sprite sheets were created there. It is user-friendly, since the only thing you have to do is import the pictures. For instance, after the conversion from video to JPG of the color time-based platform with ezgif, all the new pictures were dragged inside the page containing Leshy Spritesheet Tool. It creates a ready-to-use Sprite Sheet waiting to be downloaded and imported to Unity.

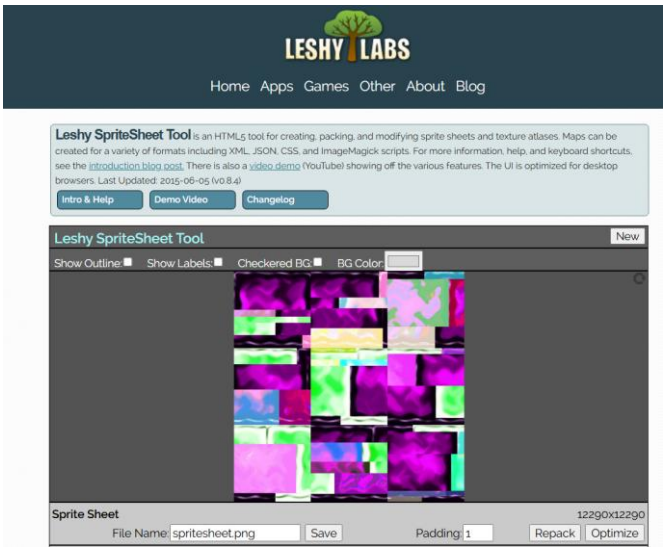


Figure 28: Leshy Spritesheet Tool Environment and Demonstration.

3.2.2 Audio Editing and Sound Design

Websites such as The Sounds Resource and YouTube as well as tools namely Audacity, Rave.DJ, SlowedAndReverb and Vocal Remover AI were utilized for downloading, editing, mixing and optimizing music and sound effects.

The Sounds Resource is a website that rips sounds from video games. It has remained active since 2003, and the audio quality is of the highest quality. Sound effects were downloaded from several games which were inspirations for the game. If sound effects were not found from that website an alternative was recording or downloading videos from YouTube which contained a number of sound effects.

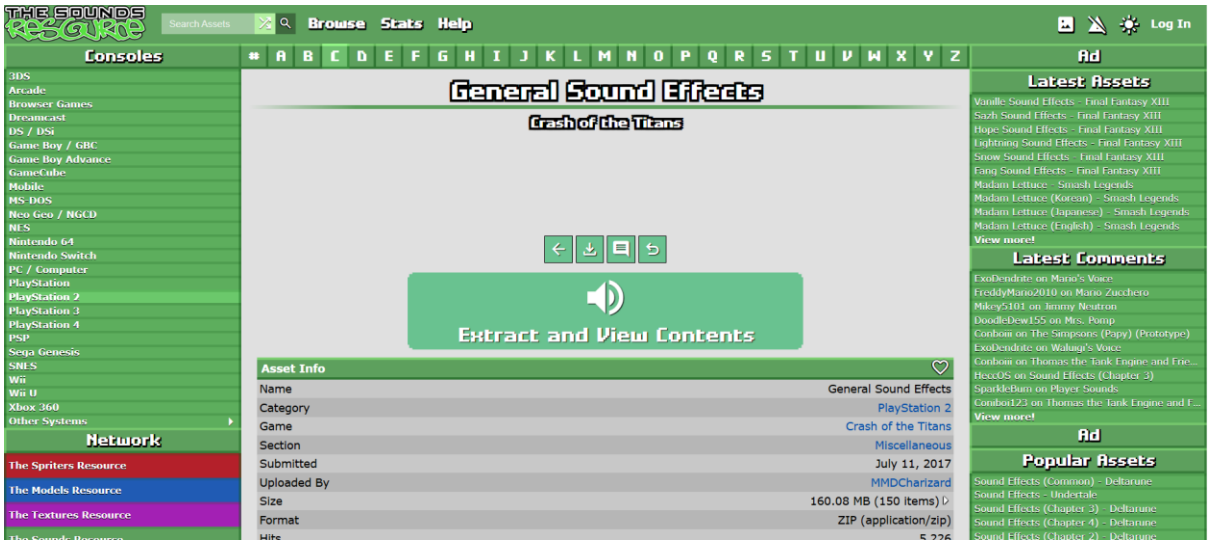


Figure 29: The Sound Resource Environment and as an Example Download of Crash of the Titans Sound Effects.

Audacity is one of the most popular free software for recording and editing audio. It was used to play around with sounds and create new sound effects with a combination of sounds downloaded from The Sounds Resource and recorded from YouTube. For example, a unique sound effect was created for the POW crate count down, which used the TNT countdown sound effect downloaded from a collection of Crash Bandicoot N Sane Trilogy sound effects found at The Sounds Resource. Then a heavily edited version of them was combined with the firework sound effect from the same pack of sound effects. Another example is the utilization of the software to create new versions of the Mojo collection sound effect. *Figure 30* shows the variety of sound effects that were edited and combined to create the Pink Mojo sound effect. The quality of the audio output relies heavily on the individual's skill level and artistic vision, whereas with Rave.DJ having a general concept of how two tracks might blend based solely on an initial idea of how they could sound together is the only input.

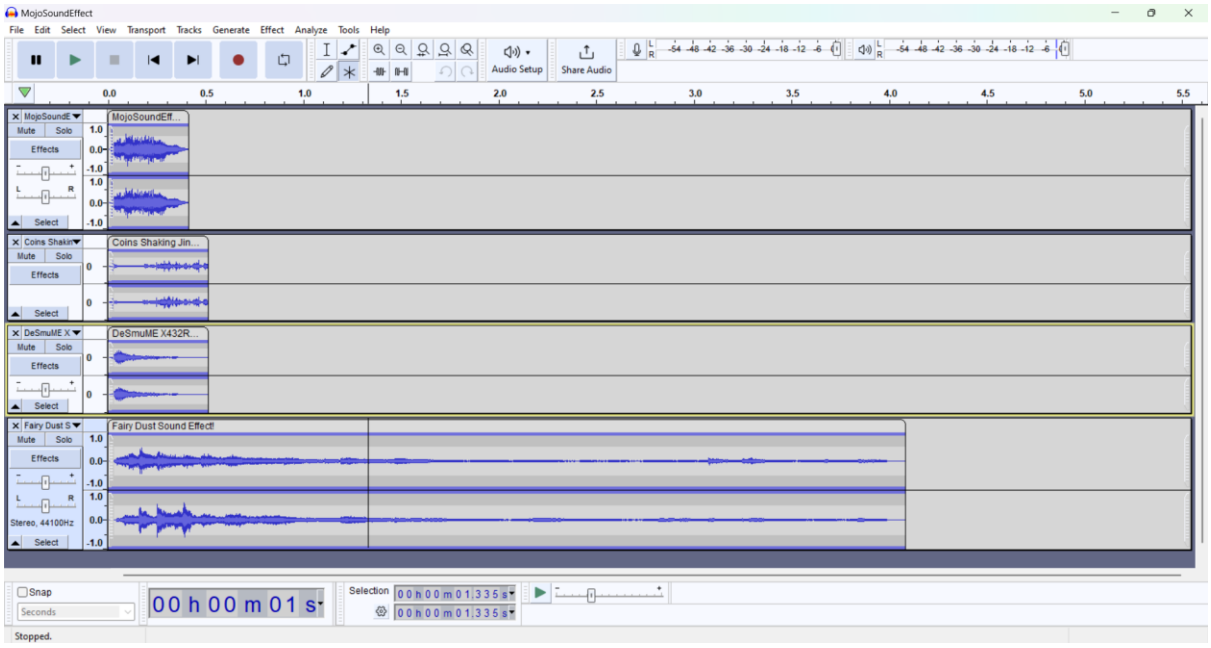


Figure 30: Audacity Environment and Example of Mojo Sound Effect Creation.

Rave.DJ is a powerful AI-powered music mixer and mashup creator that allows users to blend and merge their favorite songs from YouTube and Spotify, regardless of key or tempo. It uses artificial intelligence to thoroughly analyze these songs and automatically generate a seamless and professional-sounding mix. It was deemed useful for the creation of the game's soundtrack. To form the background music for the levels Bonsai Ruins and Colour Agony, mashups of the original soundtrack of their respective inspired levels and other songs that would be fitting were generated. For Colour Agony specifically, a subgenre of Phonk music that combines the lo-fi, distorted sound of American Memphis rap with elements of Brazilian funk called Brazilian Phonk was the genre of songs deemed fitting with Crash Twinsanity's acapella. Acapella is music performed using only human voice. In *Figure 31* the two songs that created the background soundtrack can be seen.

Figure 31: Rave.DJ Environment and Example of Colour Agony's Main Area Soundtrack.

Slowed and reverb is a technique of remixing and a subgenre, derived from chopped and screwed hip-hop and vaporwave, which involves slowing down and adding reverb to a previously existing song. Editors such as Audacity do have such features but a specialized studio browser extension was utilized to create slowed and reverb versions of some songs made using Rave.DJ. In *Figure 32* an example of its utilization is showcased.

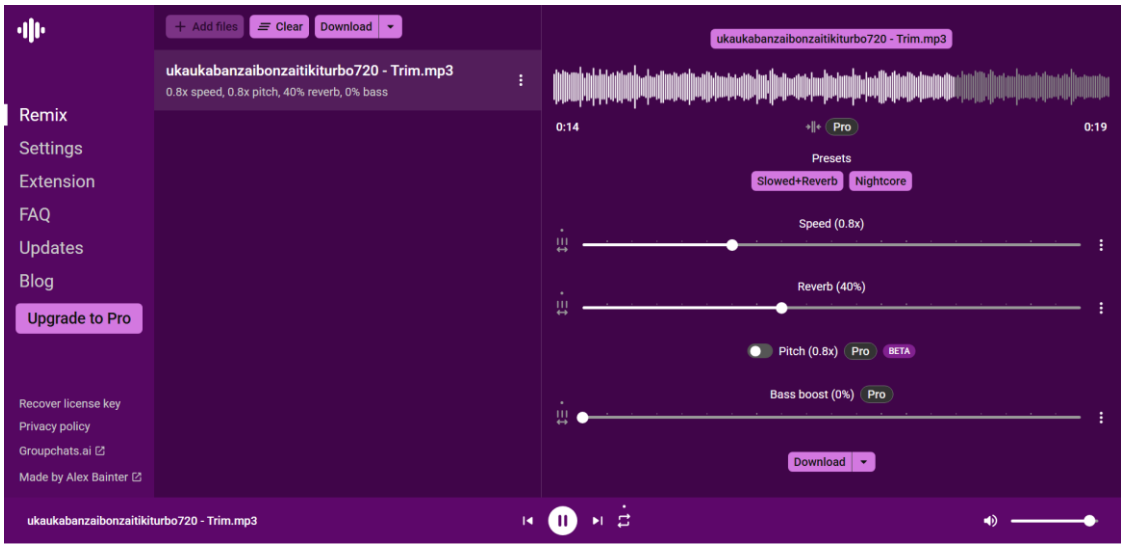


Figure 32: SlowedAndReverb Environment with an Example of the Edited Uka Uka soundtrack from Bonsai Ruins.

Vocal Remover AI is a free online application that utilizes advanced AI algorithms to isolate and remove vocals from any track, leaving the user with a clean instrumental version. It includes more features like a pitch changer and audio trimmer. At the outset of this study, it was used it to remove the vocals of the Bonsai Ruins main soundtrack and other future level music that were Rave.DJ mashup creations.

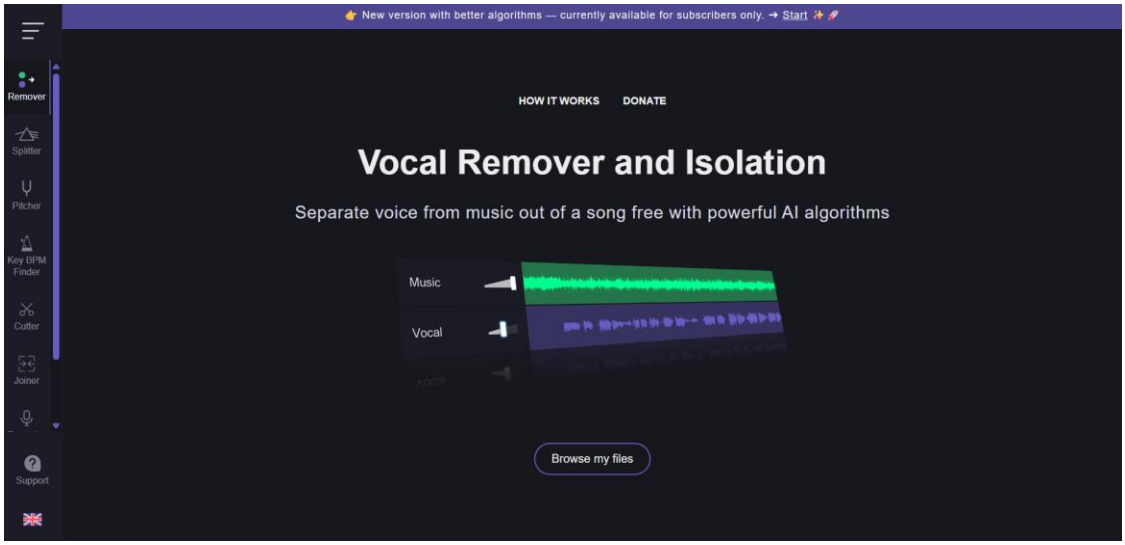


Figure 33: Vocal Remover AI Interface and Features.

3.2.3 Animation and 3D Modeling

Most of the three-dimensional assets were produced for the game using external tools. The primary tool for modeling, rigging and animating was Blender. MeshyAI was used selectively as an AI-assisted modeling tool to generate base meshes. The meshes were then refined in Blender. While MeshyAI was not used for developing core characters like Pinstripe and Tawna, using it as a base for other background enemies, objects or environmental elements was pivotal in enhancing the visual quality of the game. There are multiple instances that characterize the utilization of Blender and MeshyAI but only four representative examples will be mentioned that define the remaining cases.

The first example focuses on the utilization of MeshyAI for asset creation without the need for further editing. In *Figure 34* the creation of a platform used in the main area of the hub world of the game “The MotorWorld” is showcased. To create it a screenshot of a YouTube video which showcased Crash Tag Team Racing’s CGI cutscenes in superior video quality was used. The next step is, dragging the screenshot into Meshy’s Workspace and generating a mesh using the Meshy 5 AI model. It was important to re-mesh the object specifying a 10.000 Face limit because it is a model that will be used more than once and in close proximity. The smaller the limit is, the more gameplay performance improves. The textures for the re-mesh were also generated.

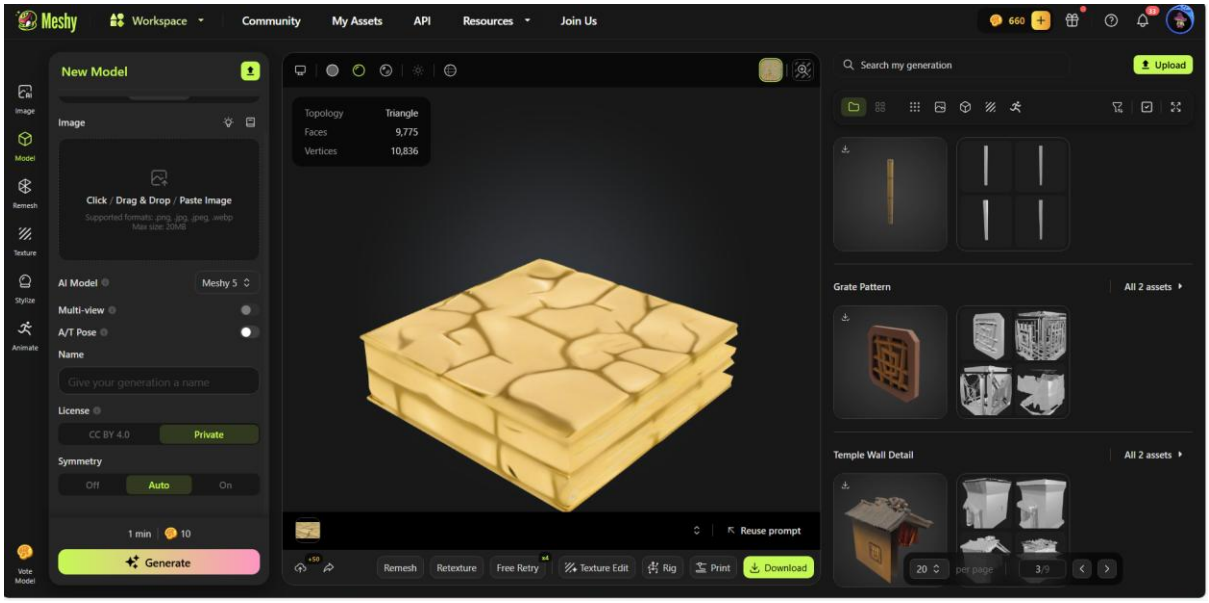


Figure 34: The MotorWorld’s Ground Platform Created in MeshyAI’s Workshop.

The second example focuses on the utilization of MeshyAI for asset and animation creation. Multiple enemies of Colour Agony have been created in MeshyAI. There is an option to add a skeleton to the model created identifying and assigning each body part. *Figure 35* depicts this procedure and shows multiple examples of models and their animations created inside the Workspace.

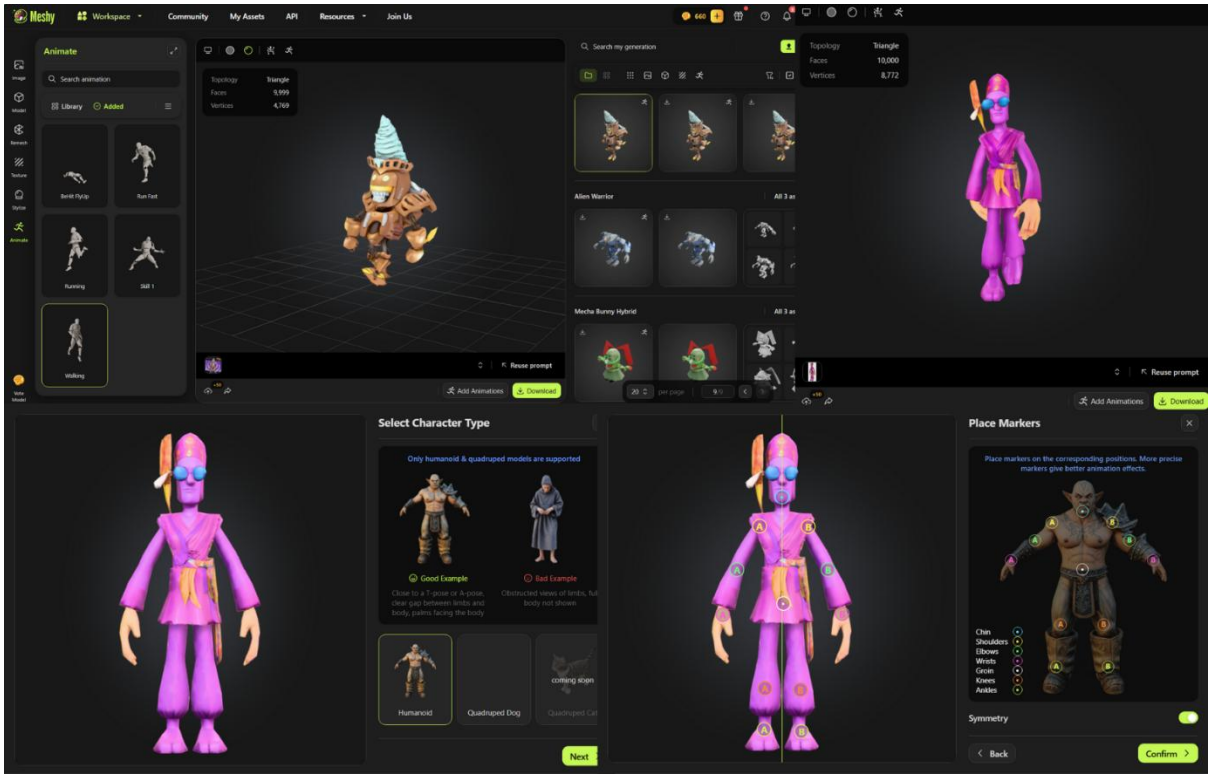


Figure 35: Models, Animations and the Rigging of the Model for Animation Assignment.

The third example focuses on the utilization of MeshyAI for mesh creation and Blender for detail refinement. In most cases, MeshyAI will generate a model of impressive quality. However, those models might be missing details or have misplaced meshes owing to low-quality reference images or complex prompts. As a result, importing those FBX models first into Blender before moving to Unity was the optimal solution. In *Figure 36* a curtain was created for Bonsai Ruins with MeshyAI. Immediately after, it was imported to Blender to remove the misplaced mesh in Edit Mode and was missing some important details, so they were added in Blender by using a morphing tool in Sculpt Mode.

The fourth example focuses on the utilization of the MikuMikuDance toolset. MMD was specifically used to import pre-rigged models with existing animations. This approach was beneficial for saving time on certain characters or objects where manual rigging was not necessary. For instance, the final Pinstripe model was initially imported through MMD tools with a complete skeleton and a set of base animations. Unfortunately, most of those animations were Pinstripe sitting on a kart and were relevant to driving not walking or performing actions on foot. New animations with the existing skeleton needed to be created. The facial bone structure animation keys of some driving animations were transferred to the custom-built animations in order to enhance the model's liveliness. *Figure 37* displays Blender's Pose Mode in which the model's bones were adjusted and rotated and within the Dope Sheet the Action Editor which is used to create, manage and organize animation actions.

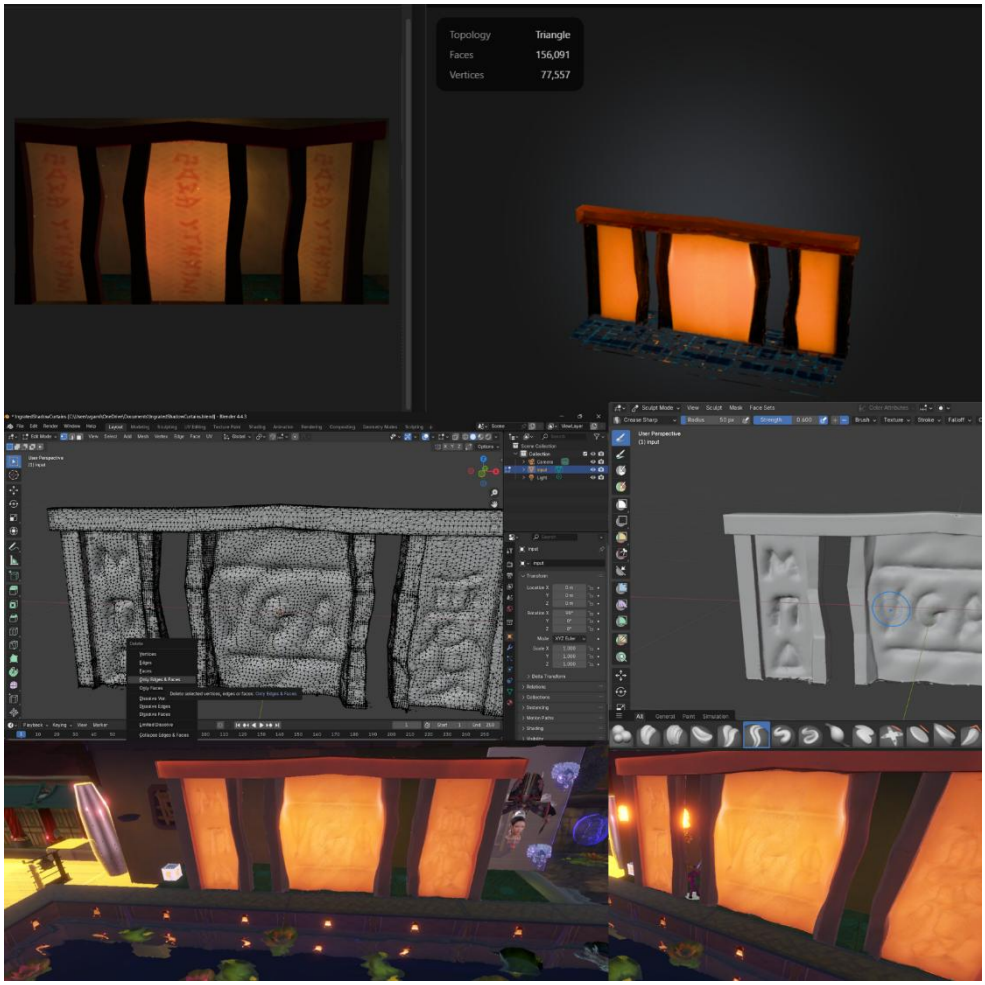


Figure 36: Curtain from Screenshot to In-Game Implementation.

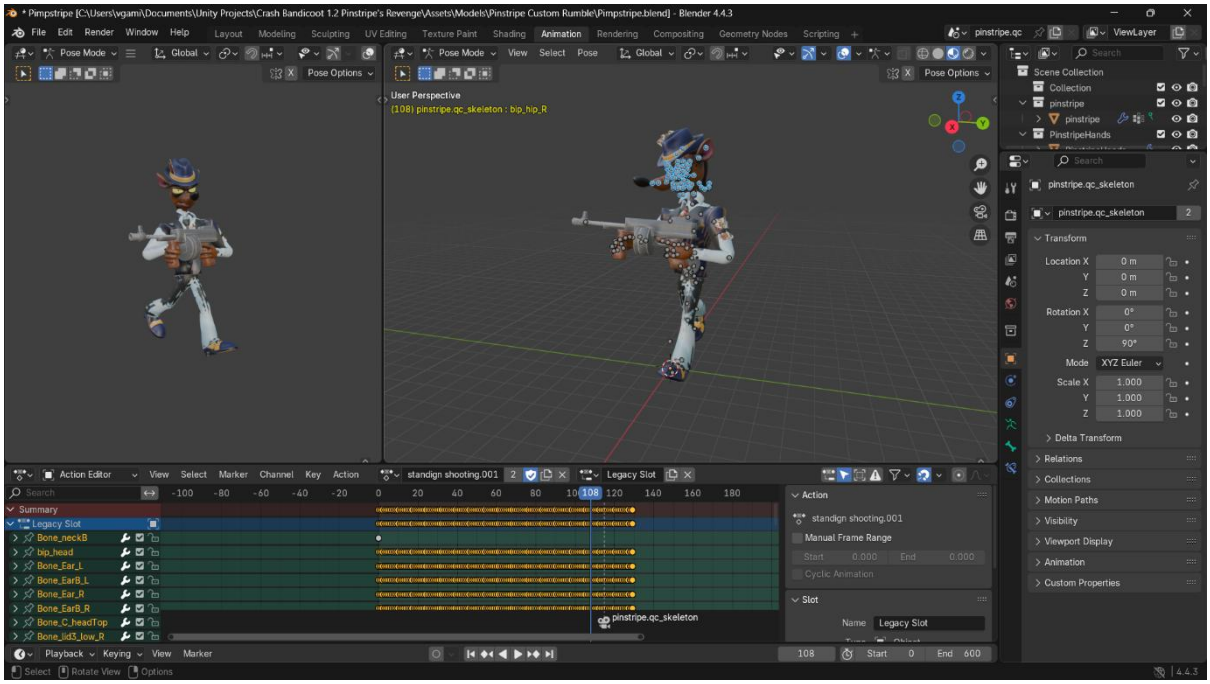


Figure 37 Blender's Animation Creation and Editing with Pinstripe as an Example

3.2.4 Programming Environment

The creation of scripts defines the game's logic, behaviors, and interactions. While Unity provides a built-in compiler for C# scripts, there is no component for writing code. For this purpose, Microsoft Visual Studio was selected to write, edit, and debug code. It's compatible with Unity and it offers an extensive feature set that is designed for professional software development.

There are so many practical examples of the application of Visual Studio. This is because the entire game relies on a complex network of scripts that handle everything. The camera system that follows the player, the player's actions and the environment's corresponding reactions are some applications based on recollection. In *Figure 38* the Visual Studio environment and a code snippet that is part of the "TargetingSystem" script are shown.

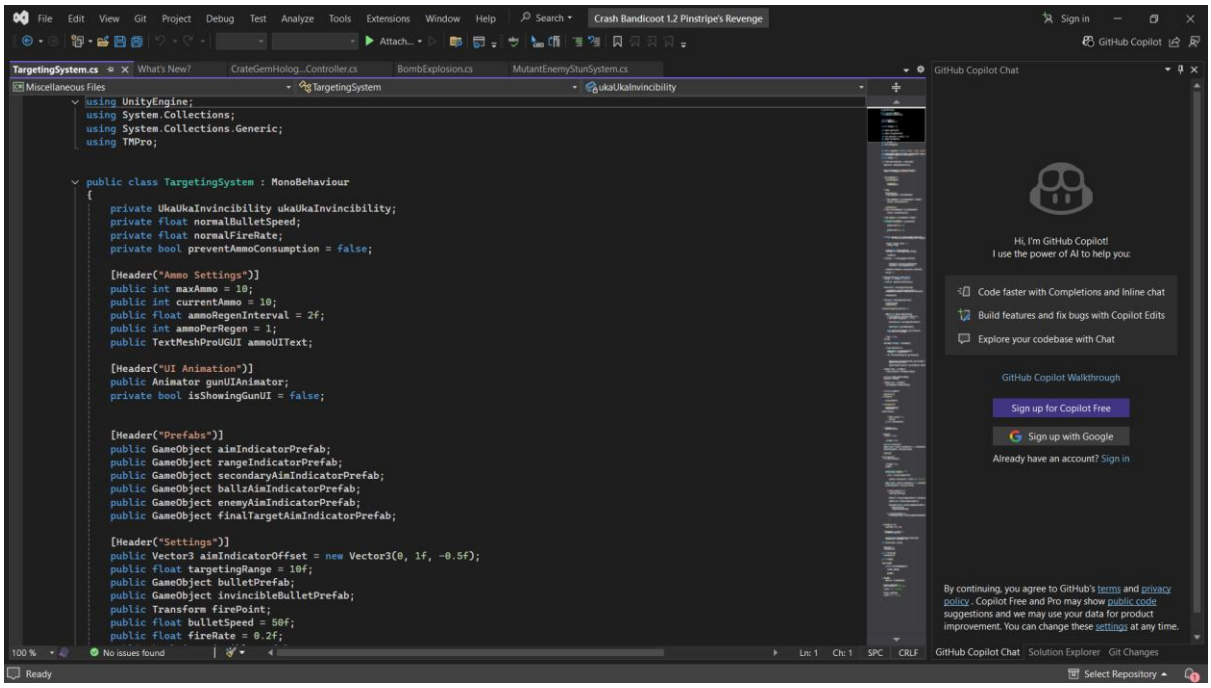


Figure 38 Microsoft Visual Studio Environment and Code Snippet of the script TargetingSystem.cs

One of Microsoft Visual Studio's useful features that were utilized is error detection and real-time feedback during coding. If the code was incorrect the game would automatically prevent the developer from testing, and an error message would appear specifying which line of code caused the error. To speed up the process, ChatGPT was used multiple times to troubleshoot those lines of code. ChatGPT has been trained on a large dataset containing programming languages including C# allowing AI to recognize patterns in C# syntax and structure while generating clean and readable code. So, by explaining the problem in plain English or Greek it was able to interpret what was wanted from the script while I was unconsciously improving my programming skills.

To illustrate the scope of the development process, approximately two hundred scripts have been created to manage various gameplay elements and systems within the game. *Figure 39* shows some of them.

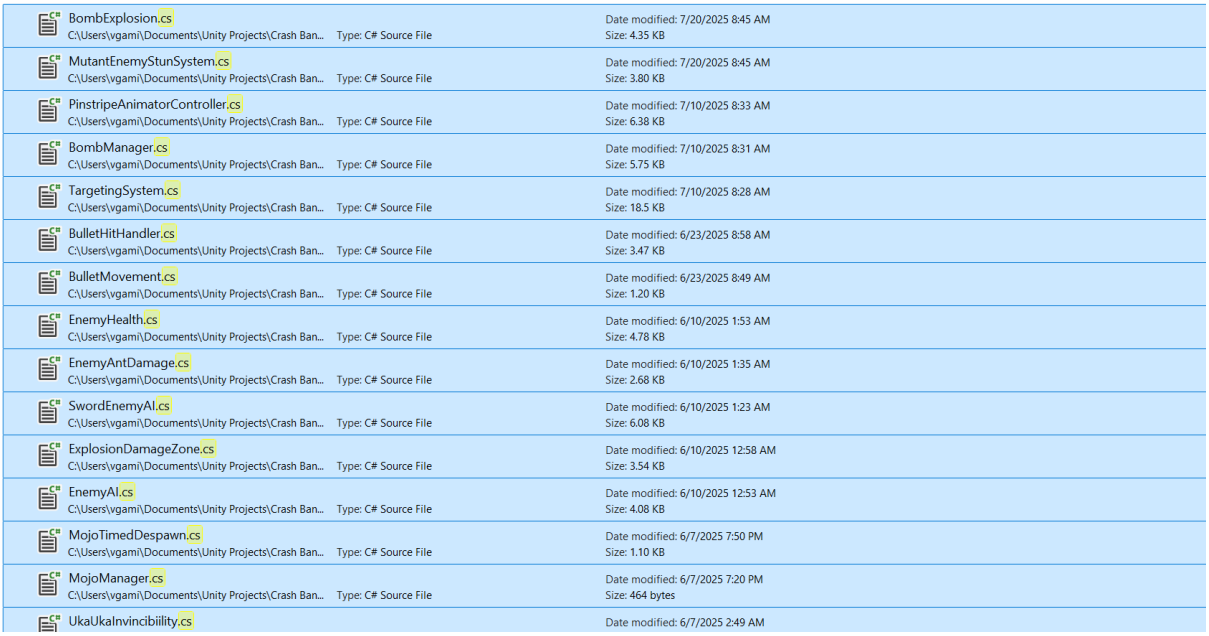


Figure 39: Some of the Scripts made with Microsoft Visual Studio for the Project.

A snippet of code for the bomb explosion and its function within the game will be presented. The script is named “BombExplosion” and contains currently 138 lines of code. From line 75 to line 94 the player’s interaction with the bomb has been coded. Specifically, if the player is near the bomb radius, three bars off his health will be removed. *Figure 40* shows the in-game effect of the code.

```
if (col.CompareTag("Player"))
{
    HealthSystem player = col.GetComponent<HealthSystem>();
    if (player != null)
    {
        if (!player.isInvulnerable)
        {
            Debug.Log("Bomb hit player dealing 3 damage.");
            EnemyDamage enemyDamage = player.GetComponent<EnemyDamage>();
            if (enemyDamage != null)
            {
                enemyDamage.TakeTrueDamageWithUI(3);
            }

        }
        else
        {
            Debug.Log("Bomb hit player but they are invulnerable.");
        }
    }
}
```

An analysis of this code will now be showcased. In line 75 this conditional statement checks if the object involved in the collision has been tagged as “Player”. It’s important in order to secure that whatever code follows will only affect the Player. “HealthSystem” is a different script attached to the player which with line 77 whatever code is inside that script will be retrieved for the “BombExplosion” script and the script can be called by typing “player”. Line 78 is important to prevent runtime errors if the “HealthSystem” does not exist. Another important conditional statement is one considering the “player” invulnerability in line 80. From other parts of code from the script “HealthSystem” there is a state programmed for the Player to be invulnerable after an attack for some seconds. In line 82 is a good example of debug code. If the code doesn’t work properly while the game is running the Debug message “Bomb hit player dealing 3 damage” will not appear in Unity’s Console window. Line 83 gets component meaning in this case a different script called “EnemyDamage” to be called by typing “enemydamage”. Line 84 calls it and checks with this conditional statement the script’s existence. Line 86 calls a method that is inside the script “EnemyDamage” which reduces the health of the player by a certain amount. A “3” was put in a parentheses meaning it will reduce this time 3 health bars. Line 89 is part of the conditional statement of line 80. If the player is invulnerable damage is not applied. Closing the analysis of the code with line 91, a different log statement that helps the developer during testing is presented, this time “Bomb hit player but they are invulnerable” meaning the bomb’s explosion affected the player but the player was hit before and thanks to the structure of the code the health bar reduction part in this case is skipped.

Understanding the code is vital for long-term development because the procedure of resolving small errors becomes faster. For instance, in line 86 knowing where the amount of health reduction is indicated inside the script helps you tweak it right away.

Figure 40: In-Game Representation of Health Bar Reduction from Explosion Damage.

3.3 Unity Systems and In-Engine Tools

Unity served as the central integration hub for the development of the game. This section focuses on Unity's built-in systems and components that were critical to implementing gameplay mechanics, managing assets, and achieving the desired visual and performance goals. Each subsection discusses a key Unity feature, its purpose, and how it was applied to this project.

3.3.1 Animation and Motion Systems

The Animator System in Unity is responsible for controlling animations of characters, enemies, platforms, background objects and UI elements. It operates as a state machine, defining how animations transition between different gameplay states. Selected examples are explained in this subsection.

A representative case is the implementation of the Animator System to a falling platform. In *Figure 42* the Animator's view is visible where all the animations can be controlled, and transitions are added. In most cases, it resembles a tree structure. For example, in the falling red platform the nodes represent the states or decision points "RedFallingIdle", "RedFallingFalling", "RedFallingStayFallen, "RedFallingRising". These are all animation states that exhibit a different motion of the platform. Those animations were created from the ground up in Unity's Animation Window. The branches represent transitions or conditions that connect the nodes. For example, in *Figure 41* the transition from "RedFallingIdle" to "RedFallingFalling" happens only if the condition "OhNo" is met. That condition is programmed in a script that controls it. Specifically, it is activated when the player collides with the platform's trigger collision. The rest of the transitions don't require any conditions, instead they have an exit time, which means that the animator follows the transition paths to determine the next state.

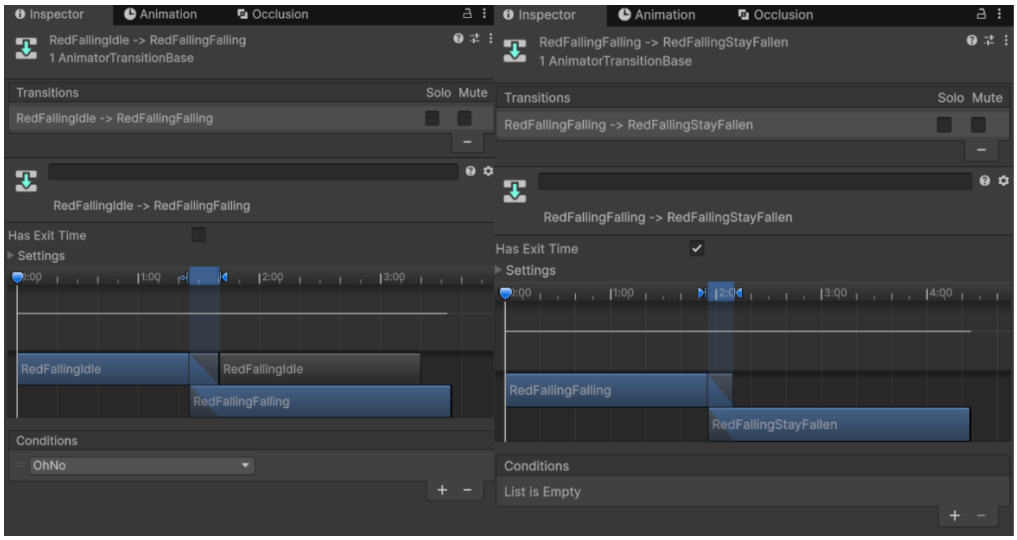


Figure 41: Basic Example of State Transitions.

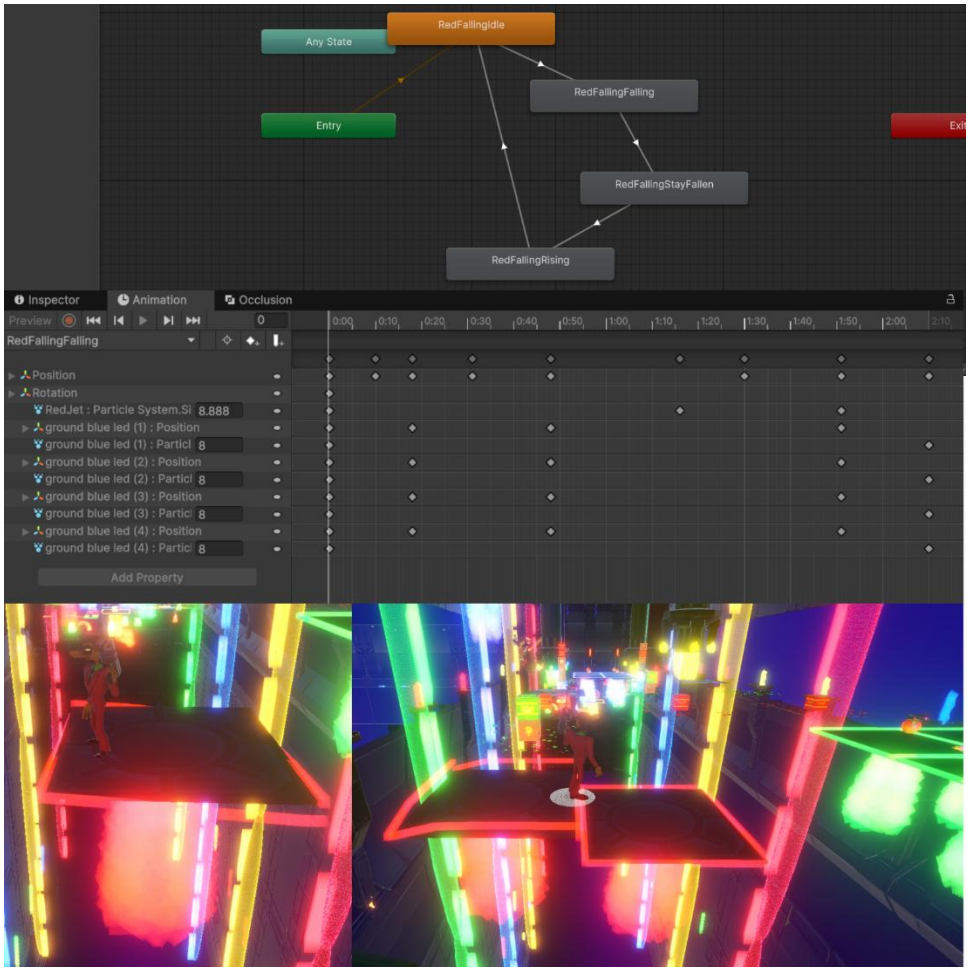


Figure 42: Animator System Environment and the Animation Window including the Falling State of the Red Platform.

An example of a multi-layered animator structure is the one created for the Sword Ant enemies in *Figure 43*. There are more transitions and conditions than the Animator for the falling platforms. There are multiple scripts that interact with the Animator component. Based on interactions between the enemy and the player or the enemy and the environment, these scripts select and play the appropriate animation to reflect the current action. A noticeable difference between the previous and this Animator setup is the use of Any State. It is a special state that serves as a global entry point, allowing transitions to occur from any other state. It is applicable in this context since the script coordinates the Animator component with the NavMeshAgent component. The NavMeshAgent component is responsible for controlling the enemy's navigation and pathfinding within a pre-baked NavMesh area. In *Figure 44* the turquoise platforms are pre-baked NavMesh areas. The script sends parameters and triggers to the Animator indicating which animation state should be played. However, some animations need to interrupt any current state immediately such as getting stunned. Once the enemy is stunned the NavMeshAgent is disabled making the enemy unable to move. After it ends the script resumes control by re-enabling the NavMeshAgent and continuing with the state transition after "Stun". The Ant's animations were imported from MeshyAI and edited in Unity's Animation Window.

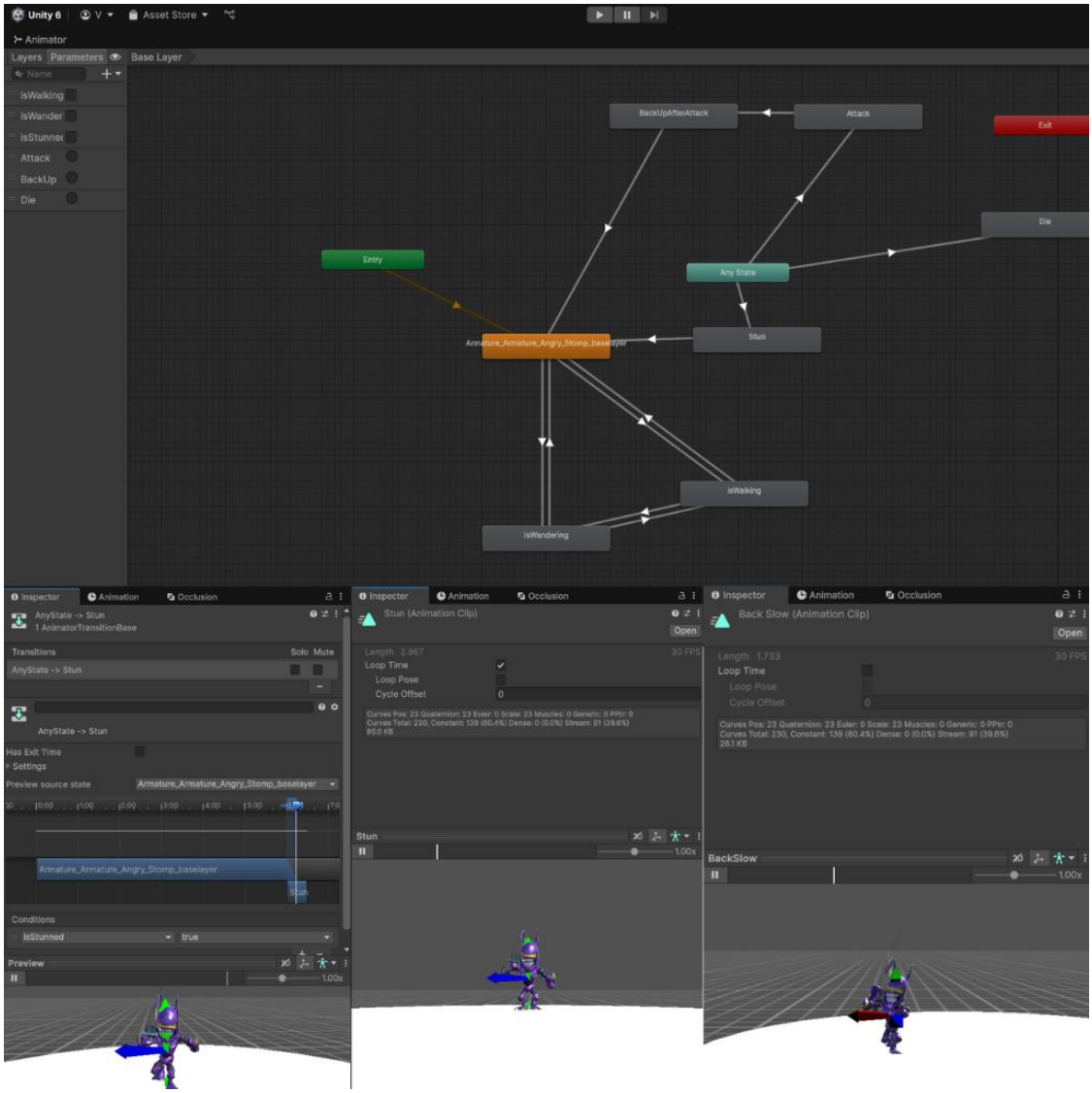


Figure 43: The Yellow Sword Enemy's Animator and Animations.

Figure 44: NavMeshAreas in Colour Agony's Scene View.

The final animator setup presented as an example is the current state of Pinstripe's Animator component. The Animator component in this case is not attached to the Parent gameobject. There is a child gameobject namely "PinstripeVisuals" which handles all of Pinstripe's animations. Initially, the Animator setup was like the rest, structured like a tree, with states connected through numerous transitions and controlled by many parameters. However, as the project progressed, the number of branches multiplied to a point where it became an unmanageable system. The main character reasonably has the most animations out of any character. So instead of removing some animations, the decision to remove all transitions and parameters was made. A custom script replaced them, handling all state switching. The existing script that handled the physics of the player's actions was updated to incorporate the new script. While the transitions are currently not as fluid, it ensured perfect synchronization between animations and gameplay actions. Furthermore, it simplifies significantly the addition of new animations, as only a few lines of code are required to integrate them. In *Figure 45* both the current Animator setup and the old project's Animator setup are presented to demonstrate the importance of optimization. The current version of the Animator setup is superior because it is simpler and incorporates more animations.

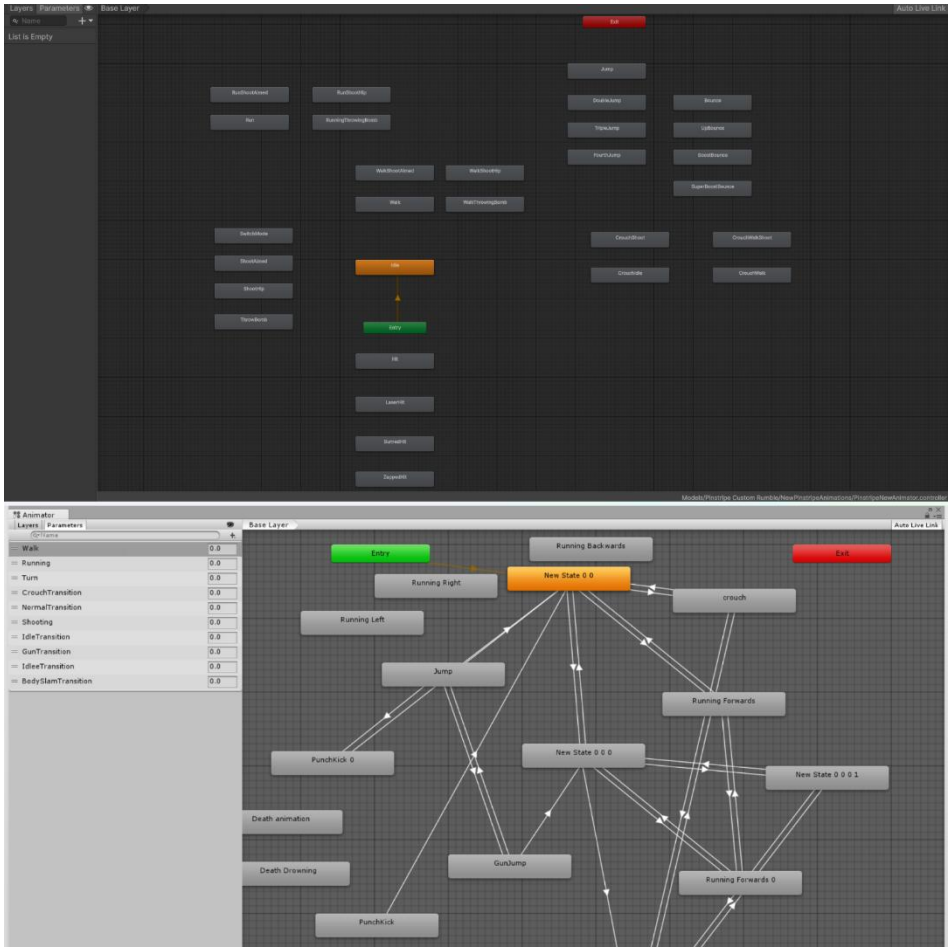


Figure 45: Comparison between Current and Old Animator Setup.

While animations present the visualization of actions, physics components such as Rigidbodies and Colliders establish the rules governing motion and interaction. Instead of the Transform properties, developers are able to simulate physics forces and torque to move the GameObject with the utilization of a Rigidbody.

A Rigidbody is required for dynamic and physically driven movement as it includes settings such as Mass affecting the weight of the character when forces are applied, Drag and Angular Drag controlling the velocity over time of the movement, Interpolation which improves visual smoothness when rendering physics and Collision Detection which is important to prevent tunneling when fast movement is applied to game objects. All these parameters were carefully adjusted to balance realistic behavior with responsive platforming controls.

In Unity, there are several different types of colliders. The three I used the most throughout this game's development are Box Colliders, Sphere Colliders and Capsule Colliders. Colliders are solid invisible physical areas for a game object, preventing other objects from passing through it. Box Colliders are rectangular, cube-shaped while Sphere Colliders are spherical, and Capsule Colliders are made of a cylinder with two half-spheres at its ends. Representative examples of both Collider and Rigidbody usage will be examined.

Pinstripe's parent object utilizes a rigidbody, a capsule collider and a box collider as illustrated in *Figure 46*. Whereas most 3D platformers made with Unity rely on a single collider type, following extensive testing a different approach was necessary to enhance collision precision and responsiveness. In the setup of dual-collider configuration, the system detects depending on the surface type which collider is to be utilized with the layer override setting. That is the main concept of multiple collider utilization for one character. All physics for the model are still not finalized but since adapting this concept to the game the precision of collide detection has improved significantly.

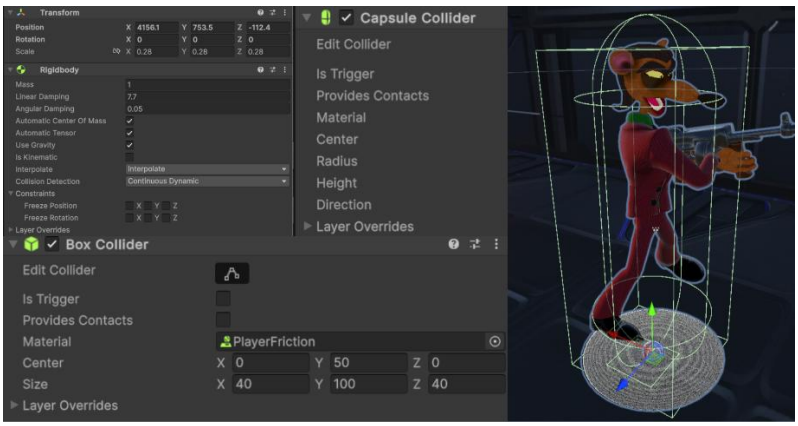


Figure 46: Pinstripe's Transform, Rigidbody and Collider Components.

In *Figure 47* at the top left corner the structure of the Pinstripe prefab is depicted. “Pimpstripe” is the parent game object, and the rest are its child game objects. “GroundDetector” is one of the child gameobjects and is responsible for the ground detection while performing actions such as jumps and movement. It is tagged “GroundDetector” and “Pimpstripe” is tagged “Player” which are useful to recognize in scripts while programming. Moreover, in the top right corner of *Figure 47* the box collider of a platform” Shadow_Panels_Floor (8)” is also visible, below its layer name is “Ground” and to the right an invisible Ground Detector game object. The game objects that have a collider and their layer name is “Ground” are programmed to be platforms where the player can perform jumps. The Ground Detector’s components are in the bottom left corner. What is different with this Box Collider component is the “Is Trigger” box is ticked. This means that the collider is no longer a solid physical boundary. Instead, it monitors when another Collider enters, remains within or exits its volume. At the bottom right corner, the main script that combines all physics and visuals of Pinstripe is presented.

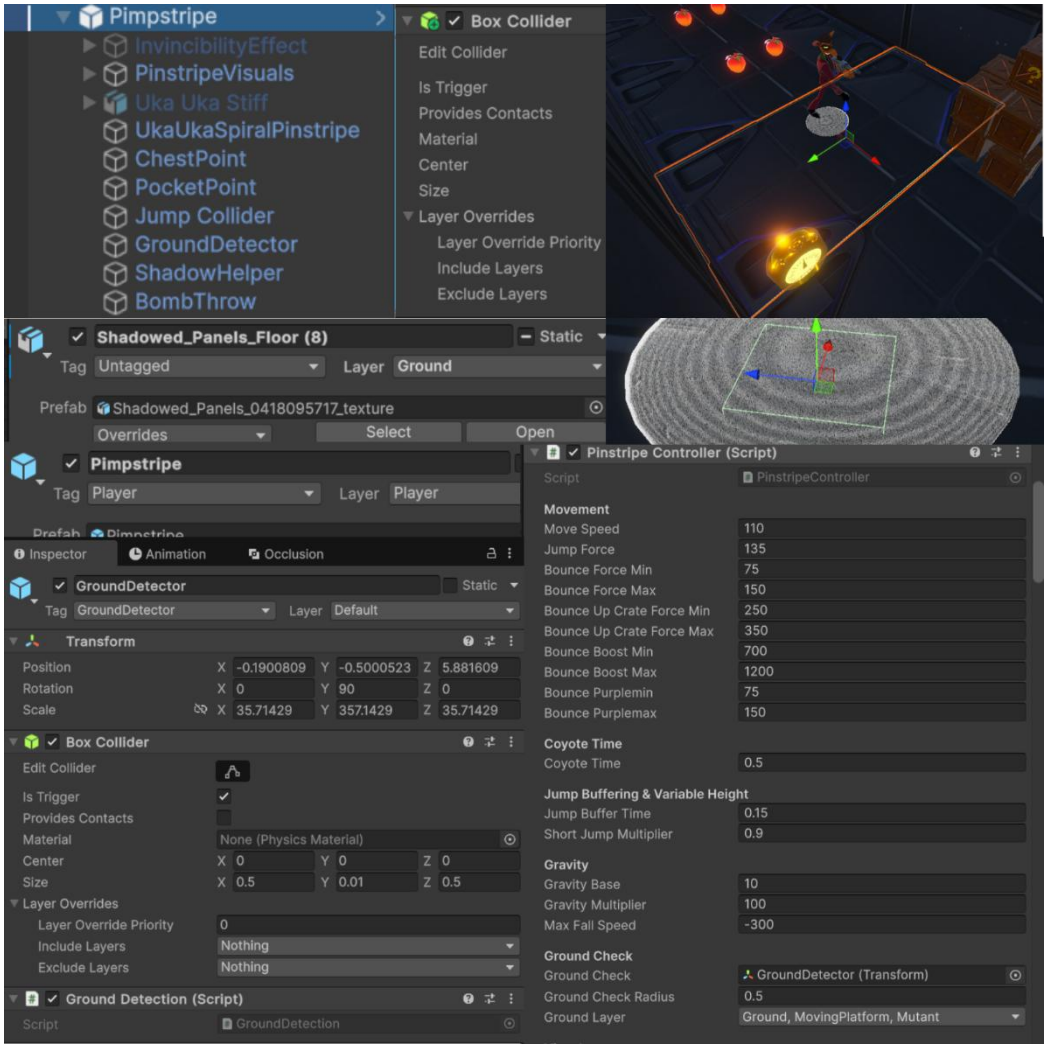


Figure 47: Pinstripe's Prefab Gameobject Structure, The GroundDetector Child Gameobject, Pinstripe Controller Script and a Platform’s Box Collider In-Game Illustration.

An example of Sphere Collider utilization is for the different types of Mojo. Specifically, I have made two versions of the same Mojo type. One version doesn't have a Rigidbody component, instead has animations to replicate a floating motion and a sphere collider which "Is Trigger" is ticked. This version is simpler and is used for all Mojo scattered around the level. The other version has a Rigidbody component as well as two different Sphere Collider components attached to it. These Mojo are stored inside crates or enemies. There are more differences between the two versions, but they are unrelated to colliders.

The second version of the Mojo is displayed in *Figure 48*. In this instance, Layer Override is operationalized. One collider is used for the interaction of Mojo with the environment. Particularly, this collision includes most layers and excludes the layer "Player". In addition, the "Is Trigger" box is unticked providing a solid area between the Mojo and, excluding the Player's colliders, most game objects that have collisions. The other collider is used only for the interaction between Player and Mojo. It is the same as the one used for the simpler version of the Mojo. It includes only the "Player" layer and the "Is Trigger" box is always ticked. This collider, with the assistance of a script, triggers the Mojo' ability to be collected by the Player.

This Mojo's Rigidbody component uses gravity and a discrete collision detection because it is not a fast-moving object, and this is the default and most performance friendly collision detection option. The rest of the Rigidbody's settings were not altered.

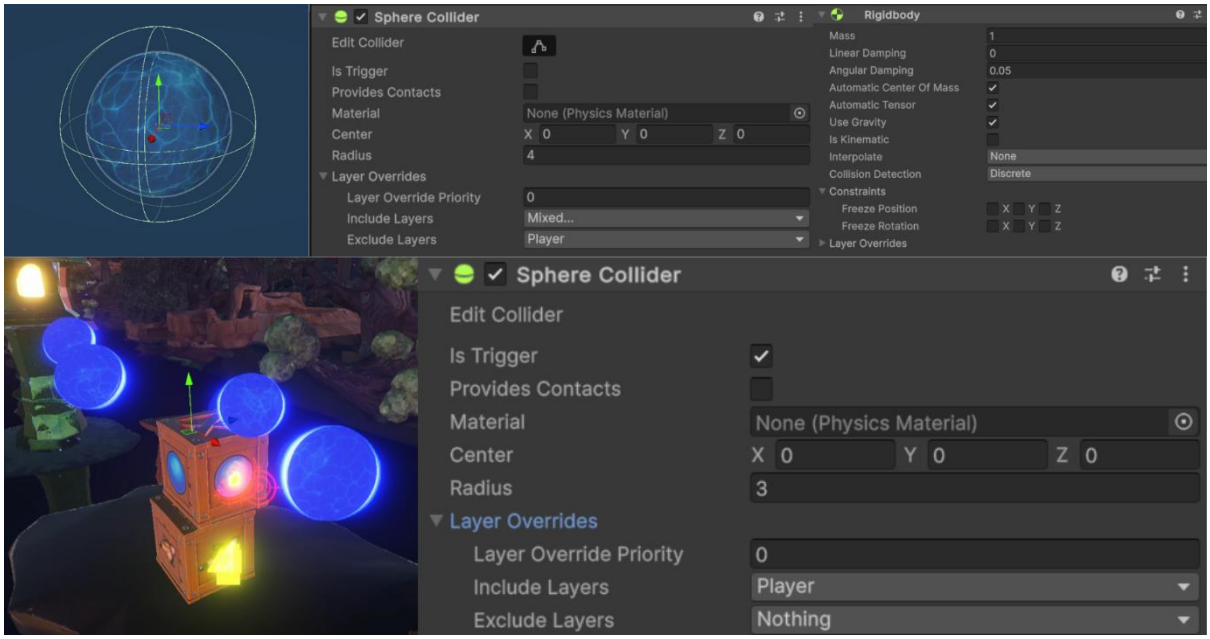


Figure 48: Mojo's Rigidbody, Sphere Colliders and In-Game Portrayal.

Another demonstration of physics behavior in the game is the balls from ball-puzzle door. Each ball utilizes two Sphere Colliders. When the bullet's collision is co-located with the ball's collision through a script "BulletHitHandler" due to the line of code "OnTriggerEnter(Collider other)" the ball's collision is necessary for the course of actions that follow to be executed. The "Is Trigger" box is of necessity ticked. It is noteworthy because if it was unticked, the part "OnTriggerEnter" of the code would have been changed to "OnColliderEnter". Additionally, in order to not go through the ball as the Player, a different Sphere Collider was added with "Is Trigger" unticked as seen in *Figure 49*. The main collectibles feature a simpler implementation of the collider. In *Figure 49* a simple box collider with the "Is Trigger" box ticked is utilized. By virtue of the script "OrangeGemPickUp", when the Player enters the Sphere Collider, the item is collected.

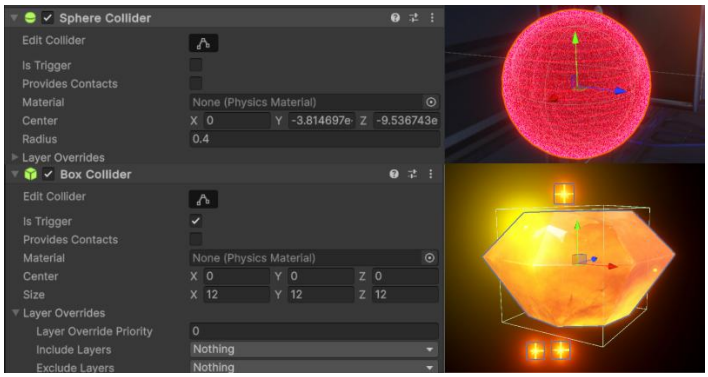


Figure 49: Game Objects and their Colliders.

Before proceeding to the next subsection, it is important to mention the Physic Material. It is a separate asset that is applied to a collider component to control how objects interact with each other. Friction is the amount of resistance there is to motion, bounciness is the amount of rebound upon impact. In the game Pinstripe's Capsule Collider utilizes a Physic Material, namely "PlayerFriction1" as shown in *Figure 50*.

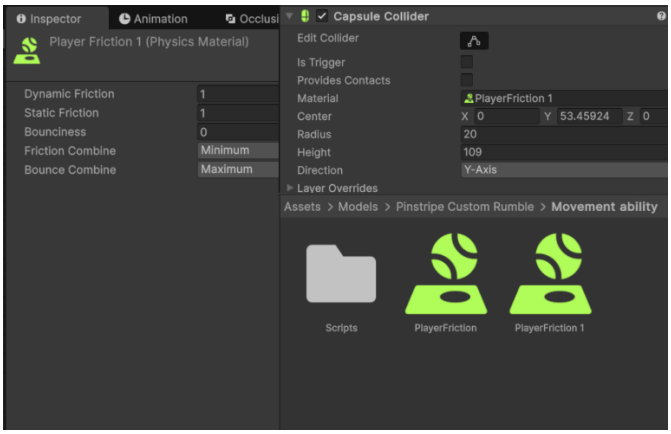


Figure 50: Physic Material Settings.

3.3.2 Rendering and Visual Systems

This section focuses on tools and workflows used to create the appearance of the game through lighting, shading and material effects. Besides the Animator tool, Unity has another visual node-based tool that eliminates the need to manually write code. That being the Shader Graph which is useful for creating shaders. Shaders are small scripts that contain mathematical calculations and algorithms for calculating the colour of each pixel rendered, based on the lighting input and material configuration. In *Figure 51* the DeepFloor shader's creation with Unity's Shader Graph tool is depicted. By combining mathematical nodes for time-based animation, color blending and depth manipulation, the shader dynamically adjusts its visual properties in real-time, producing immersive and customizable effects. Given its complexity, a decision during development was made to download the rest of the shaders used in the game.

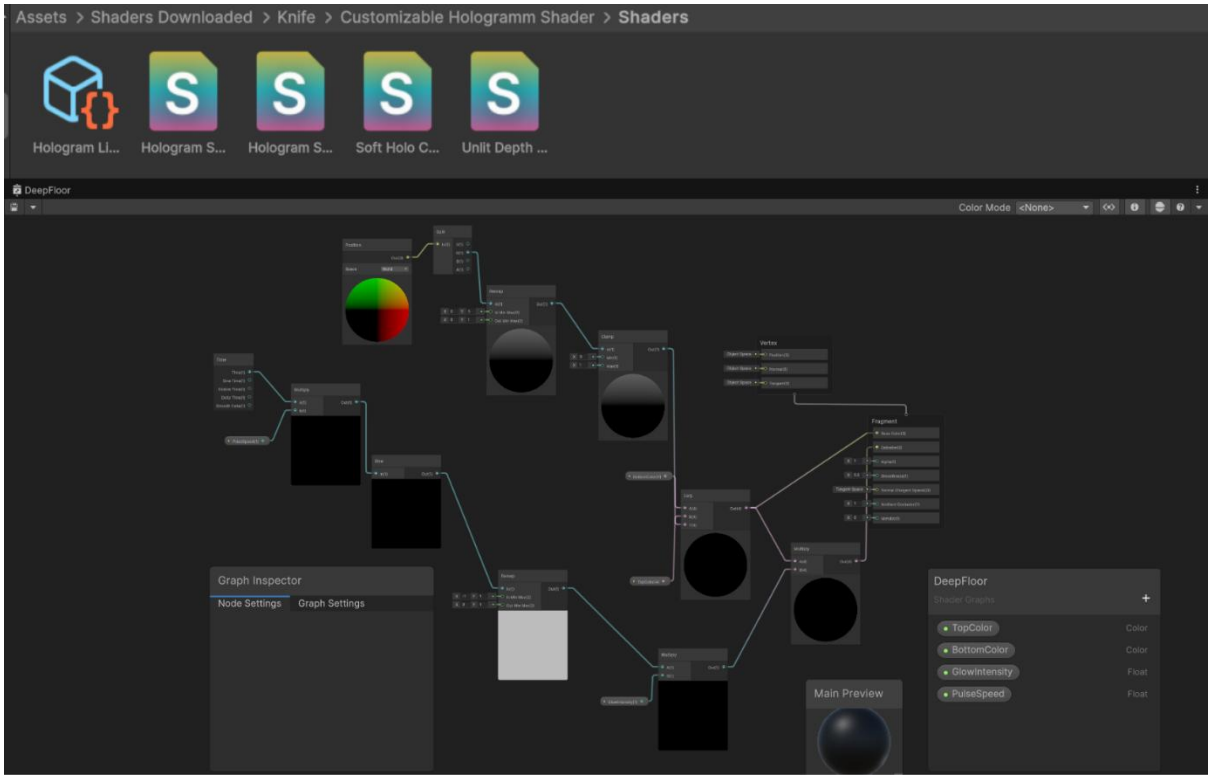


Figure 51: Downloaded Shaders and Unity Shader Graph Structure for the "DeepFloor" Shader.

These shaders are applied to materials within the game. For example, in *Figure 52* the holographic material utilizes the Hologram Shader Graph which was downloaded and imported into Unity. Materials are definitions of how a surface should be rendered, including references to textures and shaders used, tiling information, color tints and more. Textures are 2D maps that wrap around 3D objects to create variations in color, reflectivity and other properties. They provide materials with detail. All the textures were produced via external tools that were previously thoroughly analyzed.

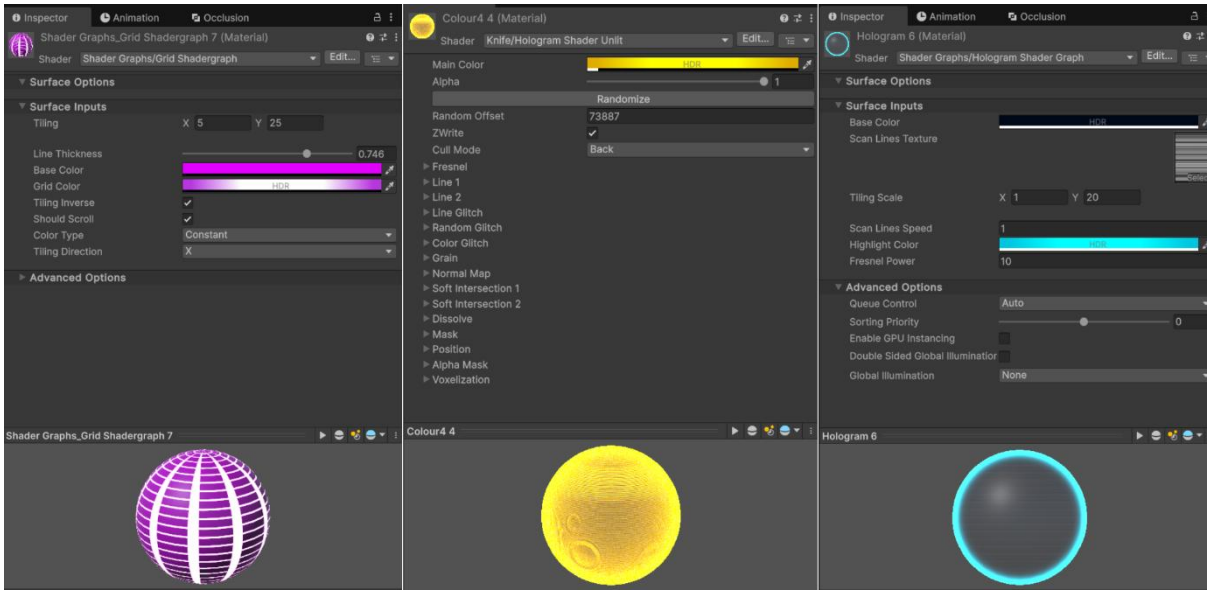


Figure 52: Material Creation by Several Different Shader Utilization.

There is an example of an object using a material in *Figure 53*. This material's properties specifically, Metallic Map and Smoothness have been adjusted to achieve the desired appearance. The shader chosen for this material is of the Universal Render Pipeline and particularly Lit. This implies that it responds to scene lighting based on its material properties. The texture used to create the material is also presented in the far left of the *Figure 53* illustration.

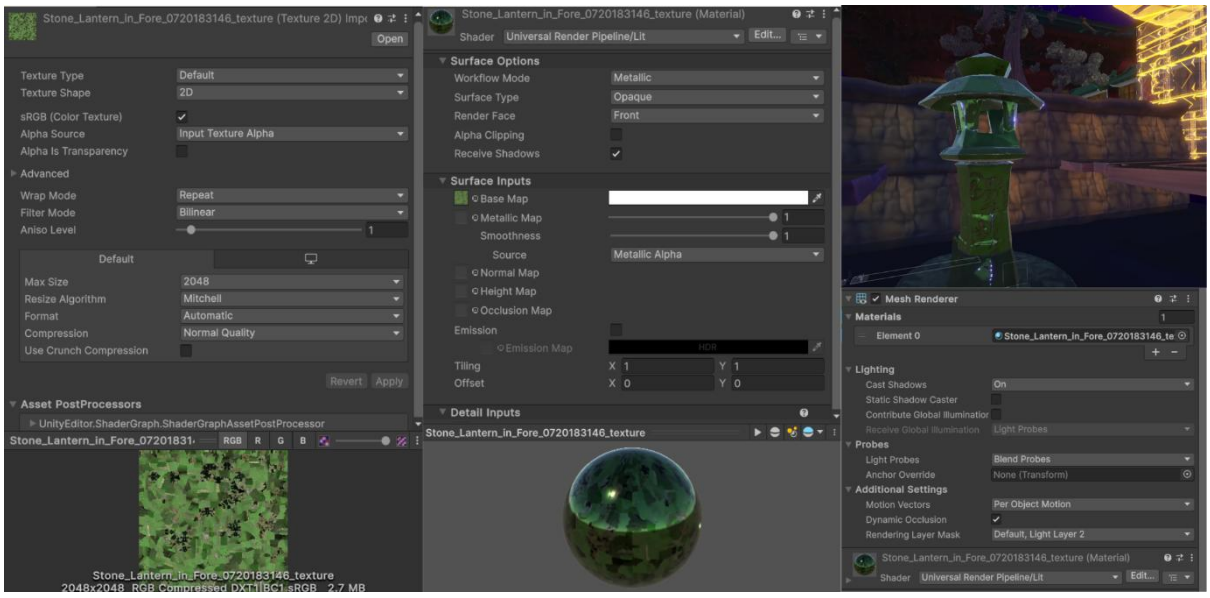


Figure 53: Assigning a Material to the Stone Lantern GameObject.

Shaders, materials and textures define the appearance of individual objects. The Universal Render Pipeline (URP) and Unity's lighting system are responsible for rendering level environments and for handling illumination processes.

To be more specific, URP governs the rendering of shaders, lighting calculations and post-processing effects while Unity's lighting system determines the visual perception of objects and environments. The ability to interact realistically with various light sources, producing accurate reflections, shadows, and color blending was the reason for choosing URP over any other Render Pipeline. In Unity, clipping planes define the visible range of the camera. In order to further optimize performance, I set clipping planes to "Far 1000". This means that objects beyond 1000 units are skipped. In *Figure 54* the Quality settings and Camera settings are displayed.

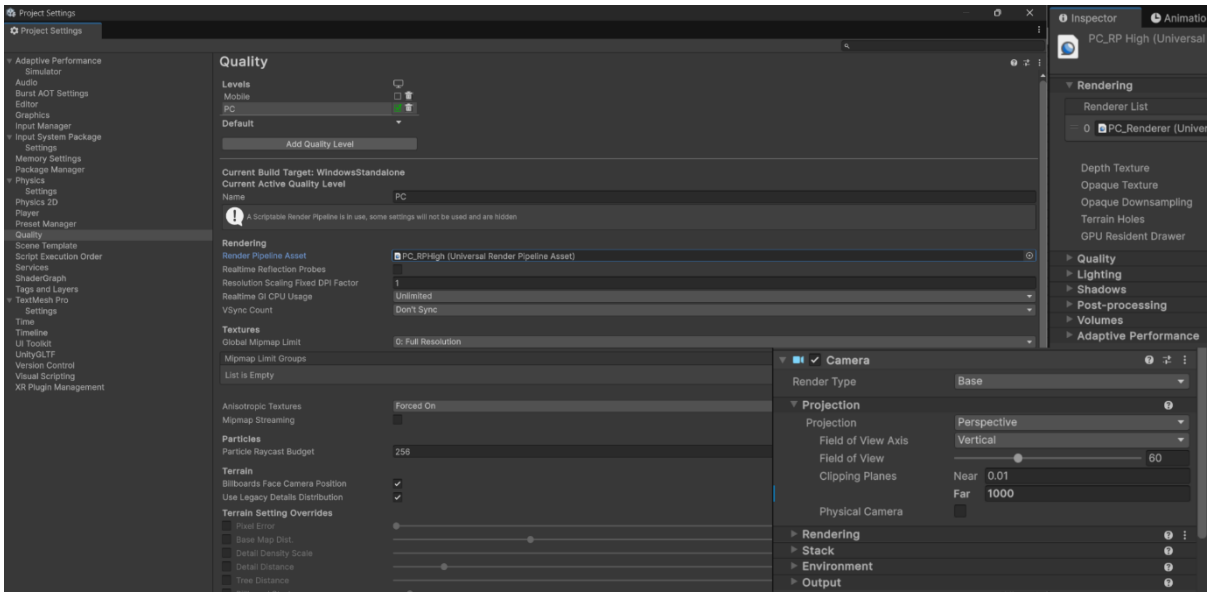


Figure 54: Quality and Camera settings.

In *Figure 55,* two of the core components of the lighting system and their settings are presented. Specifically, real-time directional lights affect the entire scene equally while the light updates continuously during gameplay. When configuring the mesh rendering of a GameObject, there are rendering layer masks as shown at the bottom right of *Figure 53*, that act as filters controlling which real-time directional light source affects the GameObject's appearance. Different light sources were utilized to create a cohesive visual environment by highlighting specific objects.

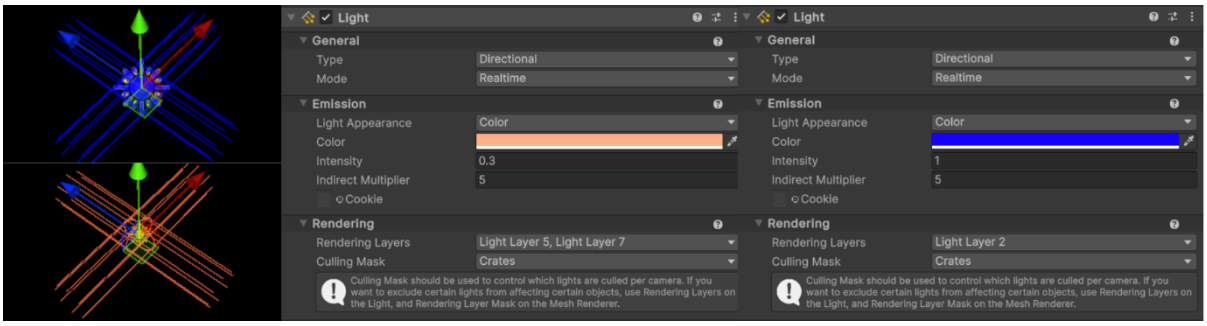


Figure 55: Realtime Directional Light Sources configured for Scene Objects.

Every level in the game needs a skybox to create the illusion of a distant environment beyond the playable area. In Unity there are four main types of skyboxes. I have utilized three of them so far. The one that has not been used yet is the six-sided skybox, which combines six individual textures to map a net layout. For the level Colour Agony, a panoramic skybox was utilized. A panoramic skybox uses one equirectangular 360° texture. While making the skybox, the texture slot remained unassigned. Instead, the tint color was set to black, offering a point of contrast to the surrounding vivid effects. For the level Bonsai Ruins, a cubemap skybox was utilized. A cubemap skybox contains one single cubemap texture which contains all sides. The cubemap was downloaded and imported from the assets store provided by Unity. The rest of the settings were adjusted to fit the new environment of the level. Finally, a sandstorm skybox has been created for a future level which uses a procedural skybox. A procedural skybox is an algorithmically generated sky with several configurable settings elaborated comprehensively in *Figure 56*. For future levels custom skyboxes might be created using shaders.

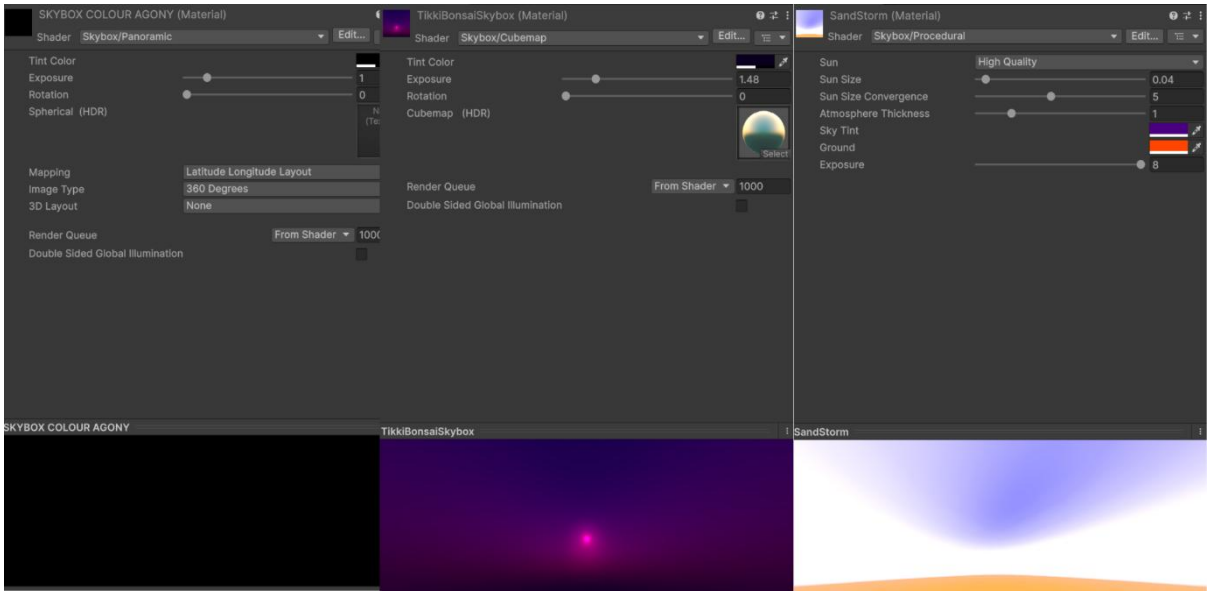


Figure 56: Examples of Different Types of Skyboxes.

A Particle System is another in-built Unity toolset. It enables developers to generate and control dynamic visual effects to enhance visually an asset. This tool can be configured to create any visual effect imaginable. However, before creating a particle effect, a goal must be formulated. The first particle system created for the game was a particle effect surrounding the color time-based platform. The goal was to enrich the platform's presentation. Other particle effects were formed to achieve the same aim, which is aesthetic improvement. In *Figure 57* examples of this purpose are depicted.

Figure 57: Examples of Particle Effects for Visual Enhancement.

An additional application of particle effects is the visual representation of state transitions or the completion of an action. Defining examples of state transitions are checkpoints, the invincibility state and the respawn sequence. Exemplary case of an action completion is the collection of rewards such as gems, crystals and Voodoo Dolls or resources like Mojo, Wumpa Coins and Uka Uka power. Another action is activating the exclamation point crate to unlock new pathways. Other actions include enemy spawns and booster or bounce crate effects depending on the player's input as presented in *Figure 58*.

Figure 58: Particle System Creation for State Transitions and Action Completion.

An alternative use of particle effects was to demonstrate the presence of dangerous hazards. To be more specific, this form of particle effects aligned perfectly with a scripted damage zone. As shown in *Figure 58* representative examples include, the radius of the TNT, NITRO and POW impact area, the collision zone of the flames produced by the campfire, the ant enemy's sword and the laser beams. The TNT explosion particle effect will be a demonstration of the Particle System and its settings.

Figure 59: Particle Effects utilized for Hazardous Damage Visualization.

Unity's Particle System Tool provides a wide variety of available parameters. In *Figure 60* the goal was the development of a cartoony TNT explosion effect. There was a vision prior to its creation. Particularly, flying TNT pieces along with a coordination of the color red, yellow and orange embodied the vision. So, there were two separate particle systems created, joined by a single parent GameObject.

The most important parameters to consider were the emission, which is the rate over time and bursts of a particle, the shape, for both particle systems a sphere was selected, the initial state, which are all the settings shown in *Figure 60* and their visual form, which is the renderer that includes any material. These materials use a specific Particles/Standard Unlit shader and single 2D sprites. The external tools of the previous sub-section were utilized to create two sprites. For the first Particle System the sprite was a copy of the already used TNT texture creating an illusion of flying TNT pieces. The latter Particle System implemented a smoke sprite. To create a brighter effect, the smoke's material settings, particularly the emission color's intensity has been modified as illustrated in the top right corner of *Figure 60*.

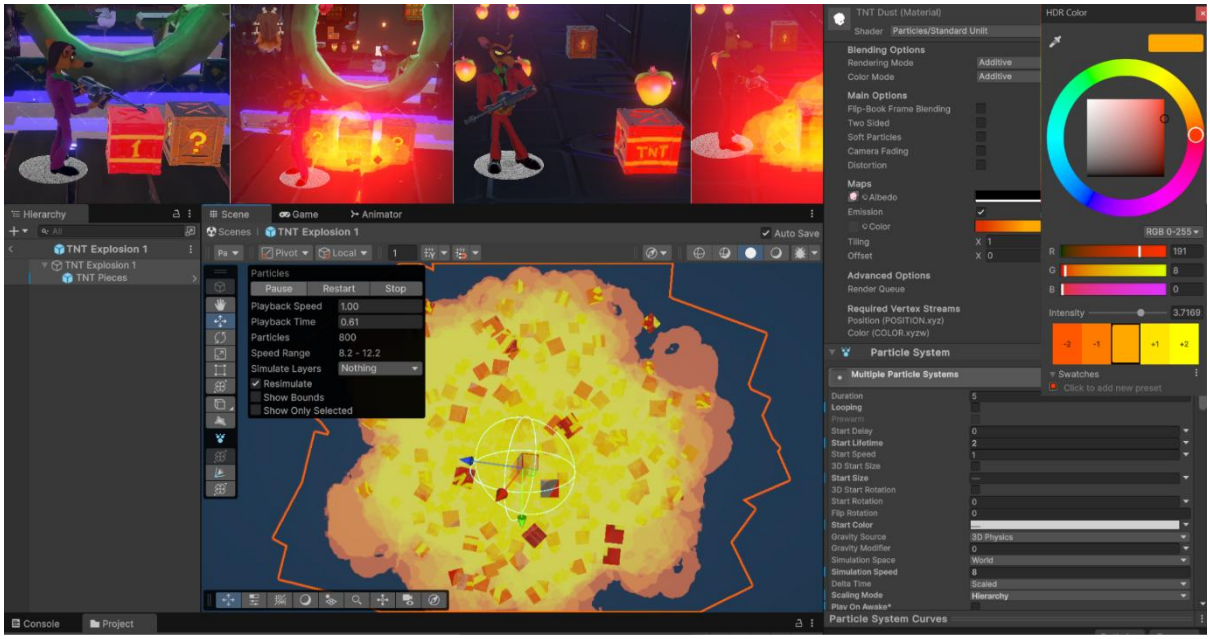


Figure 60: TNT Particle Systems Development.

3.3.3 UI and Feedback Systems

The User Interface (UI) is a fundamental component of game design since it provides essential visual feedback. In this project, multiple Unity's built-in systems, particularly Canvases, TextMeshPro and Animator tools were utilized to organize UI elements according to their purpose and context. In *Pinstripe Potoroo: The Black Gem Conspiracy* the main Heads-Up Display (HUD) Canvas communicates information about current health level, remaining ammo, the Uka Uka charge meter and the collection of rewards and resources as visualized in *Figure 61*.

Figure 61: Main HUD Canvas.

The complete Canvas is overloaded. Therefore, collectibles have been programmed to appear at the exact point in time they are obtained. Otherwise, they are visible only if collected and a specific key is held. Moreover, Unity's Rect Transform component was extensively used, applying anchoring strategies so that the UI components maintained their relative positions in various resolutions.

While Unity provides the developers with a text system, TextMeshPro was chosen for all textual elements due to its superior rendering quality and advanced customization features. Speaking of customization, I found a fan-made crash-a-like Font Asset. In *Figure 62*, a secondary canvas was created for the holographic gem barrier. It has been programmed to unlock only when all crates are collected in the level. To show the number of crates collected and the total crates in the level, TextMeshPro with the downloaded crash-a-like Font Asset were applied.

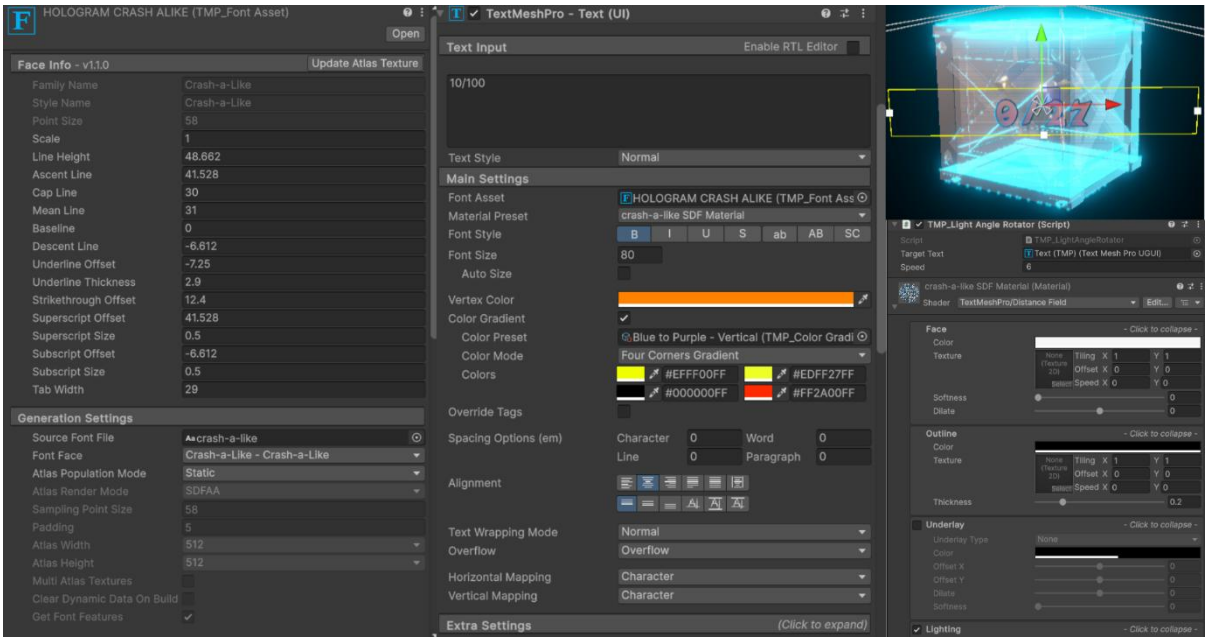


Figure 62: TextMeshPro Implementation for the Hologram Gem Barrier's Crate Counter.

Visual cues were provided with animation of the UI components. Using Unity's Animator System previously described in section 3.3.1 and spritesheets created with external tools described in section 3.2.1, were integral to this process, enabling smooth transition animations. For instance, in *Figure 63* the development process of the Uka Uka charge meter development is demonstrated.

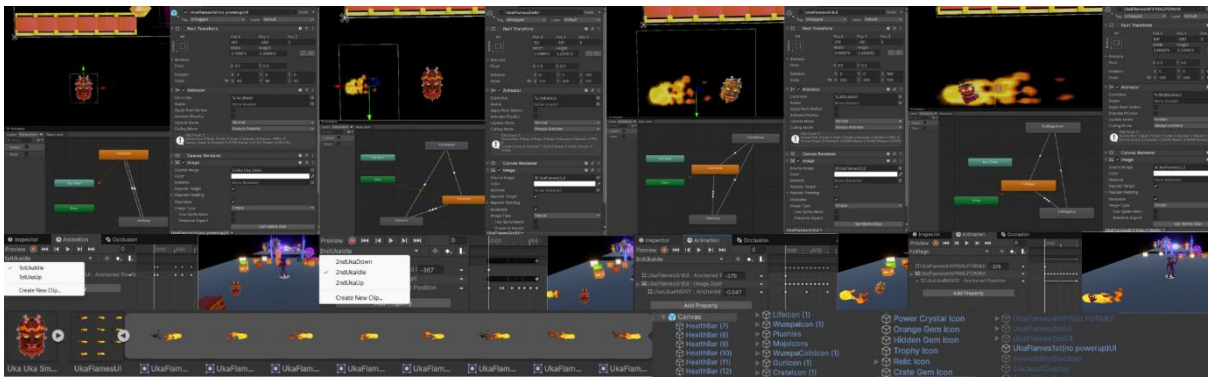


Figure 63: All Development Process and Tool Utilization of the Uka Uka Charge Meter Creation.

4. Game Design

This chapter is intentionally structured in the style of a developer's manual. Its purpose is to catalogue the game's mechanics, systems, and progression in a clear and systematic way, much like a player's guide. By presenting the design in this format, it bridges the gap between gameplay experience and the later analytical modeling in Chapter 5. In other words, this "manual-style" approach ensures that the reader understands the full scope of mechanics before seeing how they can be analyzed and optimized through game theory.

4.1 Overview of Gameplay Pillars

Pinstripe Potoroo: The Black Gem Conspiracy's gameplay is designed around platforming and combat. Where platforming manages movement and navigation, combat manages the game's interaction with enemies and obstacles actively acting against the player. The game adopted the best from previous platformer Crash Bandicoot titles focusing more on timing, accuracy and precision-crafted sequences of environmental challenges.

Pinstripe's platforming style consists of basic navigation, crouching and jumping. It is less intricate than Crash's platforming style, making the handling of the complicated combat mechanics more manageable.

Pinstripe's combat style consists of strategic resource management and a combination of third-person shooter mechanics with beat' em up gameplay elements. Combat sequences are created to complement platforming sequences rather than replacing them. For example, in several instances Pinstripe shoots at enemies while crossing between platforms.

Both platforming challenge design and combat challenge design are planned to be constructed on a scale of difficulty, gradually introducing new players to mechanics. For example, early stages incorporate simpler platforming challengers such as static platforms and narrower gaps as well as fewer and less dangerous enemies, providing a danger-free space to master the skills. Later levels introduce dynamic hazards, such as moving platforms, crumbling bridges, wall climbing sections and an increased number of enemies requiring more advanced decision-making. The combat system deepens throughout the game as well, by upgrading abilities and by introducing new mutant categories, keeping players interested while gradually ramping up complexity.

4.2 Progression and Level Structure

The player's overall journey within *Pinstripe Potoroo: The Black Gem Conspiracy* is defined by progressing through the game's storyline or by the pursuit of full completion. The predominant way to progress is by retrieving the main collectibles from levels.

4.2.1 Main Collectibles

Figure 64 illustrates the rest of the main collectibles present inside the levels.

Figure 64: Screenshots of all Main Collectibles in Pinstripe Potoroo: The Dark Gem Conspiracy.

The most important for story progression is the Power Crystal. Every level contains a Power Crystal. Collecting twenty-five power crystals provides sufficient amounts of energy to activate a dimension portal that grants entrance to the hidden location of the final boss fight. It is the only collectible necessary to experience the game's main campaign.

The next set of collectibles are the gems. There is the clear gem which is collected by breaking all the crates inside a level. In each level, there is one clear gem and also one hidden gem. Another category of gems is the colored gems. There are exactly six in the whole game, one designated to each primary color along with their combinations. They are retrieved at the end of Death Routes. There are also Power Gems. Each one is constructed by the energy of five power crystals, unlocking a new area. The final type of gem is the black heart power gem, which belongs to Von Clutch. It has been stolen, initiating a quest to retrieve it with Pinstripe's assistance.

Another form of collectible is the Voodoo Dolls. In every level there is one of each type. There is the Voodoo Dolls that require defeating all enemies who visually match the design of the voodoo doll, the Hidden Voodoo Doll, the Voodoo Doll acquired after successfully finishing a Mojo Room challenge and the Crash Voodoo Doll which is collected after finding and chasing down the egg thief as seen at the bottom of *Figure 64*. Collecting all previously mentioned Voodoo Doll types in a level rewards the player with the Golden Voodoo Doll.

The final category of main collectibles is the Time Relics. Upon activating the level's time trial mode by touching the floating clock, they are obtained only if the player finishes the level within a target time. The lowest-ranked time trial reward is the Sapphire Relic, the next rank is the Gold Relic, the following rank is the Platinum Relic, and the superior rank is the completely original Gerba Relic which is the developer's time. In the event that the player secures all collectibles in a level, a Golden Trophy is rewarded as the last main collectible.

4.2.2 Other Collectibles

The most popular collectible from the Crash Bandicoot games is the Wumpa fruit whereby collecting one hundred of them grants the player an additional life. Lives are essential for continued progression and are collectible items themselves. There is also the golden Wumpa fruit. It has a value of fifty compared to a single Wumpa fruit which has the value of one. In the game every level has both Wumpa fruit, crates and a floating clock. Collecting the floating clock activates Time Trial Mode. There is the plain Wumpa crate which upon being broken grants one Wumpa fruit. The question mark crate, if broken returns a random value between one and three. The life crate grants one extra life. The iron up arrow crate which, upon being jumped on grants the player an enhanced jump. The wooden up arrow crate which has the function as the previous one, but it can be broken upon contact with a bullet or close-range attack. The exclamation point crates, which will enable objects or other crates within the level. The iron checkpoint crate when activated, its position becomes the designated respawn point. The TNT crate which, upon being jumped on initiates a reverse countdown starting from three and its explosion deals damage to the player and their surroundings. The POW crate which functions just like the TNT differing in that its explosion has no damaging effect to the player. The Nitro crate which explodes instantly upon contact, damaging the player in the process. Lastly, there is the iron crate which acts as a normal platform. There are also the resource management crates consisting of the Mojo crates, Wumpa Coin crates and Uka Uka crates. They are described in a detailed manner in throughout the subsequent sections of the chapter. The system supports multiple categories of crates and additional crate types are planned for future implementation.

Figure 65: In-Game Screenshots of Other Collectibles.

4.2.3 Level Types and Special Challenges

The game features two primary level types. The first type is levels dedicated exclusively to platforming and combat. The second type is levels that integrate various vehicle types like the rideable robotic alien enemy in *Figure 66*. Both level types possess the same set of collectibles and objectives. The main section of the level contains the hidden gem, hidden voodoo doll, and Power Crystal. Exploration beyond the main area is not required for any level. Each level contains at least two additional distinct areas. Specifically, a Mojo Room and a classic Bonus Round.

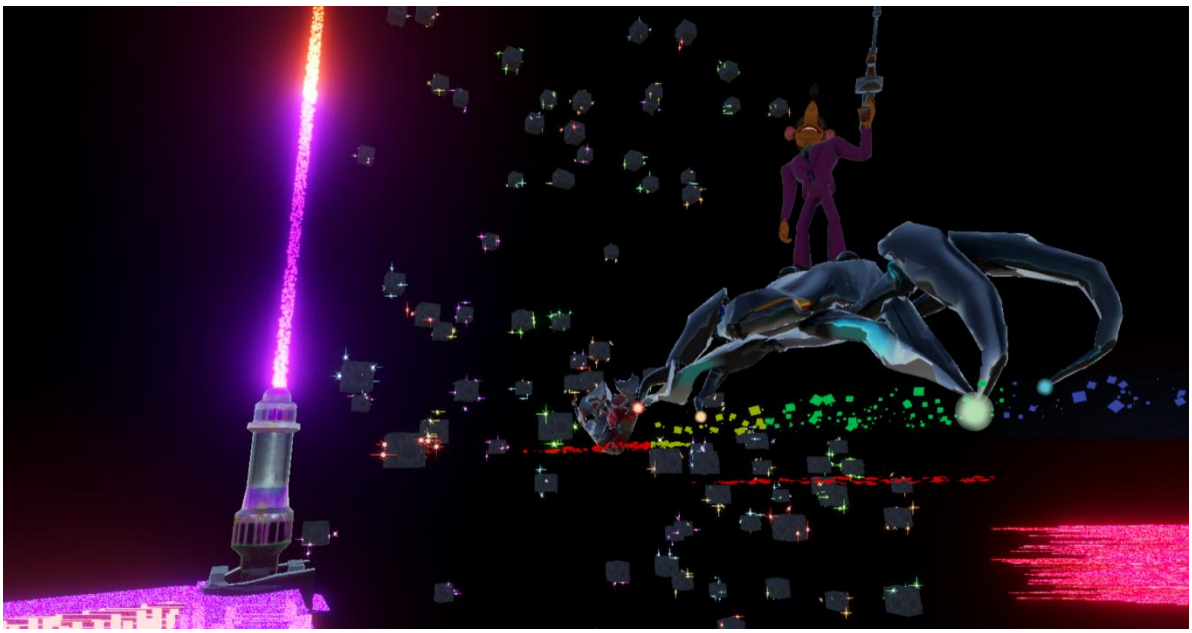


Figure 66: Rideable Enemy in Colour Agony's Death Route.

The Mojo Rooms consist of special challenges, rewarding the player with a Voodoo Doll. Some examples of special challenges are the elimination of floating jet bombs, the collection of all Green Mojo and the total elimination of enemy forces under a strict time constraint. Every level incorporates one Mojo Room. It is accessed through finding floating orbs in the main section of the level. By pressing a key while being inside the surrounding area, the player instantly teleports to the Mojo room.

Figure 67: Mojo Room's Floating Orbs Entrance in Bonsai Ruins.

The Bonus Round is an optional, separate section of a level, free of enemies that serve as an area for players to collect extra Wumpa Fruit, lives and other resources. It only contains difficult platforming and specific challenges related to gameplay mechanics utilization. Much like Mojo Rooms, every level incorporates one bonus round with difficulty gradually increasing over the course of the later levels.

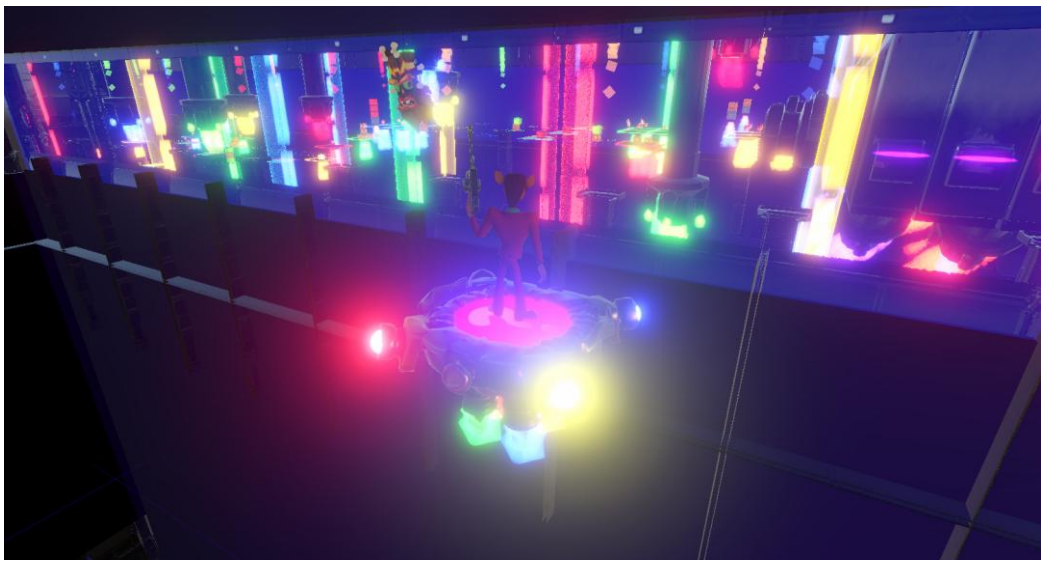


Figure 68: Bonus Round Platform in Colour Agony.

Some levels may also feature alternative routes namely Gem Routes or Death Routes. To be more specific, Gem Routes are unlocked by obtaining the required colored gem associated with them. At the end of a Gem Route there is the Hidden Gem remaining to be collected. The colored gems are located at the end of Death Routes. Both Gem and Death Routes are level segments that present greater difficulty than the main section of the level.

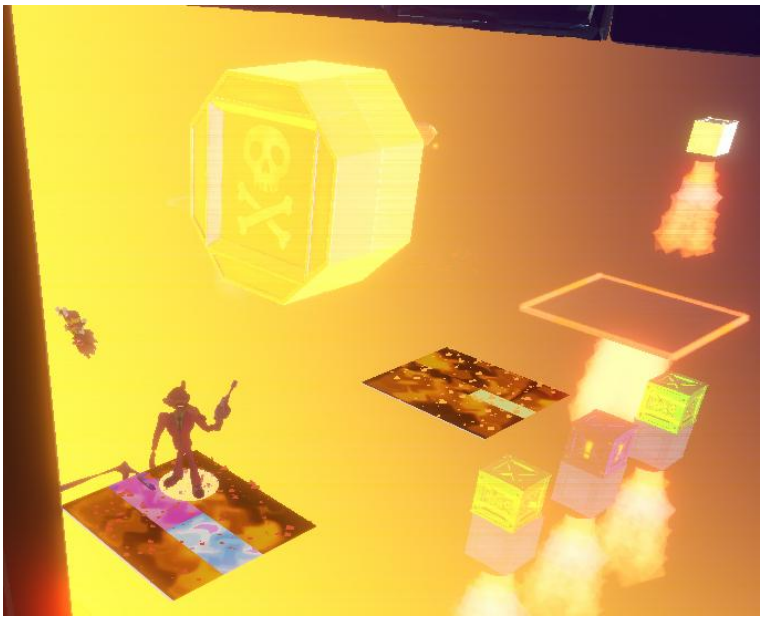


Figure 69: Death Route Entrance in Colour Agony.

Figure 70: Locked Gem Route Platform in Bonsai Ruins.

4.2.4 Narrative Progression and Hub World Design

The game starts inside Pinstripe's office after his defeat at the hands of Crash Bandicoot. A cutscene will play showing increasing tension between Pinstripe and his Mafia members. Upon the cutscene's completion, the game transitions into an instructional stage on combat mechanics, beginning with a gunfight between Pinstripe, who is the main character the player will control for the biggest portion of the game, and the Mafia members who will be the first enemy type. After finishing the combat system tutorial, a rift generator emerges.

Figure 71: Screenshot of Pinstripe's Office in Crash Bandicoot N Sane Trilogy.

In this timeline the MotorWorld is a theme park under construction and remains inaccessible to people. Only workers in the form of Park Drones are inside. People that served as NPCs roaming around the main MotorWorld hub area are absent. The entrance of the MotorWorld serves as an introductory tutorial on the game's core platforming mechanics. When the player has successfully entered the MotorWorld, it is required to first navigate through the central hub area and find the first themed land. The MotorWorld's central hub area contains five different themed lands. Inside each land there are five levels. Generally, levels are spread around the area. The player can enter any of the five levels. Each hub area contains one boss fight. The boss fight is inside an area which is accessed only by jumping on a specific booster pad. The booster pad is unlocked by collecting all the Power Crystals of the themed land's levels. After defeating the boss, a new ability is unlocked, and a power gem is rewarded. Each Power Gem collected serves as a key to opening the rest of the theme lands. A quantum rift leading to the final boss fight is unlocked after retrieving all 25 Power Crystals. To finish the main campaign of the game, the player must defeat the final boss.

Figure 72: Concept Art of an Overview of the MotorWorld's Landscape.

Figure 73: Screenshot of Von Clutch's MotorWorld's Entrance from Crash Tag Team Racing.

Figure 74: Park Drone Models created in MeshyAI to import in my version of the MotorWorld.

4.3 Resource Management

Resource Management is a core strategic layer of tactical depth shaping how players interact with the game's challenges and progression systems. To be more specific, there are four main types of resources that are managed by the player and the decisions made are fundamental to achieving success throughout the adventure.

4.3.1 Wumpa Coins

Wumpa Coins function as the game's primary economic currency, connecting level exploration to long-term progression through upgrades. They are found throughout the MotorWorld in crates and hidden explorable areas. Once collected by the player, the Wumpa Coins and crates dispersed across the area are permanently removed until the boss of the area is defeated. Defeating the boss fight again does not cause them to regenerate. Instead, smaller amounts are granted to the player upon level completion.

Park Drones handle Wumpa Coin transactions for shooting upgrades such as increased health or ammo capacity, faster ammo regeneration, mojo power capacity and extended shooting range. Players must decide which upgrades to prioritize. The placement of Wumpa Coins in theme lands directly ties into level design strategy. There are hidden caches of coins, encouraging exploration and replayability. They create a feedback loop where successful performance in levels directly translates into enhanced future capabilities reinforcing the progression system introduced in Section 4.2.

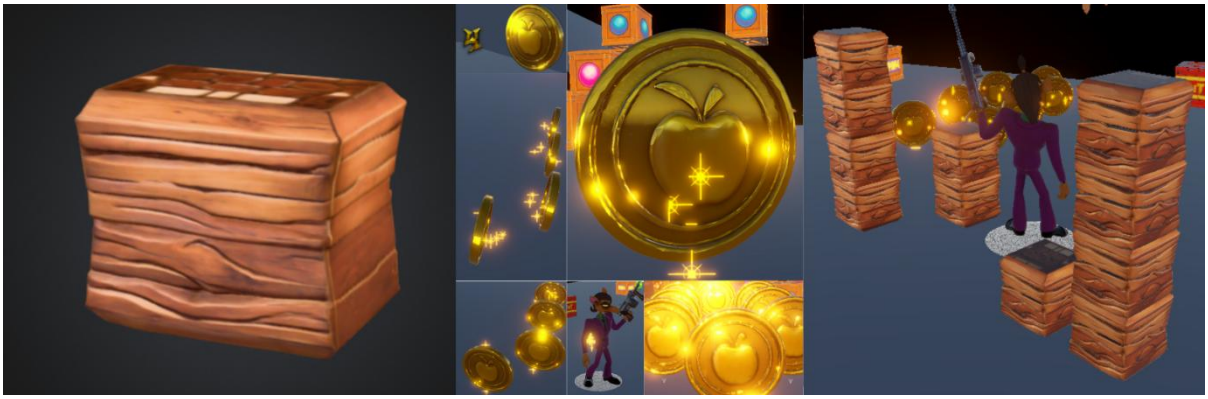


Figure 75: Wumpa Coins and Wumpa Coin Crates Inside the Game's Test Level.

4.3.2 Mojo Power

Mojo serves as a consumable energy source used for bomb mechanics. Players begin with a capacity of eight hundred Mojo Power, which does increase to one thousand six hundred Mojo Power through upgrades purchased with Wumpa Coins. Each bomb thrown consumes one hundred Mojo Power, adding tension to combat encounters by limiting the frequency of bomb usage. Mojo is obtained by defeating enemies, breaking specific crates, or completing Mojo Room challenges.

There are four types of Mojo in the game. Blue Mojo represents the lowest tier, providing the player with a value of one. Pink Mojo grants five times the value of Blue Mojo. Yellow Mojo provides ten times the value of Blue Mojo. Green Mojo is excluded from the primary Mojo Power system. It only appears in Mojo Rooms, where collecting all Green Mojo within the Mojo Room rewards the player with a Mojo Room Voodoo Doll and a prize valued at one hundred Mojo Power.

There are four types of Mojo Crates as well. The Blue Mojo Crate contains a random value between three and five of blue mojo. The Pink Mojo Crate contains a random value between three and five of pink mojo. The Yellow Mojo Crate contains a random value between three and five of yellow mojo. The Multiple Mojo Crate contains three blue mojo, two pink mojo and one yellow mojo. Each of those mojo are multiplied randomly from two times to four times.

Mojo introduces short-term resource management, as players must balance offensive capabilities with conservation. By using bombs aggressively, enemy groups are quickly cleared but risk leaving the player defenseless later in the level. The only way to terminate Mutant type enemies is by stunning them and with a nearby explosion ultimately clearing them. So, careful timing and precision in bomb deployment can turn Mojo into a powerful strategic tool.

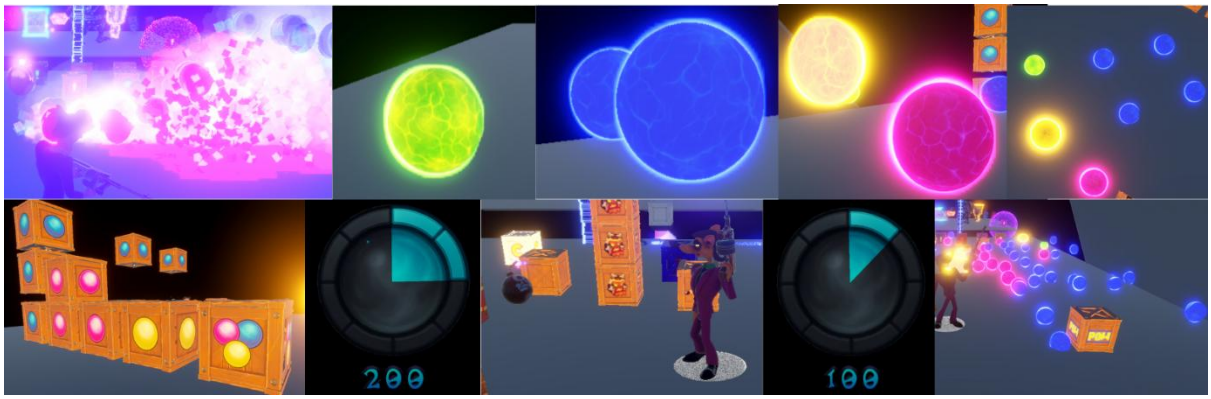


Figure 76: Mojo Power UI Counter, Different Mojo and Mojo Crate Types.

4.3.3 Ammo

Ammo is the resource for the ranged shooting attacks. Players begin each level with a base amount of ammo, which regenerates automatically over time, providing a continuous but limited flow of firepower. The ammo base capacity is five bullets but, through Wumpa Coin transactions, can be upgraded up to twenty bullets. Ammo regenerates at a fixed rate to prevent players from running completely out of ammunition during combat but also from being overpowered. Upgrades do increase the regeneration speed.

The player must decide when to hold fire and unleash rapid shots, taking into consideration both the remaining ammunition and the other usable abilities. During the Uka Uka Invincibility state ammo becomes infinite, encouraging aggressive playstyles for a limited time.

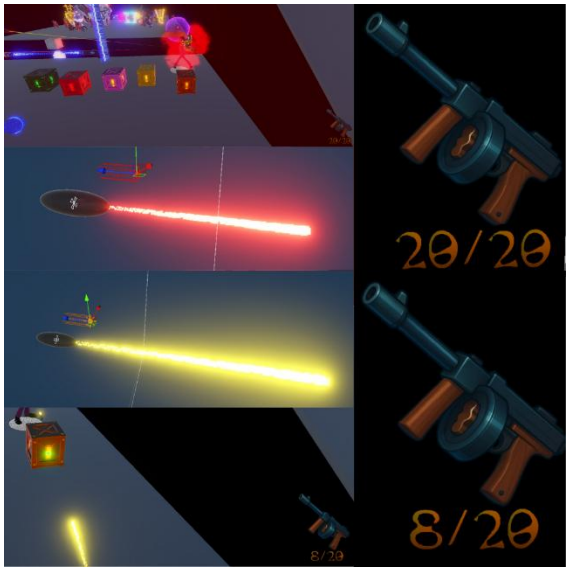


Figure 77: Bullets, Ammo UI Counter and Shooting During and Outside of Uka Uka Invincibility.

4.3.4 Uka Uka Invincibility

To conclude, there is also a power-up that heightens the combat dynamics while providing the player with invincibility over a limited timeframe. Players collect Uka Uka crates scattered throughout levels, with three crates required to fully charge Uka Uka. Then based on their judgement of necessity, they decide to press a specific key triggering the Uka Uka Invincibility for twenty seconds. This power grants complete invulnerability, infinite ammunition and increased movement speed. Activating it too early is wasted potential against more difficult circumstances. Although, if Uka Uka is fully charged and the invincibility state is not yet activated, the power of an additional fourth Uka Uka crate will remain unutilized. This system mirrors high-value resource allocation of the real world, making it an ideal candidate for Game Theory modeling.

Figure 78: Correct Use of the Uka Uka Invincibility.

4.4 Core Combat Mechanics

This game features two distinct, switchable combat systems. The first is Aim Mode, which is useful for precise targeting and long-range combat. The latter is No-Aim Mode, which includes a close-range attack and the release of bombs without the assistance of targeting. The gameplay encourages the player to shift between them fluidly depending on the situation.

4.4.1 Aim Mode

The central mechanic in this context is the ability to set a target on a specific GameObject, within a range determined by the distance between the GameObject and the player. This range is determined by a predefined, invisible radius which is visually indicated to the player via visible small green dot-like particles appearing immediately on interactable GameObjects upon entering the defined radius. The radius is capable of increasing in size to a certain extent, as mentioned in section 4.2.3, it is a type of upgrade but at the expense of the Park Drones' Wumpa Coin gains. The bright red particle effect, which includes sprites of a crosshair, enables clear visualization of the target positioned

closest to the player. The bright yellow particle effect supports the identification of the target positioned second closest to the player. By means of specific key press, the yellow target is promoted to the red target status. The mechanic is demonstrated in *Figure 79 and Figure 80*.

A hierarchical targeting system for priority-tagged GameObjects has been implemented through the "TargetingSystem" script. The current hierarchy of target tags is as follows: first Checkpoint, second FinalTarget, third Ballz, fourth PurpleCrate, then Ant, followed by Mutant, and the concluding target tag is Breakable. This signifies that the system chooses the red indicator's target with the highest hierarchy ranking before considering distance.

Aim Mode integrates two targeting-based actions. The actions performed are shooting and bomb throwing. Two separate controls are mapped. The first input is used to fire at the closest target and the second input to release a bomb toward the target.

Figure 79: Target System of Aim Mode and Shooting at the Targets.

Figure 80: Target System of Aim Mode and Throwing a Bomb at the Target.

4.4.2 No-Aim Mode

The functional role of this mechanic is to improve gameplay flow. Specifically, the player switches seamlessly between the two modes by pressing a specific key. No-Aim Mode consists of two different actions and uses a shared input mapping with the Aim Mode actions. The player performs a close-range attack, with the same input as long-range shooting, that only deals damage in the direction Pinstripe is facing. While the attack has a short cooldown, it requires no resources to execute, making it always available. However, its limited range and longer cooldown compared to Aim Mode's shooting make it a last resort in combat. Despite this, it is effective for breaking crates and temporarily becomes the primary attack as it is the remaining way to deal damage until ammunition regeneration occurs. Instead of throwing bombs, utilizing the same input in this mode, the player releases bombs. The bombs utilize Rigidbody Physics, allowing their movement to be dynamically affected by the player's movement. An illustration of the bomb's lifecycle from initialization to detonation is given in *Figure 81*.

Figure 81: Releasing a Bomb in No-Aim Mode and its Consequences.

Figure 82: Executing the Close-Range Attack in No-Aim Mode.

5. Game Theory Application

5.1 Introduction and Problem Definition

Game design could be characterized as a strategic system, where players are continuously faced with decision-making that affects their performance, progression and overall experience. Whether it involves movement, combat, exploration, or resource management, the player's decisions can be represented as a strategic choice within a game theory model. Much like the decision spaces analyzed in game theory, each level presents a network of possible pathways, actions and resource expenditures. This approach allows for a formal analysis of gameplay, where the relationship between player actions, rewards and penalties can be studied systematically.

Guardiola and Natkin (2005) propose viewing video games through the lens of classical game theory to both analyze and design game systems. They identify several fundamental concepts that are important in video game design field.

The classical game of the game theory is represented in a matrix form. Specifically, it has a finite set of players, numbers of possibilities in the payoff matrix and for each crossing of strategies in the matrix, each player receives a particular payoff [15]. In mathematical terms, there are two payoff matrices AB and BA of real numbers having the same dimensions (i,j). In *Pinstripe Potoroo: The Dark Gem Conspiracy,* when the player interacts with the environment, the player, player A chooses a variable between 1 and n, "i". The variable "i" represents one decision made by the player that is constituted by actions directed toward the environment. Simultaneously, the environment, player B responds with a variable between 1 and k, "j". The variable "j" represents the response of the environment which is independent from the sequence of actions the player had chosen. A zero-sum interaction occurs when the reward of one player is the loss of the other player [15]. If *AB(i,j)* indicates conquest for the player and *BA(i,j)* triumph for the environment, then the entries of the matrix product AB are related to those of BA according to the expression below, which holds for every row index i and column index j in the specified ranges:

$$AB(i,j) = -BA(i,j), \qquad \forall i \,\epsilon\{1,\dots,n\}, \forall j \,\epsilon\{1,\dots,k\} \tag{1}$$

Each enemy defeated, collectible acquired, or challenge completed represents a shift in advantage from the environment to the player. It is particularly useful for analyzing combat encounters and resource contests. For instance, when a player uses a bomb to destroy a mutant enemy, the resulting gain in safety and positional advantage corresponds to a loss in the environment's ability to

impose difficulty. Another strictly zero-sum interaction is level completion in which the player receives all collectibles and the environment absorbs all losses as illustrated in *Figure 84*.

Figure 83: Zero-Sum Dynamics Between Player and Environment.

Other Game Theory principles, important in video game design are dominant strategies, the Nash equilibrium and mixed strategies. A dominant strategy is a strategy "i" of the player, that provides a better outcome for the player regardless of the environment's response. For instance, in certain combat encounters or platforming sections consisting of many hazards, using the Uka Uka invincibility mechanic at a specific moment might always yield the highest survival probability. For that context, it is a dominant strategy. A Nash equilibrium, as introduced in section 1.2, represents a state in which no player can reformulate their strategy to improve their outcome without incurring a risk of loss. In the context of level design, this occurs when a player's chosen pathway, speed, safety and resource usage is optimal.

Nonetheless, many finite games do not have deterministic equilibria but instead players rely on probabilistic strategies, where players randomize their actions. The probability vector which defines the frequencies to play each individual strategy is termed in game theory as a mixed strategy [15]. However, it is important to distinguish a mixed strategy's randomness when performing actions, from imperfect information where randomization is employed from unknown optimal strategies. In *Pinstripe Potoroo: The Dark Gem Conspiracy* as the enemies and hazardous environment featured in the current levels always use fixed, predictable attacks, there is no strategic justification for randomizing one's actions. There is no mixed strategy in the game.

Instead, stochastic variation is introduced to account for uncertainty stemming from incomplete information when a player first plays the level. The game's levels consist of specific enemy placements, hidden resource locations, optimal pathways and timing-based platforming sequences. The initial playthrough is analogous to a Bayesian game, where strategies are formed under uncertainty. When players encounter an enemy, they're required to make an instantaneous decision, unaware of the next segment's degree of difficulty. They are expected to swiftly assess their resources and select the preferred course of action. This creates many different possible outcomes depending

on the decision made. For example, choosing to spend mojo on bombs for immediate survival may compromise later performance when those bombs are needed for bigger groups of enemies and mutants. A critical aspect of this problem is the evolution of player knowledge over time. Once the player learns the fixed attack pattern, the battle becomes deterministic.

Those fundamental principles create a foundation for modeling how players interact with the systems of a game. In *Pinstripe Potoroo: The Black Gem Conspiracy*, the player uses limited and renewable resources such as ammo, mojo, Uka Uka power, and wumpa coins as explained earlier in section 4.3. While Chapter 4 provided a detailed breakdown of the core gameplay mechanics and finite resources, this chapter builds upon that foundation by shifting focus to the interactions and trade-offs that emerge during gameplay. By analyzing the interactions and trade-offs that occur during playtesting, developers can assess how effectively enemies, platforms and resources are allocated.

5.2 Multiple Objectives and Nash Equilibria

In this section, a two-by-two payoff matrix is represented. Player A is a completionist meaning the player plays the game, sets a goal to collect everything, ultimately achieving full completion. Player B is a speedrunner meaning the player that plays the game, sets a goal to complete it as fast as possible. Simultaneously, they end up in a level's decision point. There, both players must either choose to avoid enemy hazards and boxes or shoot at them.

Table 2: Payoff Matrix Speedrunner VS Completionist

		Speedrunner	
		Shoot	Avoid
Completionist	Shoot	(1, 0)	(1, 1)
	Avoid	(0, 0)	(0, 1)

The payoffs are shown in (i,j) where the first number is the payoff of the completionist and the second number is the payoff of the speedrunner. Zero indicates zero gain and one indicates maximum gain. The players' payoffs are independent. Although this is formally a two-player game in a game-theoretic sense, both players possess strictly dominant strategies. The completionists dominant strategy is shooting while the speedrunner's is avoiding. The problem is modeled as two distinct optimization problems rather than as a strategic interaction analyzed through game theory.

Figure 84: In-Game Decision Point illustrating the underlying Rationale for Matrix Payoff Values.

Table 2 has a Nash equilibrium. It is unique because it requires no strategic reasoning if both players always accomplish their respective goals. Speedrunners sometimes cannot achieve the best possible times since the physics and mechanics of a game are vastly complicated. In gaming, new speedrun strategies are invented constantly by speedrunning communities. These players strive for optimal frame-perfect movement and action-performing. In general, to finish levels faster, speedrunners must adapt to the discovery of new shortcuts. Relevant to this case, interdependence between speedrunners and completionists was achieved via information updating. In a speedrun reaching the decision point in *Figure 84* initially, players avoided shooting, believing it slowed runs. After watching a completionist's playthrough and analyzed the enemy patterns, they realized time could easily be reduced by running straight, while shooting airborne. *Table 3* presents the payoff matrix for this strategic interaction between them. Game theory does not require the other player's action to change in real time therefore it is a game theory model not about direct, simultaneous choices like the Prisoner's dilemma; instead, it's about how one player's demonstration reshapes the other's payoff landscape.

Table 3: Payoff Matrix of Game: Speedrunner VS Completionist

		Speedrunner	
		Shoot	Avoid
Completionist	Shoot	(1, 0.9)	(1, 0.7)
	Avoid	(0, 0.9)	(0, 1)

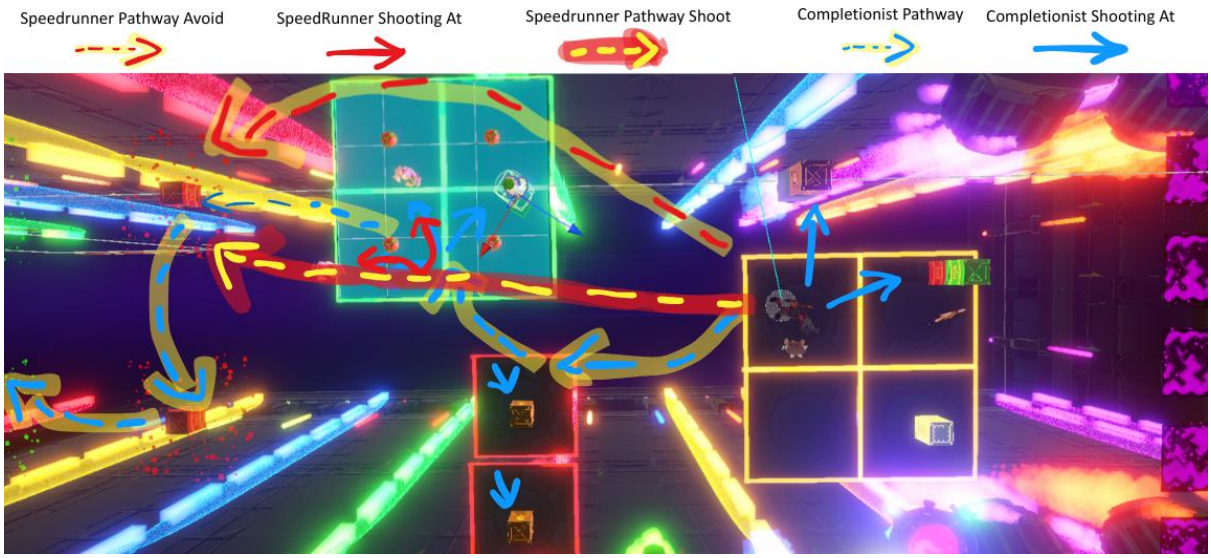


Figure 85: Decision Point illustrating Faster Pathway for Speedrun by Shooting Enemies.

Even after proving the "Speedrunner Pathway Shoot" is the least time-consuming pathway as illustrated in *Figure 85*, it also entails higher risk than "Speedrunner Pathway Avoid". This stems from the NavMeshArea present. It was mentioned in section 3.3.1 and serves as a baked navigation surface, where AI agents like enemies, roam around inside the baked NavMesh boundaries. This element incorporates a probabilistic component that determines whether the enemy will occupy a central position, thereby blocking the player's subsequent jump. Those 0.9 and 0.7 are just expected payoffs coming from the NavMesh risk. In order to explain the rationale behind those two payoff values, a model for the calculation of the "Speedrunner Pathway Shoot" expected duration has been created:

$$E[T_{shoot}] = (1 - q)T_{shoot} + q(T_{shoot} + L) \quad (2)$$

T_{shoot}: time if the "Speedrunner Pathway Shoot" is safe

$E[T_{shoot}]$: expected time for "Speedrunner Pathway Shoot"

q: probability an ant enemy blocking the jump

L: extra time loss if blocked

$T'_{shoot} = T_{shoot} + L$: time if if the "Speedrunner Pathway Shoot" is blocked by an enemy

The optimal pathway for the speedrunner depends solely on which route requires less time to complete. Accordingly, both shoot and avoid are equally preferred when:

$$T_{shoot} + qL = T_{avoid} \quad (3)$$

T_{avoid}: expected time for "Speedrunner Pathway Avoid"

The speedrunner selects shoot over avoid if:

$$T_{shoot} + qL < T_{avoid} \quad (4)$$

The speedrunner chooses avoid over shoot if:

$$T_{shoot} + qL > T_{avoid} \quad (5)$$

The NavMesh randomness ($q > 0$, $L > 0$) drags that 1 down to something like 0.9 or 0.7. If there is no enemy blocking the jump, meaning ($q = 0$) shoot would dominate as a strategy and the payoff matrix would take the form of *Table 4*.

Table 4: Payoff Matrix with Zero Risk: Speedrunner VS Completionist

		Speedrunner	
		Shoot	Avoid
Completionist	Shoot	(1, 1)	(1, 0)
	Avoid	(0, 1)	(0, 0)

Enemy and platform placement changes the risk q and penalty L of "shoot" vs "avoid," shifting which path is optimal for different players. By analyzing strategy choices of different player types at key decision points, a developer can adjust enemy and platform placement to keep multiple routes viable, thereby deepening level design. By ensuring at least two viable routes with distinct risk-reward profiles in level design, the developer engages both completionists and speedrunners by stimulating shortcut discovery and boosting replayability.

The entire problem between speedrunners and completionists is described better by two public information time states. The first state, state A occurs before having any information about the shortcut and the second state, state B is after.

Table 5: Stage-Contingent Information Game

State A	Speedrunner Shoot	Speedrunner Avoid
Completionist Shoot	(1, 0)	(1,1)
State B	Speedrunner Shoot	Speedrunner Avoid
Completionist Shoot	(1, 0.9)	(1, 0.7)

The completionist's dominant strategy is always Shoot. In State A (Shoot,Avoid) is a Nash equilibrium since avoiding is also the speedrunner's best strategy (1 > 0) and in State B (Shoot,Shoot) is a Nash equilibrium since shooting is always the speedrunner's best strategy (0.9 > 0.7).

Another example scenario is the decision the completionists and the speedrunners make to either follow the main path of a level or complete one of the bonus tasks. Completing a bonus task rewards the player with more collectibles. There are many supplementary objectives inside each level which were thoroughly analyzed in section 4.2.3. The payoffs of each decision, whether to enter or avoid the challenges, are organized as a two-by-two payoff matrix. This problem is modeled as two distinct optimization problems as well, not analyzed through game theory. *Figure 86* illustrates the decision points of entering different challenges or avoiding them throughout the level.

Table 6: Payoff Matrix of Bonus Tasks Speedrunner VS Completionist

		Speedrunner	
		Enter Challenge	Avoid
Completionist	Enter Challenge	(1, 0)	(1, 1)
	Avoid	(0, 0)	(0, 1)

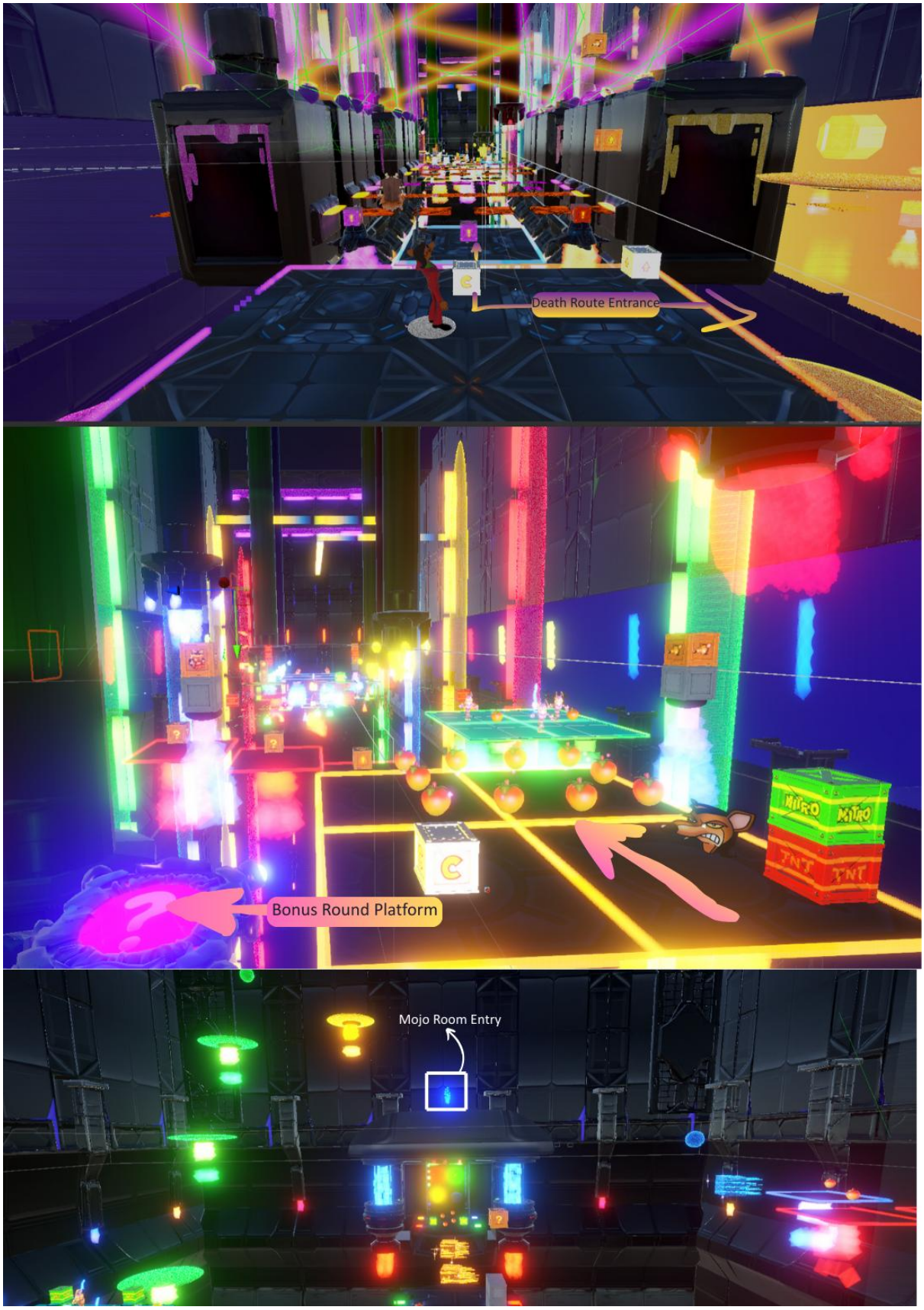


Figure 86: Entry Points of Supplementary Objectives.

5.3 Resource Management Payoff Structures

As demonstrated in the previous section, in this game there are many different ways of completing the same level. However, the approach players take to complete levels does not always fulfill their goals. For instance, an average player may attempt a speedruning strategy but their performance is suboptimal compared to that of an experienced speedrunner due to inefficient decision-making regarding actions and movement sequences. Likewise, the average player attempting to obtain every collectible of the level may demonstrate suboptimal resource allocation, leading to repeated failures. The scarcity of the game's manageable resources introduces strategic trade-offs, forcing the player to make meaningful decisions about when and how to use them.

The goal for this section is to analyze player strategies regarding resource utilization throughout various decision points of the level. After testing the game repeatedly, by having more Uka Uka invincibility states throughout the level, the gameplay flow improves significantly. Keeping that in mind, developers do not strive to design their game rewarding abusers or extreme hoarders. The optimal strategy is to activate Uka Uka at a critical point of the level.

The level selected as a case study is Colour Agony which is the final level before the boss fight of the second theme land "Happily Forever Disaster". Nine Uka Uka crates in the level are justified, since the level is large in scale, with the main path itself being considerable in length. Since Uka Uka may already be in the second phase before entering a level, either from breaking Uka Uka crates in previous levels or from skipping some Uka Uka crates, every single Uka Uka crate becomes a decision point. A payoff structure was created for the first decision point, where the player for the first time has a fully charged Uka Uka. *Figure 87* shows the first decision point of Uka Uka invincibility usage dilemma.

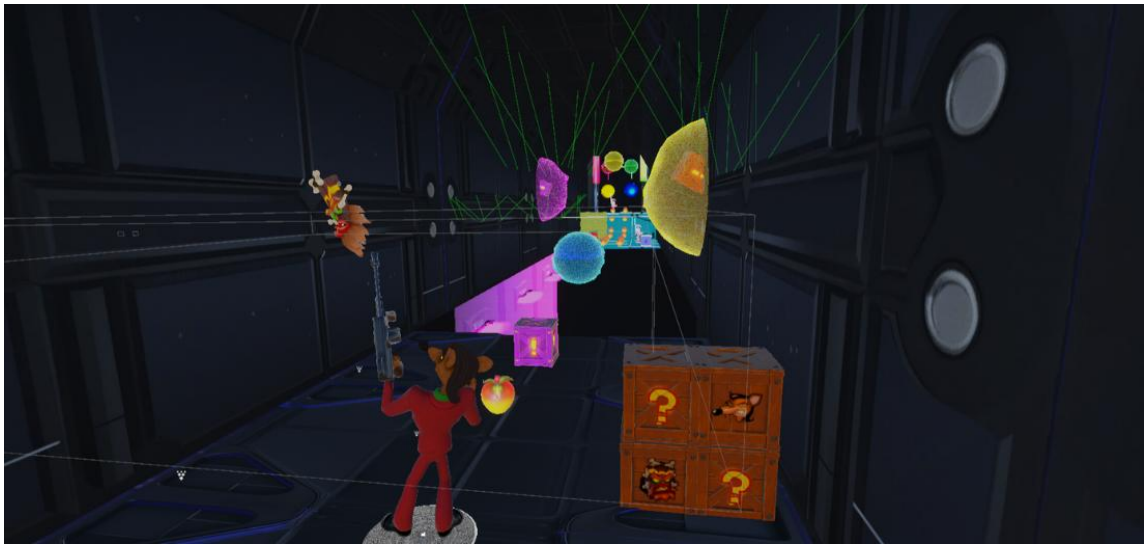


Figure 87: First Uka Uka Crate in Colour Agony and Pathway Following a Potential Invincibility Usage.

Table 7: First Decision Point Payoff Structure

Decision-Makers (Player Type)	Symbol for Short-Term Small Gain+, Small Loss-, Big Gain ++, Big loss --	Symbol for Long-Term Small Gain+, Small Loss-, Big Gain ++, Big loss --	Net Payoff
Abuser (early use)	++Immediate elimination of the two enemies ahead and TNT-Nitro crates. ++Faster time	--Doesn't have invincibility for Death Route which is in the section of the level	Medium Payoff (easy now, punished later)
Hoarder (late use)	++ Easy section compared to other sections in the game	+ Uka Uka is still charged and stored for later more difficult sections	Very High Payoff
Strategic Player: in this case the player chose to keep the Uka Uka for later usage.	Same gain as the hoarder	Same gain as the hoarder	Very High payoff (reward for smart play)

The goal is to place Uka Uka crates so the strategic player has a consistently high payoff, while Abuser and Hoarder never have high payoff across the whole level. Each must have at least one low payoff in a decision point of the level. Here, neither the hoarder nor the abuser hit low, but the strategic player has a high payoff. This is enough motive not to re-allocate this particular Uka Uka crate.

In the next decision point, the strategic player has a Medium-Low Payoff. This indicates the level has flawed resource allocation. To correct it, a re-allocation of Uka Uka crates is needed. By testing the game new locations for Uka Uka Crates are formed, creating new decision points, which need to be examined again using payoff structures.

Table 8: Second Decision Point Payoff Structure before Crate Re-allocation

Decision-Makers (Player Type)	Short-Term Gain/Loss	Long-Term Gain/Loss	Net Payoff
Abuser (early use)	- Within twenty seconds of invincibility there is a loss of at least one other Uka Uka crate ++Faster Time	-- Although the invincibility is triggered, ultimately it goes to waste since there is, two platforms ahead, another crate	Medium-Low Payoff
Hoarder (late use)	-- Enemies, difficult platforming and TNT as well as nitro crates	-- Losing at least one source of invincibility	Very Low Payoff
Strategic Player: in this case the player can choose to use Uka Uka	Same gain and loss as the Abuser	Same loss as the abuser	Medium-Low Payoff

Figure 88: Second, Third, Fourth and Fifth Uka Uka Crate in Colour Agony fit in a Screenshot before Crate Re-allocation.

A new Uka Uka crate was placed prior to reaching the second in-level Uka Uka crate as illustrated in *Figure 89*. The fourth Uka Uka crate was removed and replaced with a different resource crate, a Mojo crate as depicted in *Figure 90*. The first decision point needs to be re-examined before looking into the new second decision point. The new crate causes repercussions when using Uka Uka in the first decision point.

Table 9: First Decision Point Payoff Structure after Crate Re-allocation

Decision-Makers (Player Type)	Short-Term Gain/Loss	Long-Term Gain/Loss	Net Payoff
Abuser (early use)	++Immediate elimination of the two enemies ahead and TNT-Nitro crates. ++Faster time	--Doesn't have invincibility for Death Route which is right after in the level - In less than 20 seconds there is an additional Uka Uka Crate	Medium Payoff (easy now, punished later)
Hoarder (late use)	++ Easy section compared to other sections in the game	+ Even though in less than 20 seconds there is an additional Uka Uka Crate, Uka Uka is still charged and stored, and the distance is small enough between them	High Payoff
Strategic Player: in this case the player chose to keep the Uka Uka for later usage.	Same gain as the hoarder	Same gain as the hoarder	High payoff (reward for smart play) That is enough reason to keep this crate.

Table 10: Second Decision Point Payoff Structure after Crate Re-allocation

Decision-Makers (Player Type)	Short-Term Gain/Loss	Long-Term Gain/Loss	Net Payoff
Abuser (early use)	++Faster time -- Faster Movement of the player makes upcoming platforming sections harder	--Doesn't have invincibility for Death Route the entrance of which is next to this main path	Low Payoff
Hoarder (late use)	++ Makes the platforming easier	++ Can use Uka Uka for more difficult sections	High Payoff
Strategic Player: in this case the player chose to keep the Uka Uka for later usage.	Same gain as the hoarder	Same gain as the hoarder	High payoff (reward for smart play) That is enough reason to keep this crate.

Figure 89: Second Decision Point with New Uka Uka Crate Placement.

Table 11: Third Decision Point Payoff Structure after Crate Re-allocation

Decision-Makers (Player Type)	Short-Term Gain/Loss	Long-Term Gain/Loss	Net Payoff
Abuser (early use)	++Faster time ++It takes an average player more than twenty seconds to reach the next Uka Uka crate	- After eliminating the enemies and nearby crates, moving faster disrupts timing in the platforming section	High Payoff
Hoarder (late use)	-- Enemies, difficult platforming and TNT as well as nitro crates	--Since the next Uka Uka crate is visible from this distance, the player forfeits one instance of invincibility	Very Low Payoff
Strategic Player: in this case the player can choose to use Uka Uka	Same gains as the abuser	Same small loss as the abuser	High payoff (reward for smart play)

Figure 90: Third and Fourth and Fifth Uka Uka Crate in Colour Agony fit in a Screenshot after Crate Re-allocation.

By comparison, when revisiting the same decision point after the crate re-allocation, the strategic player has a high payoff. The new Uka Uka Crate placement is optimal, because not only is there a high payoff for proper strategy, but also resource abusers are punished in the second decision point and resource hoarders are punished in the third decision point.

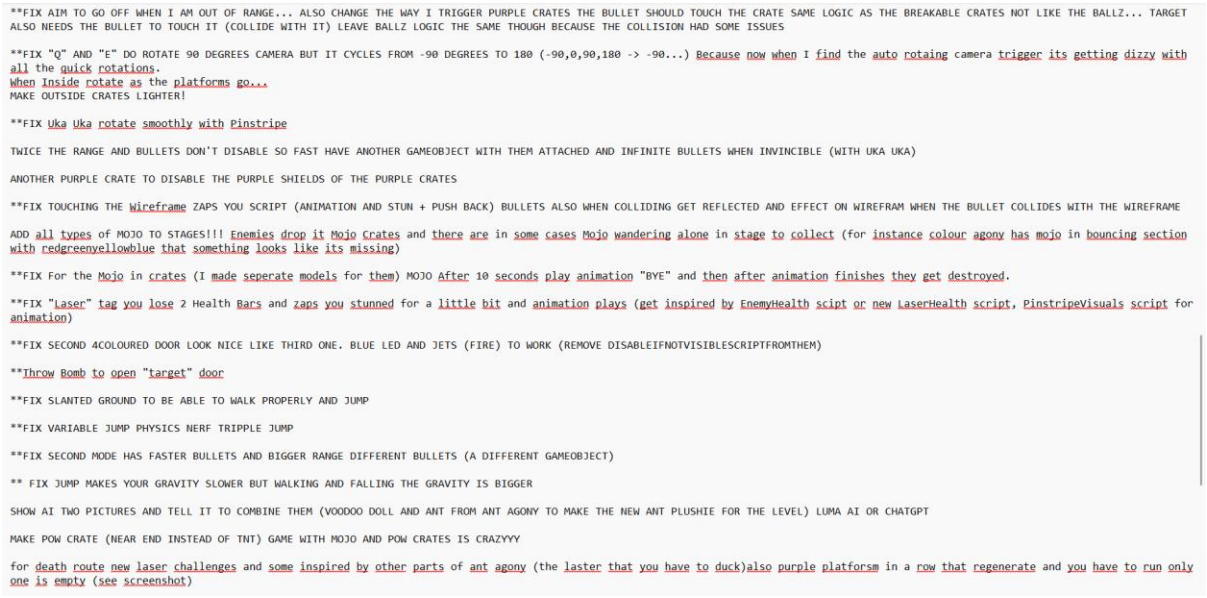
This is a clear-cut example of the game theoretical approach behind allocating resources throughout the level. Alongside the Uka Uka crates, this logic is later applied to all other resources enhancing user experience and making decisions more complex. Players ought to strategically think what the optimal decision is at each decision point and abusing or hoarding consumables is not always the dominant strategy. So, instead of designing a level blindly, modeling player decisions with game theory-inspired resource optimization, facilitates the developer by providing a structured way to predict and balance player's behavior when interacting with the game.

6. Conclusion Remarks and Limitations

The thesis achieved its main goal of proving that optimization and game theory can be integrated into video game design to elevate both player experience and developer decision-making. The project also highlighted the potential of AI-assisted development for independent developers. Despite the progress made, the project is subject to several limitations that constrain future work. There was no large-scale evaluation of the demo. Additionally, evaluation of the final game experience is currently impossible since core systems as the hub world, mechanics, levels and narrative progression remain incomplete. A major concern is, the workflow's dependence on third-party tools such as Blender, MeshyAI and ChatGPT. While effective, it reduces reproducibility and creates potential issues for long-term maintainability. Ultimately, the application of game theory models such as the payoff structures in Chapter 5, are not statistically validated through large-scale testing.

7. Future Work

Unfortunately, due to time constraints there are many bugs in the current version of the game that need to be resolved. Bugs are a defect or error in the game's code that causes it to behave in an unintended, incorrect or unexpected way. While testing the game and experimenting with different mechanics and collisions, execution did not always follow the plan. In *Figure 91* a portion of a list of identified problems is displayed.



**FIX AIM TO GO OFF WHEN I AM OUT OF RANGE... ALSO CHANGE THE WAY I TRIGGER PURPLE CRATES THE BULLET SHOULD TOUCH THE CRATE SAME LOGIC AS THE BREAKABLE CRATES NOT LIKE THE BALLZ... TARGET ALSO NEEDS THE BULLET TO TOUCH IT (COLLIDE WITH IT) LEAVE BALLZ LOGIC THE SAME THOUGH BECAUSE THE COLLISION HAD SOME ISSUES

**FIX "Q" AND "E" DO ROTATE 90 DEGREES CAMERA BUT IT CYCLES FROM -90 DEGREES TO 180 (-90,0,90,180 -> -90...) Because now when I find the auto rotaing camera trigger its getting dizzy with all the quick rotations.
When Inside rotate as the platforms go...
MAKE OUTSIDE CRATES LIGHTER!

**FIX Uka Uka rotate smoothly with Pinstripe

TWICE THE RANGE AND BULLETS DON'T DISABLE SO FAST HAVE ANOTHER GAMEOBJECT WITH THEM ATTACHED AND INFINITE BULLETS WHEN INVINCIBLE (WITH UKA UKA)

ANOTHER PURPLE CRATE TO DISABLE THE PURPLE SHIELDS OF THE PURPLE CRATES

**FIX TOUCHING THE Wireframe ZAPS YOU SCRIPT (ANIMATION AND STUN + PUSH BACK) BULLETS ALSO WHEN COLLIDING GET REFLECTED AND EFFECT ON WIREFRAM WHEN THE BULLET COLLIDES WITH THE WIREFRAME

ADD all types of MOJO TO STAGES!!! Enemies drop it Mojo Crates and there are in some cases Mojo wandering alone in stage to collect (for instance colour agony has mojo in bouncing section with redgreenyellowblue that something looks like its missing)

**FIX For the Mojo in crates (I made seperate models for them) MOJO After 10 seconds play animation "BYE" and then after animation finishes they get destroyed.

**FIX "Laser" tag you lose 2 Health Bars and zaps you stunned for a little bit and animation plays (get inspired by EnemyHealth scipt or new LaserHealth script, PinstripeVisuals script for animation)

**FIX SECOND 4COLOURED DOOR LOOK NICE LIKE THIRD ONE. BLUE LED AND JETS (FIRE) TO WORK (REMOVE DISABLEIFNOTVISIBLESCRIPTFROMTHEM)

**Throw Bomb to open "target" door

**FIX SLANTED GROUND TO BE ABLE TO WALK PROPERLY AND JUMP

**FIX VARIABLE JUMP PHYSICS NERF TRIPPLE JUMP

**FIX SECOND MODE HAS FASTER BULLETS AND BIGGER RANGE DIFFERENT BULLETS (A DIFFERENT GAMEOBJECT)

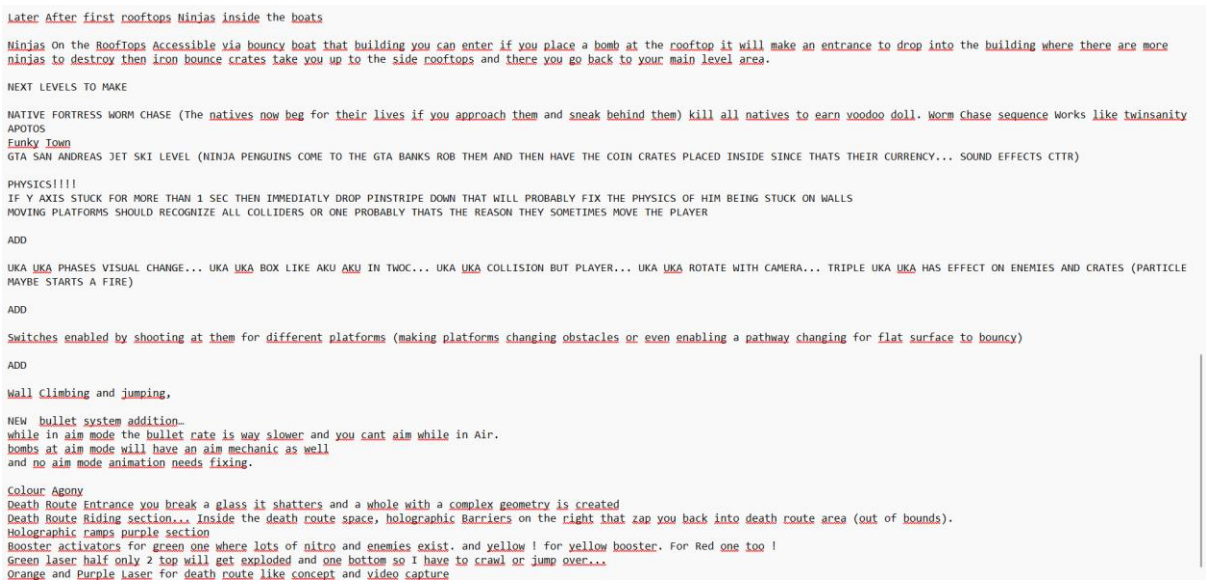
** FIX JUMP MAKES YOUR GRAVITY SLOWER BUT WALKING AND FALLING THE GRAVITY IS BIGGER

SHOW AI TWO PICTURES AND TELL IT TO COMBINE THEM (VOODOO DOLL AND ANT FROM ANT AGONY TO MAKE THE NEW ANT PLUSHIE FOR THE LEVEL) LUMA AI OR CHATGPT

MAKE POW CRATE (NEAR END INSTEAD OF TNT) GAME WITH MOJO AND POW CRATES IS CRAZYYY

for death route new laser challenges and some inspired by other parts of ant agony (the laster that you have to duck)also purple platforsm in a row that regenerate and you have to run only one is empty (see screenshot)

Figure 91: List of Identified Problems.



Later After first rooftops Ninjas inside the boats

Ninjas On the RoofTops Accessible via bouncy boat that building you can enter if you place a bomb at the rooftop it will make an entrance to drop into the building where there are more ninjas to destroy then iron bounce crates take you up to the side rooftops and there you go back to your main level area.

NEXT LEVELS TO MAKE

NATIVE FORTRESS WORM CHASE (The natives now beg for their lives if you approach them and sneak behind them) kill all natives to earn voodoo doll. Worm Chase sequence Works like twinsanity
APOTOS
Funky Town
GTA SAN ANDREAS JET SKI LEVEL (NINJA PENGUINS COME TO THE GTA BANKS ROB THEM AND THEN HAVE THE COIN CRATES PLACED INSIDE SINCE THATS THEIR CURRENCY... SOUND EFFECTS CTTR)

PHYSICS!!!!
IF Y AXIS STUCK FOR MORE THAN 1 SEC THEN IMMEDIATLY DROP PINSTRIPE DOWN THAT WILL PROBABLY FIX THE PHYSICS OF HIM BEING STUCK ON WALLS
MOVING PLATFORMS SHOULD RECOGNIZE ALL COLLIDERS OR ONE PROBABLY THATS THE REASON THEY SOMETIMES MOVE THE PLAYER

ADD

UKA UKA PHASES VISUAL CHANGE... UKA UKA BOX LIKE AKU AKU IN TWOC... UKA UKA COLLISION BUT PLAYER... UKA UKA ROTATE WITH CAMERA... TRIPLE UKA UKA HAS EFFECT ON ENEMIES AND CRATES (PARTICLE MAYBE STARTS A FIRE)

ADD

Switches enabled by shooting at them for different platforms (making platforms changing obstacles or even enabling a pathway changing for flat surface to bouncy)

ADD

Wall Climbing and jumping,

NEW bullet system addition…
while in aim mode the bullet rate is way slower and you cant aim while in Air.
bombs at aim mode will have an aim mechanic as well
and no aim mode animation needs fixing.

Colour Agony
Death Route Entrance you break a glass it shatters and a whole with a complex geometry is created
Death Route Riding section... Inside the death route space, holographic Barriers on the right that zap you back into death route area (out of bounds).
Holographic ramps purple section
Booster activators for green one where lots of nitro and enemies exist. and yellow ! for yellow booster. For Red one too !
Green laser half only 2 top will get exploded and one bottom so I have to crawl or jump over...
Orange and Purple Laser for death route like concept and video capture

Figure 92: List of Future Implementations.

There are many levels planned for the game. In *Figure 94* a list of them is presented. Some levels introduce new vehicles. Number “2.” and “27.” will include a flying vehicle. Number “4.”, “11.” and “34.” will include a jet-ski. Number “0.” will be the only level exclusively a vehicle level. Specifically, it is a kart-driving level acting as an introduction to the kart-driving mechanics. Number “9.”, “14.”, “18.” and “33.” combine kart driving sections with the platforming third person shooting sections. These hybrid levels provide variety to the game while maintaining balance and keeping the emphasis on the core gameplay mechanics. The ratio is ten out of the conceptual thirty-four levels. Level 26 is a minigame type level, something unexpected to maintain player engagement.

Level 17 is a unique type of level as well, it takes place inside a casino, where only basic movement and action input are essential for progress. The stage will be designed with the blend of a real-world casino feel and atmosphere and Crash-themed decorations. The level includes bright lights, slot machines, card tables, spinning wheels. Players will be able to convert Wumpa coins into casino credits. Fictionalized versions of real gambling activities such as Blackjack will be present having the Trophy Girls from Crash Team Racing as the dealers. Beating the AI in games of chance will reward you with different main collectibles. Game Theory concepts and optimization techniques are strongly tied with casino gambling activities because players and casinos make strategic decisions under uncertainty.

0. Racing To the MotorWorld
Boss Fight: Von Clutch and crew

Mystery Island
1. Avalanche (TWOC)
2. Jak 3 Flight Level (Pasadena+Pinstripe)
3. Pirate Ship Boom Bang
4. Don't be making waves (Tawna+Pinstripe) (Blue Gem)
5. Apotos (Obsidian Gem) Στην Μυλο ο πολυτιμότερος Λίθος

Boss Fight: In level Papu Papu in Apotos Temple
(you unlock time trial mode)

Happily ever Faster
6. Far Far away*
7. Gone a bit Loco
8. Hi-5 House
9. Sandy's Dream
10.Colour Agony (Orange Gem)

Boss Fight: In level Evil Twins?

Tyrannosaurus Wrecks
11. Watering Hole (mdgscr2) (decision making level) (Yellow
12. Night Worm Chase (including natives stealth section lev
night)
13. Bonsai Ruins (Tikki Bonsai)
14. Boiler Doom (crashtwinsanity Pasadena section dsi conce
15. 11th Dimension (boom bang desert) (Red Gem)

Boss Fight: In level Bird Mama Boom Bang

Tomb Town
16. Egyptian & Arab theme level mixed (Crash 3 Warped) πολλες
17. Casino Level (decision making level)
18. Rockslide Rumble (through the realms like old 7 year old a
19. Ants To Ashes (Crash Twinsanity inspired Lava Caves)
20. Summoners Rift (Green Gem)

Boss Fight: In level Urf & Rusty Walrus

Astro Land
21. Nina's Public School
22. Hang Castle Cortex's lair (Sonic heroes and Crash Of The Titans DS)
23. Academy Of Evil
24. Funky Town (Purple Gem)

Boss Fight: In level N-tropy with 2 T.K's Chase section
25. Nightmare madness (Last Hidden Crash Bandicoot Doll)

Final Boss Fight: Dr. Neo Cortex (svr arena match)

Extra Warp Room (Unlocked by completing Relic Challenges):

26. Zoomba (Bust a Groove) Challenge for a Crystal (5 relics)
27. Weathering Heights (+ Airship cut content from Crash TWOC). (10 relics)
28. Restless Coastside (Sonic Unleashed (Wii) - Adabat Night Stage 2) (15 relics)
29. Minority Tunnels (Minority Rapport cott+ Toxic Tunnels cb4) (20 relics)
30. Mar's Tomb (Jak II Level) instead of monkey bars use (precursor boosters)) (25

Extra Boss Fight (Jak II Mayor)

Final Stage?
31. Coco's TimeTwister (crash boss fight collect all Crash Voodoo Dolls)
True Final Boss Fight: All the Doctors
32. VOODOO MIA (Unlocked by collecting all Golden Voodoo Dolls)

levels I don't know where:
33. Compactor Reactor with ctr like jak 3 cars section at first
34. Crash N Burn with Jet Ski section at first.

Figure 93: List of planned Future Levels.

New mechanics related to platforming will be introduced at later stages. Such abilities are wall climbing, rope swinging and monkey bar swinging. Furthermore, new type of enemies with different AI behaviours will be incorporated as well. For example, the Ninjas in Bonsai Ruins will be able to teleport when they detect a threat.

Other types of enemies that will be part of every level are the doll thieves. Inspired by Spyro the Dragon's egg thieves, they are enemies who will strategically respond to the player's actions. In a game-theoretic frame, Player A is the player and Player B is the thief. The player chooses whether to move, shoot, or throw a bomb. At the same time, the thief chooses whether to remain idle or start running based on detection. The player's payoff is catching the thief and retrieving the doll. The thief's payoff is successfully escaping. It could be represented within the framework of a pursuit-evasion game aligning closely with the behavior of a Hunter-Rabbit game. A hunter attempts to capture a rabbit moving on a graph, while the rabbit tries to avoid capture [16].

Time Trial Mode is a major addition to the game. It is a mode conceptualized and slightly referenced in this project but currently entirely missing. Very reminiscent of the time trial mode in classic Crash Bandicoot games, this iteration will include all time-freezing crates. As illustrated in *Figure 94* sprites of the numbers have already been created. The mechanics of Pinstripe Potoroo: The Dark Gem Conspiracy will give this mode a different nuance. Potentially, a significant enhancement to the mode is AI ghost racers. Rather than static replays, AI ghost racers will implement A*/NavMesh algorithms combined with path and speed optimization [17].

Figure 94: Time Crate Number Sprites.

Figure 95: Rough Sketch of Initial AI Ghost Implementation Concept (from the Bonus Logistics 9th-Semester Presentation).

References

[1] Granic, I., Lobel, A., & Engels, R. C. (2014). The benefits of playing video games. *American Psychologist, 69*(1), 66–78. https://doi.org/10.1037/a0034857

[2] Egenfeldt-Nielsen, S. (2006). Overview of research on the educational use of video games. *Nordic Journal of Digital Literacy, 1*(3), 184–214.

[3] Ashinoff, B. K. (2014). The potential of video games as a pedagogical tool. *Frontiers in Psychology, 5,* 1109.https://doi.org/10.3389/fpsyg.2014.01109

[4] Cole, T., Alhadeff, J., Brown, C., & Smith, M. (2024). Video games in secondary classrooms: A scoping review. *Journal of Research on Technology in Education, 56*(3), 311–329.

[5] Monley, C. M., Ferguson, C. J., & Colwell, J. (2023). Gamers' and non-gamers' perspectives on the benefits of video games. *JMIR Serious Games, 11,* e43304. https://doi.org/10.2196/43304

[6] Chaarani, B., Agrawal, A., & Garavan, H. P. (2022). Association of video gaming with cognitive performance among children. *JAMA Network Open, 5*(10), e2235725.

[7] Qaffas, A. (2020). An operational study of video game genres. *International Journal of Computer Games Technology, 2020,* 8827510.

[8] Chin, J. P. (2012). Third-person shooter/adventure game framework. In C. Crawford (Ed.), *Game development essentials: Game interface design* (pp. 197–220). Cengage Learning.

[9] Abstracting Games. (2023, January 1). Collectathon-athon. *Abstracting Games*. https://abstractinggames.com/2023/01/01/collectathon-athon/

[10] Osborne, M. J. (2004). *An introduction to game theory.* Oxford University Press.

[11] Gregory, J. (2018). *Game engine architecture* (3rd ed.). CRC Press.

[12] Öztürk, S., Koc, B., & Aydın, E. (2025). Analyzing maintainability factors in open-source game engines. *Journal of Systems and Software, 210,* 111041.

[13] Theseus. (2024/2025). *Platformer game: Full Unity platformer prototype; covers pipeline, mechanics, UI.* Theseus.fi.

[14] Roungas, B. (2019/2021). *The game between game theory and gaming simulations: Design choices.* Delft University of Technology.

[15] Guardiola, E., & Natkin, S. (2005). *Game theory and video game: A new approach to analyze and conceive game systems*. https://www.researchgate.net/publication/221249068

[16] Karlin, A., & Peres, Y. (2017). *Game theory, alive.* American Mathematical Society. https://doi.org/10.1090/mbk/111

[17] Theseus. (2025). *AI development for racing games: Unity racing AI with NavMesh; path/speed optimization on tracks.* Theseus.fi.

Appendix A – Unity Terminology

GameObject → the fundamental object in Unity that represents characters, props and scenery. It does not accomplish much in itself, but it acts as a container for Components, which implement the real functionality. For example, a Light object is created by attaching a Light component to a GameObject.

Prefab → A prefab, in its simplest form, is a template from which you can create multiple instances of the same object, keeping all its properties. And when you need to make a change to an instance, you simply update the prefab, and all its children will be updated. For example, if you make an "Enemy" GameObject and save it as a Prefab, you can drag as many copies into your scene as you want. If you change the Prefab, all copies update too.

Spritesheet → A spritesheet is basically just an image that contains images meant to be used separately.

Sprites → it refers to a small, often transparent, 2D graphical element that can be animated and integrated into a larger scene or background in computer graphics, especially video games.

Inspector → Games in Unity are made up of multiple GameObjects that contain meshes, scripts, sounds, or other graphical elements like Lights. The Inspector displays detailed information about your currently selected GameObject, including all attached Components and their properties.

Rigged → Rigging in game development is the creation of a digital "skeleton" or rig within a 3D character model, which then acts as a control system to allow animators to move and pose the character. This skeletal structure consists of bones, joints, and controllers that dictate how the character's mesh deforms and moves, bringing it to life for gameplay and enhancing realism.

Mesh → A mesh is a set of polygons that create a 3D object. When designing a 3D model, the result is a hard-to-render point layer. Therefore, developers transform these points into a set of polygons.

Parent GameObject → It is an object that has one or more child GameObjects associated with it, forming a hierarchy.

Child GameObject → It is an object that is nested under another GameObject, its parent.

Components → They are functional pieces of every Gameobject.

Build settings → They are configurations that determine how software, games, or applications are compiled and packaged for a specific platform or environment.