THE EXPLORER – A 3D ADVENTURE GAME USING UNITY

A Main Project Report

submitted by

AMAN ABDUL MALIK K P (MES20MCA-2004)

to

the APJ Abdul Kalam Technological University
in partial fulfillment of the requirements for the award of the Degree

of

Master of Computer Applications

Department of Computer Applications

MES College of Engineering
Kuttippuram, Malappuram - 679 582

July 2022

DECLARATION

I undersigned hereby declare that the project report **THE EXPLORER – A 3D ADVENTURE GAME USING UNITY**, submitted for partial fulfillment of the requirements for the award of degree of Master of Computer Applications of the APJ Abdul Kalam Technological University, Kerala, is a bona fide work done by me under supervision of Syed Feroze Ahamed M, Assistant Professor, Department of Computer Applications. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place: Kuttippuram

AMAN ABDUL MALIK K P (MES20MCA-2004)

Date: 06/07/2022

DEPARTMENT OF COMPUTER APPLICATIONS
MES COLLEGE OF ENGINEERING, KUTTIPPURAM

CERTIFICATE

This is to certify that the report entitled **THE EXPLORER – A 3D ADVENTURE GAME USING UNITY** is a bona fide record of the Mini Project work carried out by **AMAN ABDUL MALIK K P (MES20MCA-2004)** submitted to the APJ Abdul Kalam Technological University, in partial fulfillment of the requirements for the award of the Master of Computer Applications, under my guidance and supervision. This report in any form has not been submitted to any other University or Institution for any purpose.

Internal Supervisor(s)

External Supervisor(s)

Head Of The Department

Acknowledgements

First of all, I would like to give thanks to God for giving me the opportunity to complete and submit this project. Without the help and blessings of God the Almighty, it would not have been possible to accomplish this project.

With great respect I express my sincere thanks to Dr.Rahumathunza I, Principal, MES College of Engineering, providing facilities for this project. I would also like to express my gratitude towards my supportive and encouraging project coordinator, Ms.Priya J.D Assistant professor, Department of Master of Computer Applications, for allowing me to do this project and for her inspiring guidance, reliability, constructive criticism and challenging but motivational feedback throughout the course of this project.

I would like to express my sincere thanks to Mr.Hyderali K, Associate professor, Head of the Department and Mr.Syed Feroze Ahamed M, my internal guide and all of the faculty members of the Department of Master of Computer Applications for equipping me with the skills and knowledge of our field of education required to accomplish this project. Your contribution will doubtlessly be acknowledged and I will always remain grateful to you all. Last but not least my graceful thanks to my parents, friends and also the persons who supported me directly and indirectly during the project.

AMAN ABDUL MALIK K P (MES20MCA-2004)

Abstract

This report focuses on the development of a 3D TPP (Third Person Perspective) action adventure game, The Explorer. The project is based on classic action adventure games. This game contains two challenging levels for the the player to play. It has a powerful short ranged melee weapon, computer controlled intelligent enemies. The game have situational problems for the player to solve, with very little or no action to give the player a more interesting game play and other interesting features like loot items and puzzles. And as a 3D adventure game the game is aesthetically very pleasing and provide a beautiful open world for the player to explore.

This project explores a new dimension to the traditional action-adventure games by mixing the features of open world action games where player destroy enemies to survive and can navigate throughout the game world. With the simplicity of this game it simply aims to bring fun and make you look back to your childhood's classic games like The Legend of Zelda, Tomb Raider and Syberia with some new modern features.

Keywords: 3D, TPP (Third Person Perspective), action-adventure game, Desktop Game

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

The gaming industry has had much growth in recent years. It is a great industry to get involved in as it allows creativity, innovation and freedom for developers and hobbyists. They get a chance to experiment with all forms of media including sound design, environment design and programming. As there has been an explosion of the new ‘Indie’ market of games, the gaming industry now allows smaller-scale games to be developed and released more freely than in the past. Today it has become so easy to create a game. A large studio is no longer needed and the freely available tools make it so simple to create an idea from the comfort of your own home. Indie games show more innovation and developers are willing to take risks on their games. ‘Indie’ games also have a large market base with many of the games being released on PC, Android etc.

There is a massive popularity for not just adventure games, but particularly innovative survival action-adventure games. I put a lot of research into trying to find out what users want from a game. I had to decide between a first person or a third person game, and what type of environment would be best to build my game. I finally decided to go with a third person perspective in a 3D hand-painted style environment.This style is characterized by freehand texture painting. All shadows, highlights, and sometimes small details are drawn on the texture without geometry.This significance of style takes the game to a completely different scale. I decided to go with a 3D game over a 2D game as I think it makes the game that extra bit challenging and interesting.

1.1.1 Motivation

In the fast growing field of software engineering and development and even more rapidly growing sector of game development the future is hard to predict. In general software project is a project focusing on the creation of software. Consequently, Success can be measured by taking a look at the resulting software. In a game project, the product is a game. But and here comes the point: A game is much more than just its a software. It has to provide content to become enjoyable. Just like a web server: without content the server is useless, and the quality cannot be measured. This has an important effect on the game project as a whole. The software part of the project is not the only one, and it must be considered in connection to all other parts: The environment of the game, the story, characters, gameplays, the artwork, and so on.

Video games are not just any computer software which are made to benefit user's daily life, games are rather made for user's entertainment purpose, so more than anything we need to pay attention to what the user wants from the game, how to make it more entertaining, just making any game will not do, that is why it's more challenging because I always have to carefully consider if I'm developing it correctly to entertain users. I also have to invest a lot of time on the proper game designing to make it visually accepted. And to add that a game requires a lot of scripts. The scripts are like pieces of a puzzle which you need to put all of them together to make it work.

Despite the economic instability and crisis deeply affecting the world, the analysts published that the game industry has grown at a rate of 57% surprisingly. Even as I type these words millions of people are playing games in front of their computers.The reason of this growth can be stated that the game industry can appeal any user with different tastes. And there's no wonder that the gaming industry became the number one entertainment industry in the world. Thus I think a video game is a perfect project to prove myself as a computer application student and this will be a stepping stone to the world's number one entertainment industry.

1.2 Objective

The aim of this project was to create a fully functional 3D survival action-adventure game in third person perspective. The overall aim of my game was to create a tense and unnerving experience for the player. I chose a sci-fi environment setup as atmosphere is essential for the game. As I have path finding on my enemies, they will not attack until the player is in a certain range and field of vision. They might not always be visible to the player as the scene has many hiding places. This adds a level of fear in the game as you will not know how close the next enemy is or when they will attack. I also included audio and sound effects to support the atmosphere of the game. The purpose of this project is to develop a third person perspective 3D game with good graphics, audio and animations made with basic technologies and a low budget. I wanted to make a game that was easy to install and show what can be done with just a bit of time and effort.

I chose the Unity 3D game engine to develop my game as it has a large community which provides learning support from the Unity website and forums. There are many tutorials also available on YouTube. You can code your game in either C# or JavaScript which made it ideal, as both languages is very common and lot of learning materials can be found Online or libraries. The game will run on a Windows PC. This gives me the option of publishing my game to the popular PC online gaming market Steam. This would make my game available to many users online.

The game engine used is Unity3D. The game scripts is written mainly in C#. Visual Studio 2019, the main IDE which is used with Unity was used to edit scripts. Assets in my game were sourced from the Unity asset store.

1.3 Report Organization

This project report is divided into four sections. Section 2 describes literature survey. Section 3 describes the methodology used for implementing the project. Section 4 gives the results and discussions. Finally Section 5 gives the conclusion.

Chapter 2

Literature Survey

2.1 Unity Game Development Engine

2.1.1 Unity Game engine

In this section of the proposed paper, the literature survey has been conducted. The documentation of relevant works and examination of the collected sources fundamentally is demonstrated here. Therefore, the literature survey has been considered as one of the vital aspects of this report as it supports authors as well the readers in understanding and examining several facets that the relevant to the current topic and are conducted previously. Unity, a multi-platform game engine, commercially available and is used for 2d and 3D video games production accompanied by visualizations and non-game interactive simulations. Moreover, Unity is one of the popular engines for games that is easily accessible, and in the current epoch is communal amid the developers due to its ease, flexibility, efficiency, and power consumption. Multiple tools have been featured in the Unity Editor that permits rapid iteration and editing in the cycles of development comprised of smart previews play mode in real-time [1]. Furthermore, Unity is offered on Mac, Linux, and Windows it contains an artist friendly range of tools for immersive designing and game worlds, in addition to a strong developer tools suite for implementing high-performance gameplay and game logic [2]. Moreover, Unity supports 3D and 2D development with functionalities and features for the specific needs that are further focused on categories.

Similarly, [3] has quoted that the navigation system has also been included in Unity that permits to create NPCs that logically move around the world of gaming. Navigation meshes have been used in the system that is automatically created from the scene geometry, or sometimes even from the dynamic obstacles, to change the character's navigation at runtime. Unity Prefabs that are known as the pre-configured objects of gaming provide the workflow flexibility and efficiency that permit in the confident working without nerve-wracking about the time-consuming errors [4]. According to [5] it has been found that Unity built-in UI system permits to create the UI intuitively and smartly with less consumption of time. Unity takes Box2D advantage the novel DOTS-based NVIDIA PhysX and Physics system that aids in the provision of high-performance and high realistic gameplay. Developers can extend the Editor of Unity with tools need to match the workflow of teams. It also supports the creation and customizing of the extensions that feature many possessions, extensions, and tools for the sake of speediness of the projects.

In addition, some of the key features of unity are related to the simple workflow that allows developers to quickly combine scenes in a workspace referred to as (intuitive editor). It also supports creating high quality games like AAA images, high definition sound, and action at full speed with no gaps on the screen [6]. Also, some extra features are [7]access the components, co-routine and return types, creating and destroying GameObjects, dealing with vector variables and timing variables, events for GameObject, and physics-oriented events. Primarily, Unity supports scripting in C, and there are two ways to design C scripts in Unity: object-oriented design, which is the more traditional and widely used approach, and data-oriented design, which is now possible in Unity [8].

Chapter 3

Methodology

Unity is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Inc.'s Worldwide Developers Conference as a Mac OS X exclusive game engine. The engine has since been gradually extended to support a variety of desktop, mobile, console and virtual reality platforms. It is particularly popular for iOS and Android mobile game development and used for games such as Pokemon Go, Monument Valley, Call of Duty: Mobile, Beat Saber and Cup-head. The engine can be used to create three-dimensional (3D) and two-dimensional (2D) games, as well as interactive simulations and other experiences.The engine has been adopted by industries outside video gaming, such as film, automotive, architecture, engineering, construction, and the United States Armed Forces.This project is entirely built on Unity engine.

3.1 Introduction

The Explorer is a 3D Third-Person survival action adventure game developed using Unity Game Engine. An action-adventure game can be defined as a game with a mix of elements from an action game and an adventure game, especially crucial elements like puzzles. I've created a playable 2 level game using this system for the player to explore. Discover the mysterious alien planet where our Captain, Ellen has crash landed. Avoid the hazards and defeat the enemies lurking within the ancient ruins of this unknown civilization.

This video game is entirely built using Unity game engine. Unity is a cross-platform game engine developed by Unity Technologies. The engine can be used to create three-dimensional

(3D) and two- dimensional (2D) games, as well as interactive simulations and other experiences. The engine has been adopted by industries outside video gaming, such as film, automotive, architecture, engineering, construction, and the United States Armed Forces. The explorer is a 3D Third-Person perspective game, in a third-person game, the player can see the character they are controlling (usually from behind, or above). They differ from First-person video games where the players are centered on guns and other weapon-based combat in a first-person perspective, with the player experiencing the action through the eyes of the protagonist and controlling the player character in a three-dimensional space.

The primary design focus is exploration and combat, mainly involving melee weapons. The engine offers a primary scripting API in C#, for both the Unity editor in the form of plug-ins, and games themselves, as well as drag and drop functionality. For 3D games, Unity allows specification of texture compression, mipmaps, and resolution settings for each platform that the game engine supports, and provides support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects. The level is designed by using ProGrids in unity. ProGrids is an essential tool that is used to help place objects with ease and precision. We can add game objectives in unity easily. The objective system works simply by adding GameObjects with an Objective component on them coupled with a specific components like “Interact on Trigger” or “Damageable”. The game will have moving platforms and player teleportation facilities which will make the gameplay more fun and also needs lot of game mechanisms to work with. The game has different types of enemies, each with different characteristics which the user can interact. The level is designed by using ProGrids in unity. ProGrids is an essential tool that is used to help place objects with ease and precision. We can add game objectives in unity easily.

The player assumes the role of a protagonist in an interactive story or objective driven by exploration and/or puzzle-solving. The story is heavily reliant upon the player character’s movement, which triggers story events and thus affects the flow of the game. The Explorer is a massive 3D open world game where the players can enjoy the gameplay by exploration and puzzle solving and there are so many elements that will help the players to be immersed in the experience.

3.2 Action-Adventure Games

Action-adventure is a hybrid genre and can include many games which might better be categorized under more narrow genres. Typically, pure adventure games have situational problems for the player to solve, with very little or no action. If there is action, it is generally confined to isolated minigames. Pure action games have gameplay based on real-time interactions that challenge the reflexes. Therefore, action-adventure games engage both reflexes and problem-solving in both violent and non-violent situations. An action adventure game can be defined as a game with a mix of elements from an action game and an adventure game, especially crucial elements like puzzles.Action-adventures require many of the same physical skills as action games, but also offer a storyline, numerous characters, an inventory system, dialogue, and other features of adventure games. They are faster-paced than pure adventure games, because they include both physical and conceptual challenges. Action-adventure games normally include a combination of complex story elements, which are displayed for players using audio and video. The story is heavily reliant upon the player character's movement, which triggers story events and thus affects the flow of the game. Some examples of action-adventure games include The Legend of Zelda, God of War and Tomb Raider series.

Action-adventure games are faster paced than pure adventure games, and include physical as well as conceptual challenges where the story is enacted rather than narrated. While motion-based, often reflexive, actions are required, the gameplay still follows a number of adventure game genre tropes (gathering items, exploration of and interaction with one's environment, often including an overworld connecting areas of importance, and puzzle-solving). While the controls are arcade-style (character movement, few action commands) there is an ultimate goal beyond a high score. In most action-adventure games, the player controls a single avatar as the protagonist. This type of game is often quite similar to role-playing video games.

3.3 Third-Person Perspective

This game is developed in Third Person Perspective(TPP). Also known as the "over-the-shoulder" view, the third-person perspective is another popular choice among game designers these days, especially because most games are in 3D. A third-person perspective is when the

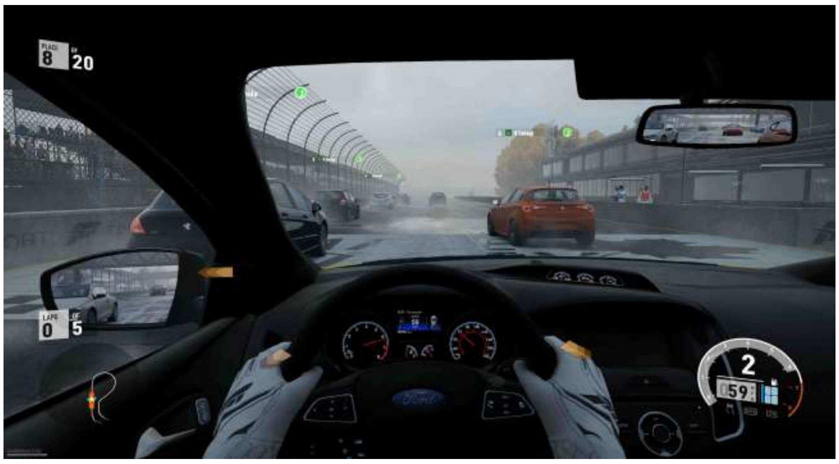


Figure 3.1: First Person Perspective

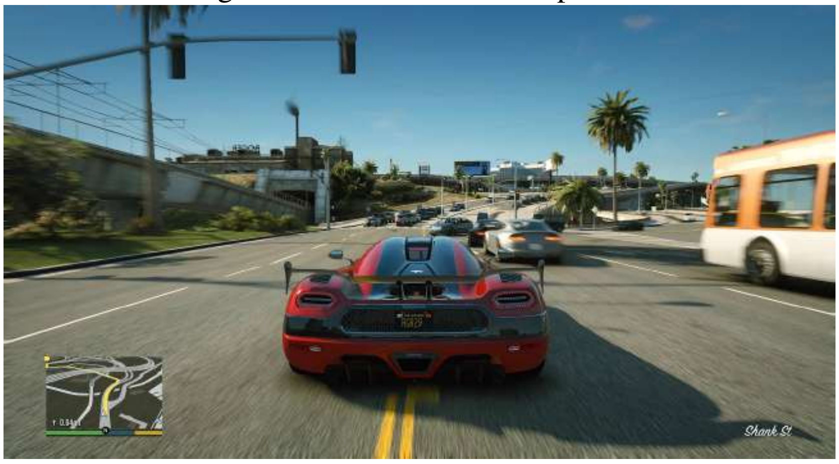


Figure 3.2: Third Person Perspective

player can see the character he or she is playing onscreen. As opposed to first-person perspective video games, you do not see the character's personal point of view (Figure 3.1), but instead you see a theoretical "third person's" view point "watching" the character (Figure 3.2). The third-person perspective offer the benefits of allowing the player to get a wider view of their surroundings, allowing an easier approach to multi-directional movement, making third-person the natural choice for 3D platformers like Super Mario Galaxy.

Eidos Interactive's Tomb Raider was one of the first big hits to use a third-person perspective with an "intelligent" camera that followed heroine Lara Croft around. Some games—such as Microsoft's Halo can switch between the first-person and third-person perspective. While on foot, the game is played from a first-person perspective. Inside one of the many vehicles, it swaps to a third-person view so players can see more of the environment.

3.4 Modules

3.4.1 Module - 1 : Prototype Design

A sample scene with templates describing the main theme or story of the game. The scene will contain:

- Main character (Hero)
- Main weapon
- World or terrain
- Enemies
- Other game objects

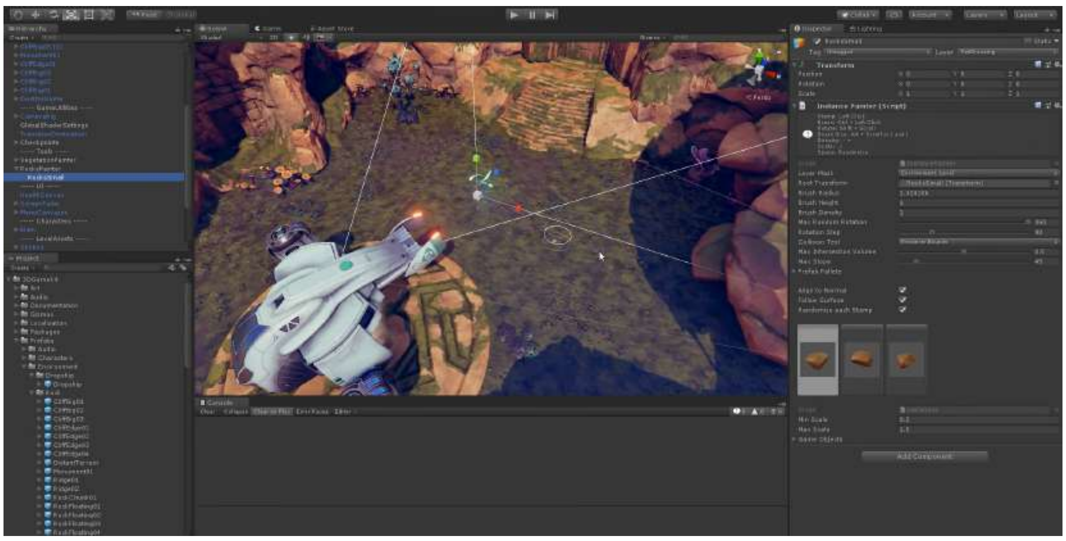


Figure 3.3: Prototype designing.

3.4.2 Module - 2: Design

Game Design

Like most video games, an action adventure game involves an avatar, one or more types of weapons, and a varying number of enemies. Because they take place in a 3D environment, these games tend to be somewhat more realistic than 2D adventure games, and have more

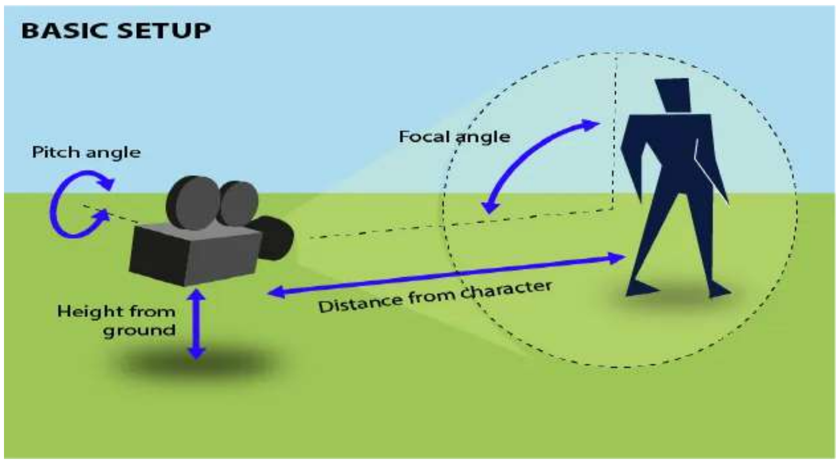


Figure 3.4: Basic TPP camera setup

accurate representations of gravity, lighting, sound and collisions. In Third-person perspective video games the camera view is set outside of your characters body (Figure 3.4). Third person view games played on personal computers are most often controlled with a combination of a keyboard and mouse. This system has been claimed as superior to that found in console games, which frequently use two analog sticks: one used for running and sidestepping, the other for looking and aiming. The camera follows behind the player character, allowing you to see more of what's around them than a first-person game does.Because third-person views let you see your character all the time, these games can more easily show their personality. Seeing the way that a character walks, interacts with the environment, reacts to events, and even what they wear can help you learn more about them through visual clues. With a UI display showing health, ammunition and location details. Often, it is possible to overlay a map of the surrounding area. The third-person view allows the relationship between the main character and the environment to be better demonstrated.

The third person view is done using the concept of interactive cameras. Interactive cameras are by far the most common third-person camera type today, interactive cameras follow behind your character but allow you to control them. Usually, with a controller, you use an analog stick to do this, while on PC it's controlled by the mouse. Interactive cameras are the most common because they provide the most player control. However, they aren't perfect. A poorly-made camera system can get stuck on objects in the world or otherwise make it difficult for you see what you're trying to view (Figure 3.5).

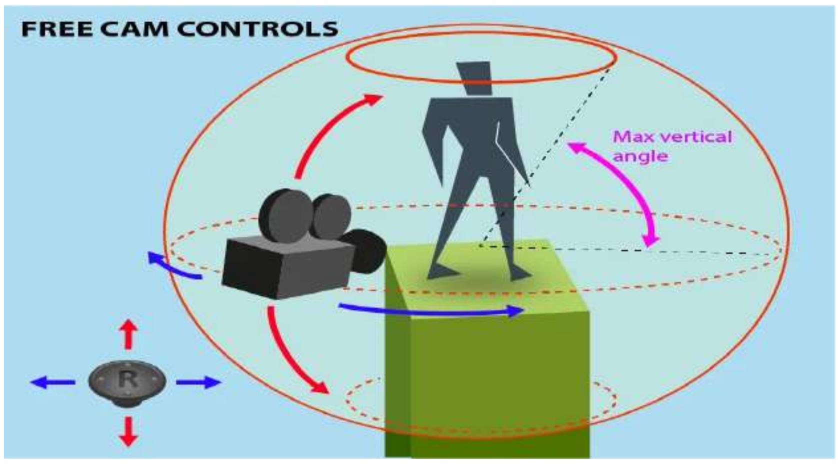


Figure 3.5: Using an X/Y input (more often a mouse or a pad stick), the player can move the camera freely on a sphere around the character, always looking at him. This move use a specific acceleration/max speed couple. The camera move is vertically limited by a limited angle.

Level Design

Action-adventure games can be structurally composed of levels, or use the technique of a continuous narrative in which the game never leaves the third person perspective. Others feature large sandbox environments, which are not divided into levels and can be explored freely. In action-adventure games, protagonists interact with the environment to varying degrees, from basics such as using doors, to problem solving puzzles based on a variety of interactive objects. In some games, the player can damage the environment, also to varying degrees: one common use is the usage of melee weapons to destroy game objects like boxes and rocks, sometimes explosive barrels too, destroying them can harm nearby enemies and the player too if they are too close. Other games feature environments which are extensively destructible, allowing for additional visual effects. The game world will often make use of science fiction, historic (particularly World War II) or modern military themes, with such antagonists as aliens, monsters, terrorists and soldiers of various types. Games feature multiple difficulty settings; in harder modes,the puzzles are harder to solve, enemies are tougher, more aggressive and do more damage, and power-ups are limited. In easier modes, the player can succeed through reaction times alone; on more difficult settings, it is often necessary to memorize the levels through trial and error.

The Explorer have two well designed levels. First level which is the entry point to the game is set in an outdoor environment (Figure 3.6). This gives the player the experience of a console level open world experience. The second level is set in an indoor environment where the gameplay is a little more tricky and challenging (Figure 3.7). The two levels are connected using scene controls and in the game, using a teleportation gateway. The levels are designed with help of ProBuilder and Polybrush.

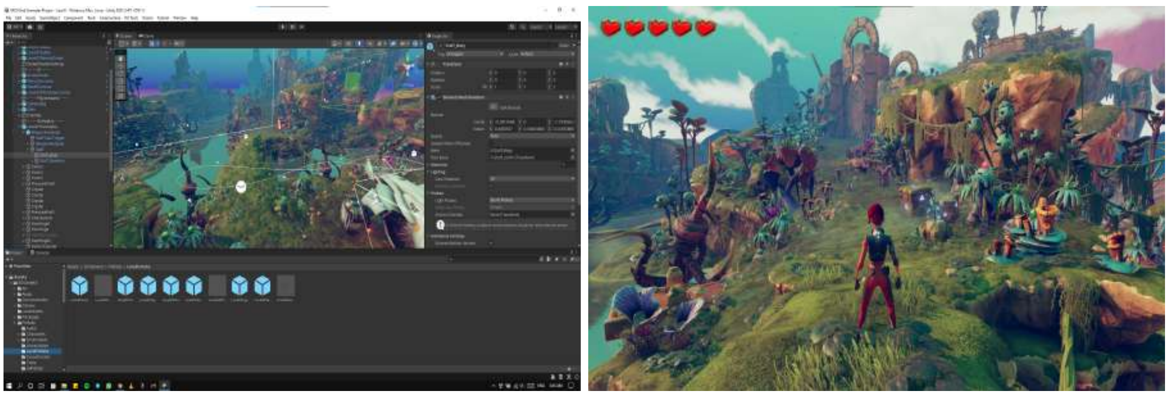


Figure 3.6: Level 1

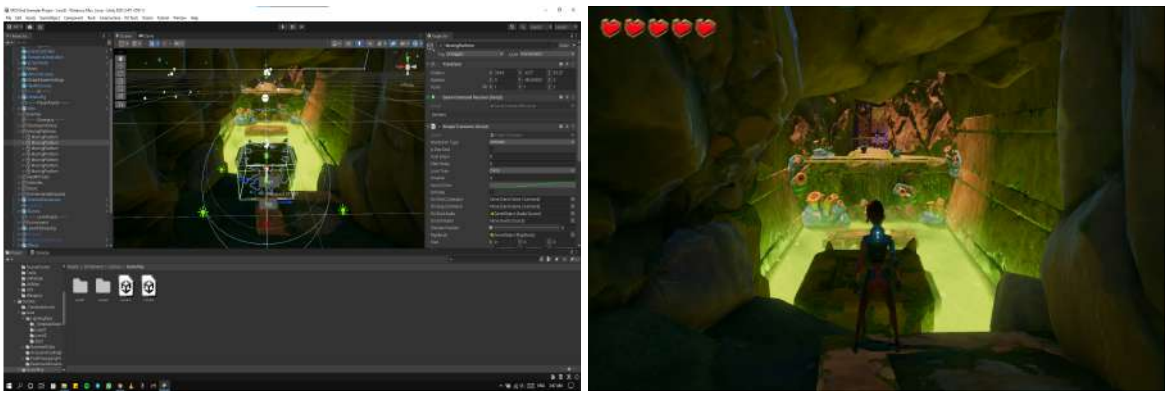


Figure 3.7: Level 2

Combat and power-ups

Action adventure games often focus on narrative and adventurous game play, with problem-solving and logic puzzles, though some place a greater emphasis on fast-paced and bloody firefights. Commonly combat and weapons are extensively used in action adventure games.In addition to that in some games long ranged weapon are also used, and u can also throw the melee weapons at your enemies. Melee weapons are especially powerful and deals a great amount of damage, a reward for the risk the player must take in maneuvering his character into close proximity to the enemy. In other situations, a melee weapon may be less effective, but necessary as a last resort.

Unlike first-person shooters action- adventure games typically doesn't give players a wide choice of weapons. The gameplay is mainly based on problem solving, puzzles, and exploration, so the player need to use his intelligence. Meaning the gameplay depends on the players thinking capabilities rather than depending on weapons. But some game designs have realistic models of actual existing or historical weapons. Other action-adventure games may incorporate imaginative variations of weapons, including future prototypes, "alien technology" scenario defined weaponry, and/or utilizing a wide array of projectiles, from industrial labor tools to laser, energy, plasma, rocket and grenade launchers or crossbows. These many variations may also be applied to the tossing animations of grenades, rocks, spears and the like. Also, more unconventional modes of destruction may be employed from the viewable user's hands such as flames, electricity, telekinesis or other supernatural constructions. There are often options to trade up, upgrade or swap out in most games. Thus, the standards of realism vary between design elements.

In The Explorer the player can obtain a very powerful melee weapon which can be used to destroy common enemies as well as used in boss fights. In video games, a boss is a significant computer-controlled opponent. A fight with a boss character is commonly referred to as a boss battle or boss fight. Bosses are generally far stronger than other opponents the player has faced up to that point and winning requires a greater knowledge of the game's mechanics. The melee weapon is a staff (a weapon that look likes a stick) designed to give the impression of an alien weapon. The weapon is obtained by Ellen after a short period when she explore the game environment. The staff will be placed on a weapon pedestal. Killing enemy bosses will be harder compared to the common enemies, but this will make the gameplay more engaging and challenging. The player will have to use different tactics and techniques to defeat the enemy bosses. The staff has beautiful animations and gives stunning visual effects. Once obtained the staff will be only seen when we use it (Figure 3.8). After we kill the enemy characters their dead body will start to disintegrate (Figure 3.9). The protagonist can generally be healed and re-armed by means of items such as first aid kits, simply by walking over them. Some games allow players to accumulate experience points similar to those found in role-playing games, which can unlock new weapons and abilities. In here Ellen can easily heal herself using health

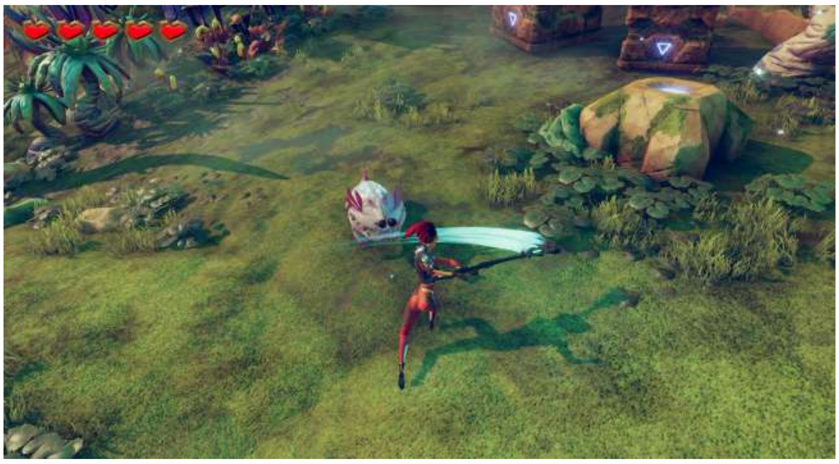


Figure 3.8: Ellen attacking an enemy character.

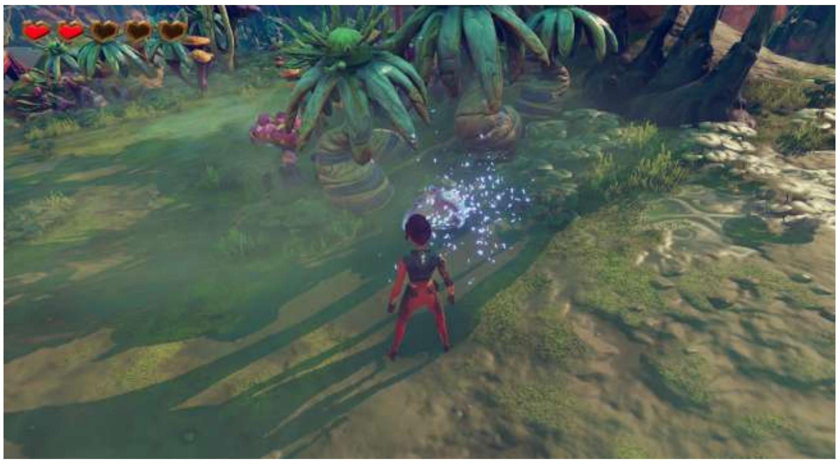


Figure 3.9: Enemy disintegration.

boxes. The player only need to touch it or walk to it (Figure 3.10).

Other design aspects:

- Character design and customization
- Weapon design and modifications

Puzzles

Puzzles are key to adventure games and can be a crucial component of action-adventure games. They are a source of great satisfaction once solved, and potentially a source of just as much frustration. Everyone remembers quitting a captivating game when stuck with an impossible puzzle, or even being unable to find the puzzle in the first place! Surprisingly, there is very

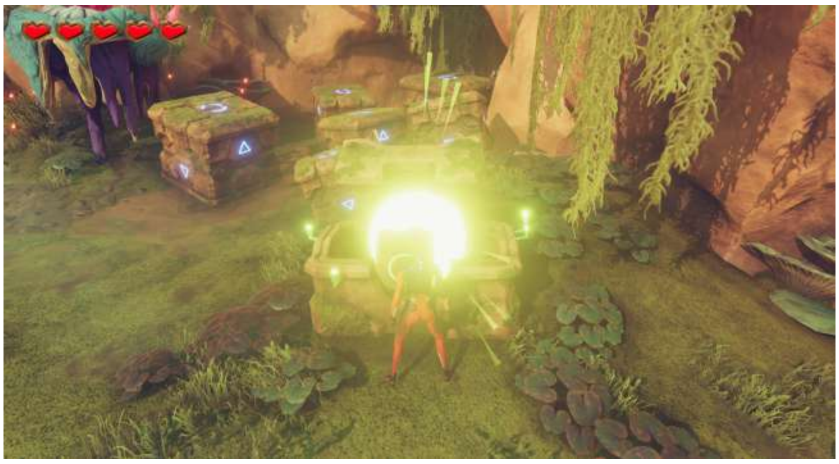


Figure 3.10: Health box pickup

little literature and research dedicated to this major aspect of game design. This feature offers a set of tangible rules for designing and integrating puzzles.

A detailed analysis of the great adventure game classics reveals one key aspect in their gameplay: investigation. Adventure games rely on the same mechanisms as a real-life investigation.What this tells us is that investigation is the driving force of adventure games; so puzzles should encourage the player to explore and interact with the environment, not spend hours on end in front of a static screen. The puzzle should not be limited to a lock mechanism. It must make a seamless whole with the clues surrounding it.

The puzzle elements in this game are about opening doors. That is the player need to explore the world need to find ways to open the doors in the game. Indeed, gameplay is virtually always limited to a cycle where a player must explore locations to discover objects, clues or mechanisms until the player realizes he or she is stuck in a game area and needs to find a way to move on when finally, the player discovers the lock/unlock feature using clues previously collected. Sometimes the player need to find simple stepping stones to open some doors, but the stepping stones won't be easily visible to the player or it will be a bit challenging to get there (Figure 3.11). And some door will only open if you complete some other objects like special crystals (Figure 3.12).

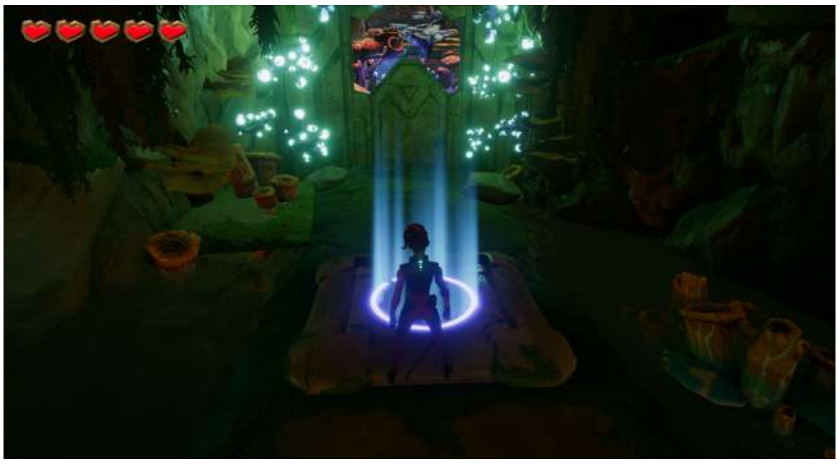


Figure 3.11: Stepping stone activates the door.

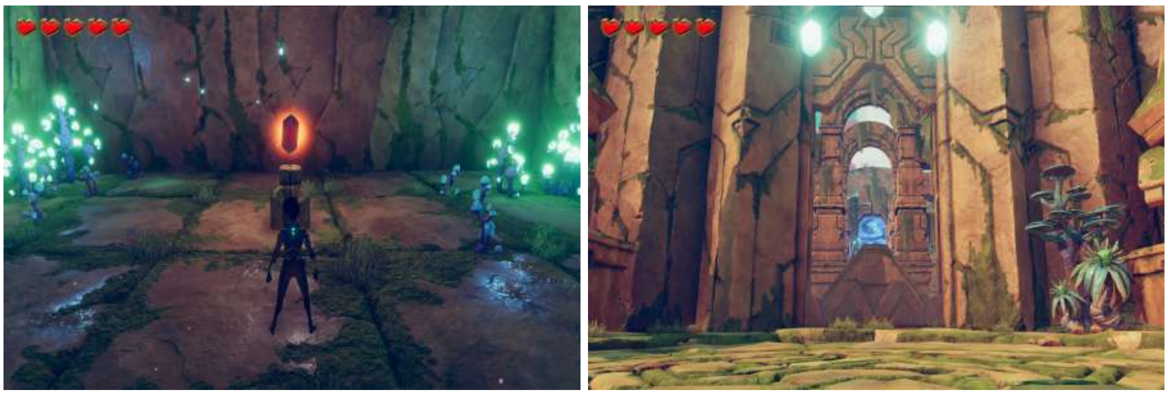


Figure 3.12: After the player collects all the necessary crystals the door opens.

3.4.3 Module - 3: Game Mechanics and Animation

Animation in game

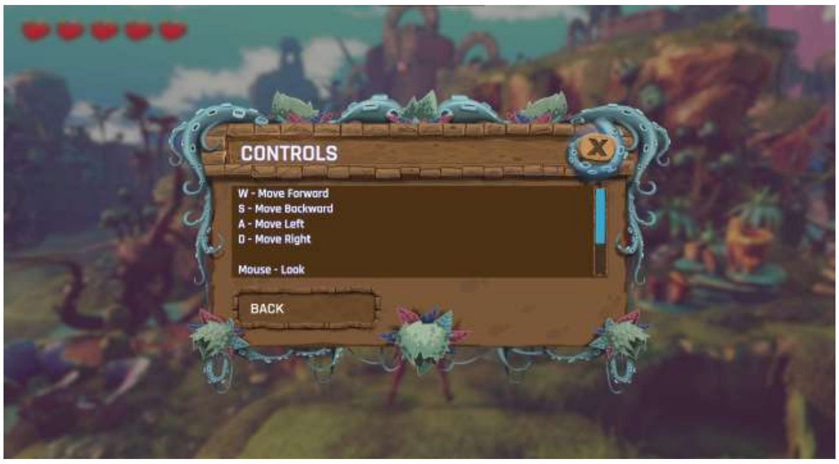
Computer animation is the process used for generating animated images. The more general term computer-generated imagery (CGI) encompasses both static scenes and dynamic images, while computer animation only refers to the moving images. Modern computer animation usually uses 3D computer graphics, although 2D computer graphics are still used for stylistic, low bandwidth, and faster real-time renderings. Sometimes, the target of the animation is the computer itself, but sometimes film as well.

To create the illusion of movement, an image is displayed on the computer monitor and repeatedly replaced by a new image that is similar to it, but advanced slightly in time (usually at a rate of 24, 25 or 30 frames/second). This technique is identical to how the illusion of movement is achieved with television and motion pictures. For 3D animations, all frames must be rendered after the modeling is complete.

In The Explorer almost every objects have it's own unique animations. If the player stay idle for some seconds the player will start to do random animated poses. All the enemy characters can roam around the world freely. The plants have unique animations based on the their types and they give the effects of wind in the game. Other animations are player movements and the movement of other NPCs(Non-Playable Characters).

Key bindings and game controls

The player can control Ellen using a keyboard and a mouse. The controls are the commonly used key combinations. The player can use either the arrow keys or the w,a,s,d combination. Space bar is used for jumping. Mouse is used for looking around and left mouse button is used for attacking (Figure 3.13).



W - Move Forward
S - Move Backward
A - Move Left
D - Move Right

Mouse - Look

Figure 3.13: Controls

Other game mechanics:

- Game physics
- Actions

3.4.4 Module - 4: Game AI

A.I. in Game

AI in gaming refers to responsive and adaptive video game experiences. These AI-powered interactive experiences are usually generated via non-player characters, or NPCs, that act intelligently or creatively, as if controlled by a human game-player. AI is the engine that determines an NPC's behavior in the game world.

While AI in some form has long appeared in video games, it is considered a booming new frontier in how games are both developed and played. AI games increasingly shift the control of the game experience toward the player, whose behavior helps produce the game experience.AI procedural generation, also known as procedural storytelling, in game design refers to game data being produced algorithmically rather than every element being built specifically by a developer. AI implementations in this project are the following:

- Enemy behavior
- Combat AI
- Player recognition
- Hunting

There are three types of enemies in this game. All three of them can be classified into three difficulty levels. The low level and easier to kill is called Chomber.They are the most common enemy in the game and you can find them almost everywhere. They can only see you when you are in front of them but they have a high player detection radius and angle. Once they see you they will follow you and will start to attack. You can kill them with one hit or run. After leaving the detection radius they will stop chasing the player (Figure 3.14). The second type of enemies are called Spitter. They are medium level enemies. They can also be killed with one blow from the staff. But what make them challenging is their unique ability to spit venom on to our body. And if we approach to kill them they will run away and if we get too close we will lose our health because of their venomous thorns. You will have to dodge the venom and kill them quickly (Figure 3.15). The last and powerful enemy is called Grenadier. They are boss level enemies and the highest difficulty level enemy. The player need to be very careful while dealing with these guys. They have a special abilities to fire explosives at us and can smash us on to the ground with very powerful electric surges. They can be only killed with keenness and timing. And make every hit countable (Figure 3.16).

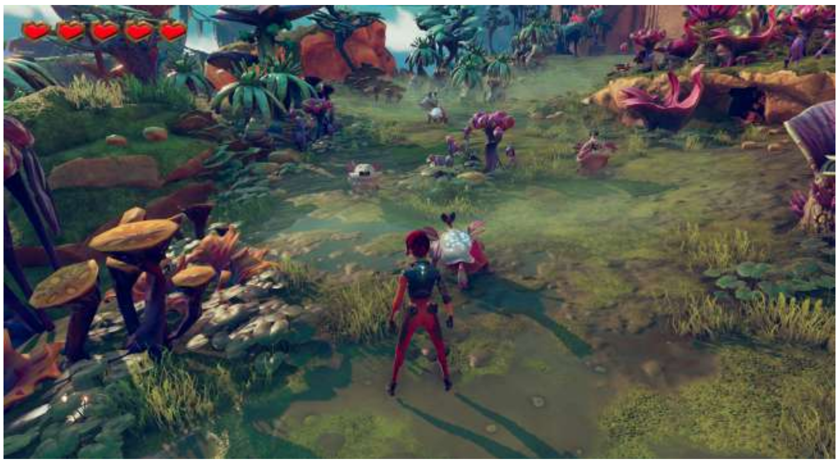


Figure 3.14: Chombers

Figure 3.15: Spitter

3.4.5 Module - 5: User interface

The game UI works on basic Canvas. But the Main menu is built using both canvas and 3D builder which was created for the animated intro scene (Main Menu). This intro scene was further duplicated and modified to create other canvases. The main menu is easily navigable. You can start the game right away or explore the options menu. The options menu have the audio and controls options (Figure 3.17). The player can edit the audio levels in audio options.

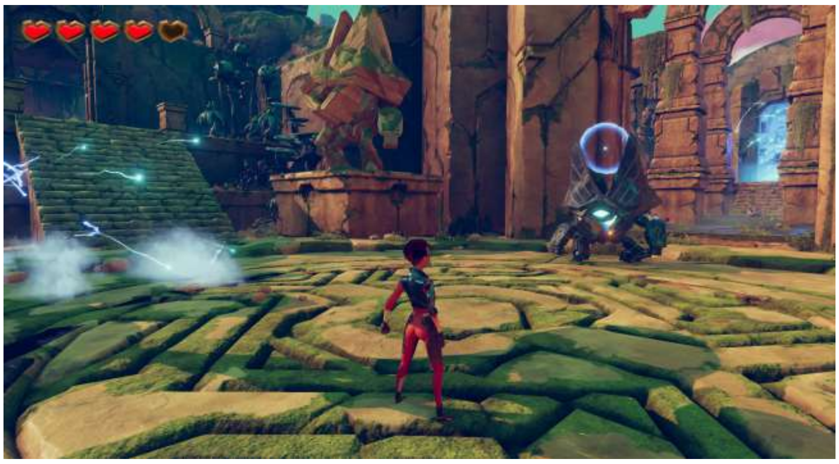


Figure 3.16: Grenadier

You can change the master volume, music volume and the SFX volume (Figure 3.18). The controls will show you the keys for playing the game. The back button will bring you back to the previous pages. The exit button will stop the game and will close the window.

Main Menu and Game Controls:

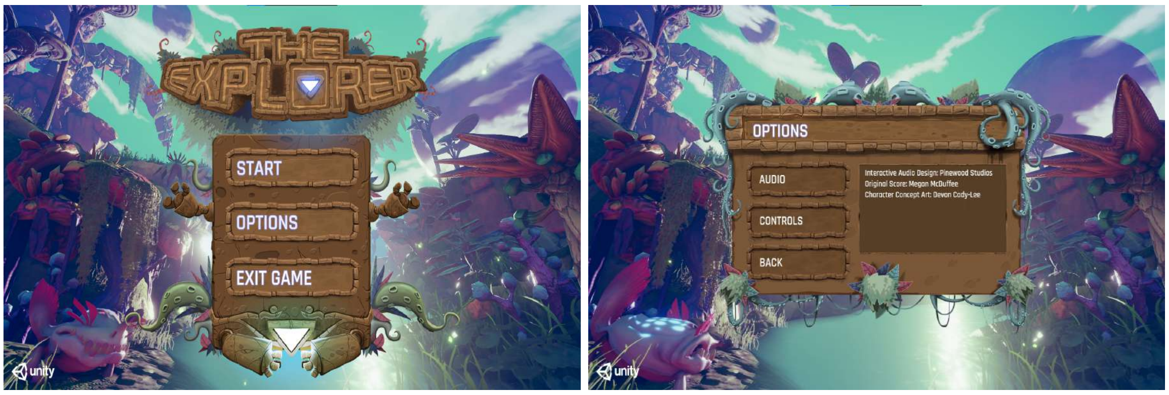

THE EXPLORER
START
OPTIONS
EXIT GAME
unity
OPTIONS
AUDIO
CONTROLS
BACK
Interactive Audio Design: Pinewood Studios
Original Score: Megan McDuffee
Character Concept Art: Devon Cody-Lee
unity

Figure 3.17: Main menu, Options

After pressing the start button the loading screen will appear and the game will load. After starting the gameplay you can pause the game by pressing the escape key on keyboard (Figure 3.19). You can restart the level if you want to. You can access the options menu from here also. Or you can quit the game from there. By clicking on the X mark on the UI will bring us back to the gameplay. You can also return to the gameplay by pressing escape again.

Loading Screen and Pause Menu

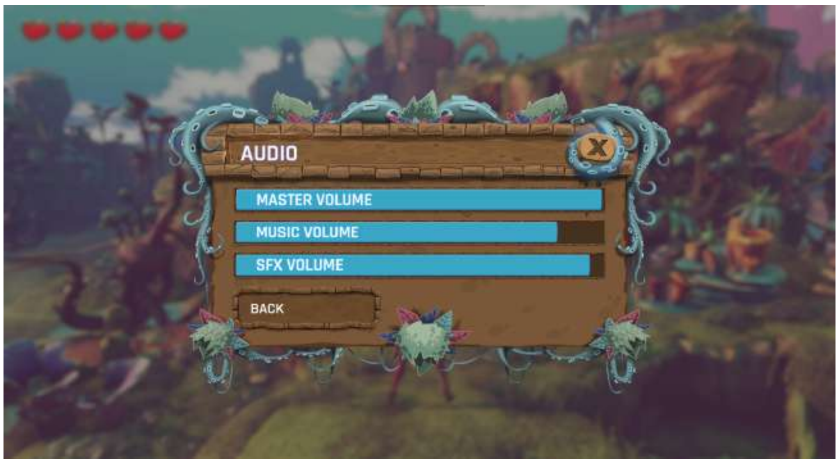


Figure 3.18: Audio Controls

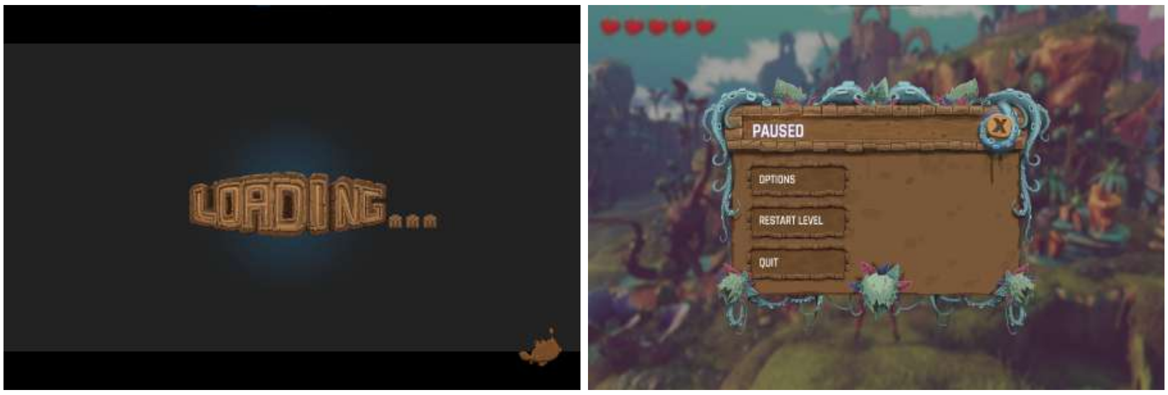


Figure 3.19: Loading Screen, Pause Menu

3.5 Developing Environment

3.5.1 Hardware Requirements

- Processor - Intel Core i5 (min)
- Clock Speed - 1.5 GHz (min)
- RAM - 8 GB (min)
- Hard Disk - 100 GB or (min)
- GPU - 2 GB (min)

3.5.2 Software Requirements

- Operating System - Windows 10 or above
- Game Engine - Unity
- Programming Language - C
- IDE - Visual Studio 2019

3.6 Introduction to the Platform

Unity is an all-purpose game engine that supports 2D and 3D graphics, drag and drop functionality and scripting through C#.

Two other programming languages were supported: Boo, which was deprecated with the release of Unity 5 and UnityScript(JavaScript) which was deprecated in August 2017 after the release of Unity 2017.1. The engine targets the following graphics APIs: Direct3D on Windows and Xbox One; OpenGL on Linux, macOS, and Windows; OpenGL ES on Android and iOS; WebGL on the web; and proprietary APIs on the video game consoles.

Additionally, Unity supports the low-level APIs Metal on iOS and macOS and Vulkan on Android, Linux, and Windows, as well as Direct3D 12 on Windows and Xbox One. Within 2D games, Unity allows importation of sprites and an advanced 2D world renderer.

For 3D games, Unity allows specification of texture compression and resolution settings for each platform that the game engine supports, and provides support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects. Unity also offers services to developers, these are: Unity Ads, Unity Analytics, Unity Certification, Unity Cloud Build, Unity Every play, Unity IAP, Unity Multiplayer, Unity Performance Reporting and Unity Collaborate.

3.6.1 Game Engine

A game engine is a software development environment designed for people to build video games. Developers use them to create games for consoles, mobile devices, and personal computers. The core functionality typically provided by a game engine includes a rendering engine (”renderer”) for 2D or 3D graphics, a physics engine or collision detection (and collision response), sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, localization support, scene graph, and may include video support for cinematics. The process of game development is often economized, in large part, by reusing/adapting the same game engine to create different games or to make it easier to port games to multiple platforms.

3.6.2 Types of Game Engine

XNA Game Studio (Microsoft)

Microsoft XNA is one of the best opportunities in the game industry right now. It’s a .NET programming package that lets you make any small to medium sized game you’re capable of coding, and then it allows you to share your creation with the world on Xbox Live. How cool is that? It’s like a cheat code or a cool friend who lets you bypass the long line at a club; the exposure of Xbox Live isn’t a free perk but it’s something that’s hard for the others on this list to match.

Unreal Engine (Epic Games)

This is pretty much the undisputed best game engine in the current era of the game industry. With literally hundreds of games – even MMO games – that have used it in the past, the Unreal Engine is capable of creating any game for any genre with any budget. While it’s by no means something an amateur should tackle solo to make the game of their dreams, it is however a great tool for creating the 3D level of your dreams.

Unreal Tournament 2004 comes with UnrealEd – the Unreal Engine’s level editor – bundled in for free, and after one or two nights of reading and watching tutorials online you’ll be able to create your own 3D level that you and your friends can play online. It lets you use hundreds of objects and textures from the game to create your own levels, but importing your own data is also an option. Those interested in just the FPS angle should also check out Valve’s Hammer Editor, which is another valuable game bundle worth having and knowing.

Unity (Unity Technology)

With the humongous rise of handheld electronics in recent years, the usage of Unity has soared and has even stolen a bit of thunder from the rest on this list. It's similar to the Torque and Blender game engines – which are also really worth checking out – and it's capable of creating games for PC, Mac, Wii, Xbox 360, and PS3, but its popularity in the past year has been largely due to its use in developing iPad and iPhone games. It's so popular right now that I actually plan to take my own advice and learn Unity in the coming months to stay up to date with the game industry!

3.7 Unity 3D Game Engine

Unity is a cross-platform game engine developed by Unity Technologies. which is primarily used to develop video games and simulations for computers, consoles and mobile devices. First announced only for OS X, at Apple's Worldwide Developers Conference in 2005, it has since been extended to target 27 platforms.

Six major versions of Unity have been released. At the 2006 WWDC show, Apple named Unity as the runner up for its Best Use of Mac OS X Graphics category.

Features of Unity:-

- 2D
- 3D
- Graphics
- Physics
- Scripting
- Multiplayer and Networking
- Audio
- Animation
- Timeline
- UI

- Navigation and Path finding
- Virtual Reality
- Open-source repositories
- Asset Store Publishing
- Platform-specific
- Experimental

3.7.1 ProBuilder

Now part of Unity, ProBuilder is a unique hybrid of 3D modeling and level design tools, optimized for building simple geometry but capable of detailed editing and UV unwrapping too.Use ProBuilder to quickly prototype structures, complex terrain features, vehicles and weapons, or to make custom collision geometry, trigger zones, or nav meshes.ProBuilder also takes advantage of Unity's seamless round-tripping capabilities with digital content-creation tools (like Maya), so you can further detail and polish models with your favorite tools.Many games made with Unity feature ProBuilder modeling and level designs, including SUPERHOT, Tacoma, Tunic, Manifold Garden, Super Lucky's Tale, DESYNC, and more.

Key features

Extrude and Inset:

Select any face or open edge, and simply hold Shift while moving, rotating, or scaling to extrude or inset. You can shape objects in any form. You can even extrude multiple faces/edges at once. You can flip the normals of all faces on the selected object(s), which is especially useful for converting an exterior modeled shape into an interior space. You can use the Mirror action to create identical copies of objects and make symmetrical items: Build one half, mirror it, then weld the two together for a perfectly symmetrical result (Figure 3.20).

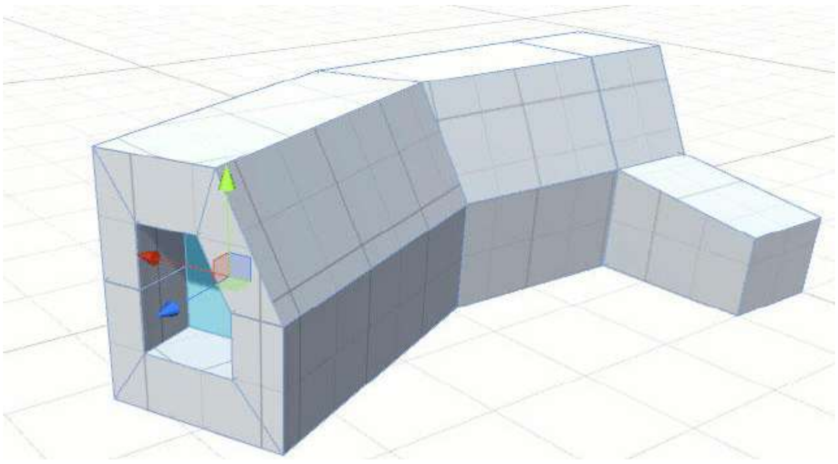


Figure 3.20: Extrude and Inset

Versatile Poly Shapes:

Once you create a Poly Shape, you can edit its control points and other settings at any stage. In the Scene view, simply click to add control points. These will form the outer bounds of your mesh. Once you finish placing points, press Return to finalize the shape. Even after the initial creation steps, you can still modify a Poly Shape. You can move existing control points by clicking and dragging, add/remove new points, or even flip the normals (Figure 3.21).

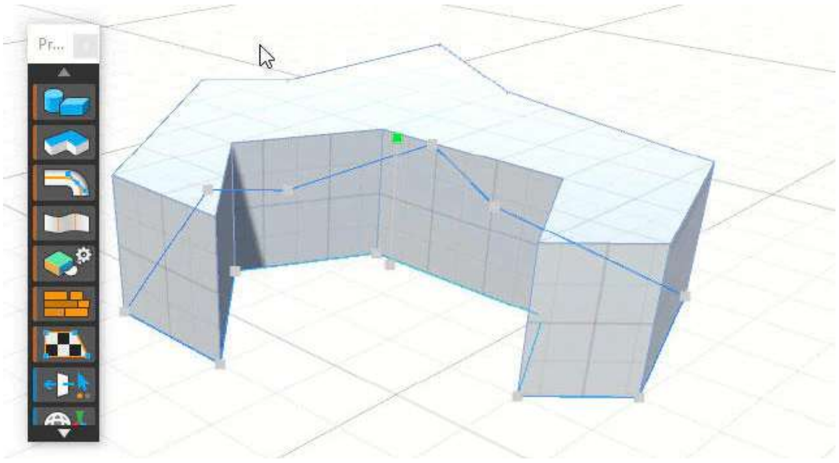


Figure 3.21: Versatile Poly Shapes

Dynamic User Interface:

Customize the toolbars to fit your needs, and take advantage of the ProBuilder window which is a ”smart toolbar” to dynamically match your current edit mode and element selection. Modify default settings, shortcut keys, resize the Toolbar, and switch between text and icon modes. ProBuilder will reorder the icon or text content to best fit the window size. The Toolbar is color-coded to help you quickly choose tools by type: Orange for Tool Panels, blue for Selection Tools, green for Object Actions, and red for Geometry Actions (Vertex, Edge, Face). Most ProBuilder windows can be changed from docking to floating or vice-versa through their context menu (Figure 3.22).

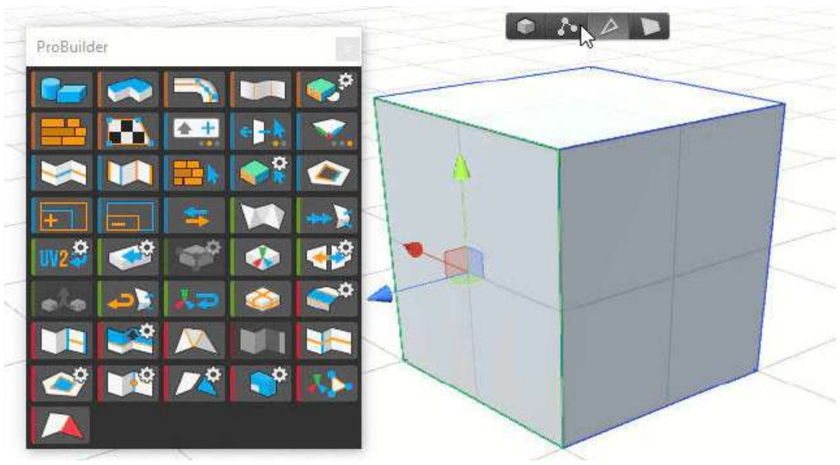


Figure 3.22: Dynamic User Interface

In-scene UV controls

ProBuilder features a complete UV Editor window for manual unwrapping and fine- grained control, including projection modes, UV welding, seam control, element pivots, and much more. Texture Mapping in ProBuilder makes it fast and easy to apply materials (textures) to an object, and adjust the offset, rotation, and tiling of the object's UV. ProBuilder provides both automatic and manual texturing (Figure 3.23).

methods:

Auto UVs: Use this for simple texturing work, especially architectural or hard-surface items. Tiling, offset, rotation, and other controls are available, while ProBuilder automatically handles projection and updates as you work.

Manual UV Editing: Use a full UV Editor to precisely unwrap and edit UVs, render UV templates, project Us and more. You can use a mix of auto and manual UVS, even on the same object. This is especially useful when some parts of a model need tiling textures, while others are unwrapped.

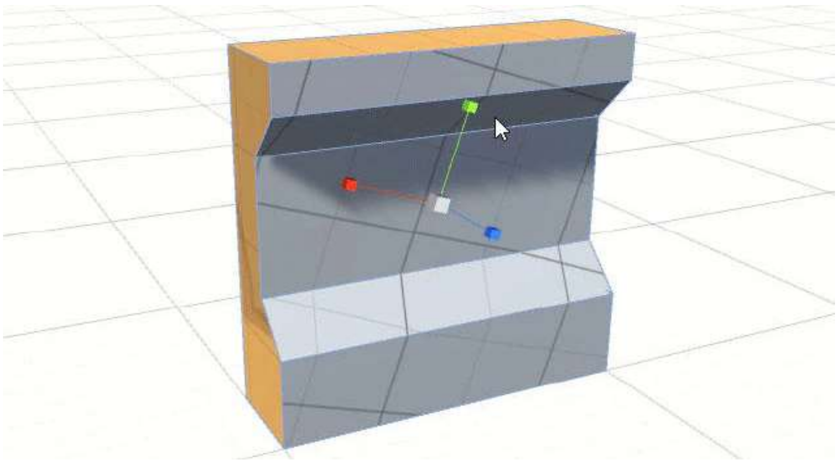
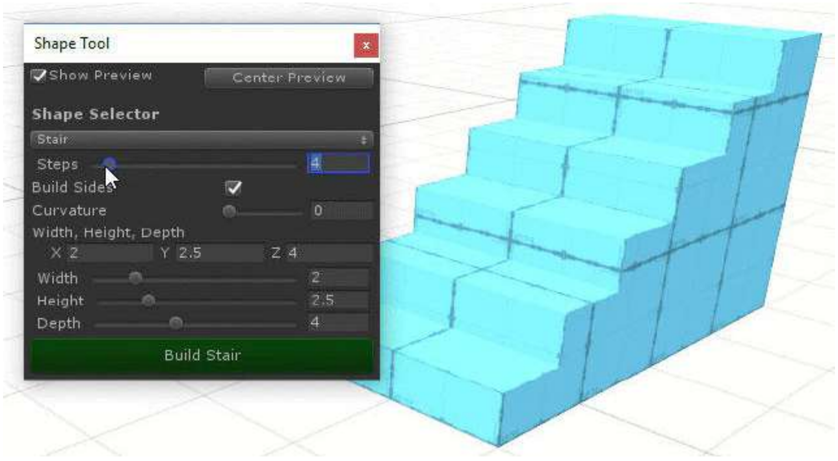


Figure 3.23: In-scene UV controls

Procedural Shapes

Create new editable shapes like cylinders and stairs. Each shape has specific properties that can be customized before creation. For example, the Stairs shape lets you choose items like step height, arc, and the parts of the stairway to build. Choose a shape to start with, then set unique parameters, such as height, width, depth, and number of cuts. It includes many shapes, including: Cone, Arch, Pipe, Stairway, Door, Cylinder, and Sprite (Figure 3.24).



✓ Show Preview
Stair
Steps 4
Build Sides ✓
Curvature 0
Width, Height, Depth X 2 Y 2.5 Z 4
Width 2
Height 2.5
Depth 4

Figure 3.24: Procedural Shapes

Vertex Coloring

Quickly apply overlay colors to individual faces. Vertex coloring is also especially useful for early map testing and designating team areas You can do more with vertices in ProBuilder:

for example, collapse all selected vertices to a single point regardless of distance, merge them within;a set distance of one another, create a new edge connecting the selected vertices, or split a vertex into its individual vertices so that they can be moved independently (Figure 3.25).

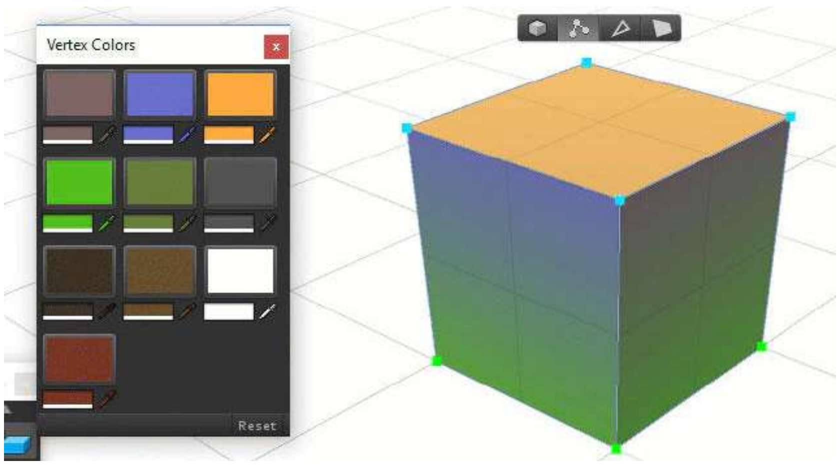


Figure 3.25: Vertex Coloring

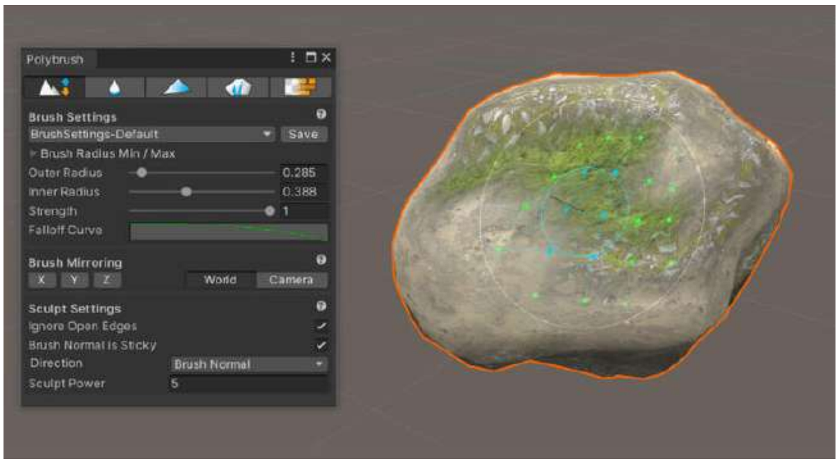
3.7.2 Polybrush

Polybrush enables you to blend textures and colors, sculpt meshes, and scatter objects directly in the Unity Editor. Combined with ProBuilder, Polybrush gives you a complete in-editor level design solution. You can also take advantage of Unity's seamless round-tripping with digital content-creation tools (like Maya) so you can further polish models with your favorite tools.

Key features

Sculpt Meshes:

Great for custom terrains and organic objects. You can sculpt complex shapes directly in the Editor and use the Push/Pull tool to move vertices in positive and negative directions along an axis. Simply select any mesh object, then click and drag while hovering over the selected mesh. The actual movement of vertices is affected based on your Brush Settings (Figure 3.26).



Polybrush

Brush Settings
BrushSettings-Default · Save
Brush Radius Min / Max
Outer Radius 0.285
Inner Radius 0.388
Strength 1
Falloff Curve

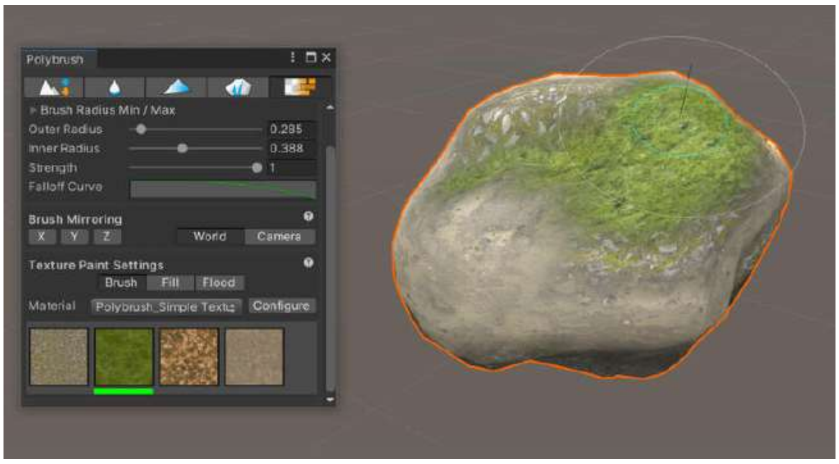
Brush Mirroring
X Y Z · World · Camera

Sculpt Settings
Ignore Open Edges ✓
Brush Normal is Sticky ✓
Direction: Brush Normal
Sculpt Power: 5

Figure 3.26: Sculpt meshes

Blend Textures and Colors:

Use blend shaders and customization options. Texture Painting mode is for texture blending shaders. Polybrush includes a few different blend materials to get you started: Standard Texture Blend, TriPlanar Texture Blend, and Unlit Texture Blend. You can also write your own texture blending shaders (Figure 3.27).



Polybrush

Brush Radius Min / Max
Outer Radius 0.285
Inner Radius 0.388
Strength 1
Falloff Curve

Brush Mirroring
X Y Z · World · Camera

Texture Paint Settings
Brush · Fill · Flood
Material: Polybrush_Simple Textu · Configure

Figure 3.27: Blend Textures and Colors

Paint Vertex Colors:

Create colorful shapes, enhance textures or even hand-paint lighting. Color Paint mode enables you to set the vertex colors of a mesh with a brush or paint bucket. Use the toolbar under Paint Settings to choose between the two modes. Note that Vertex Color mode will only work if the shader material supports vertex colors. Polybrush includes some default materials that support vertex colors. Use one of these materials if you want to paint colors on a mesh (Figure 3.28).

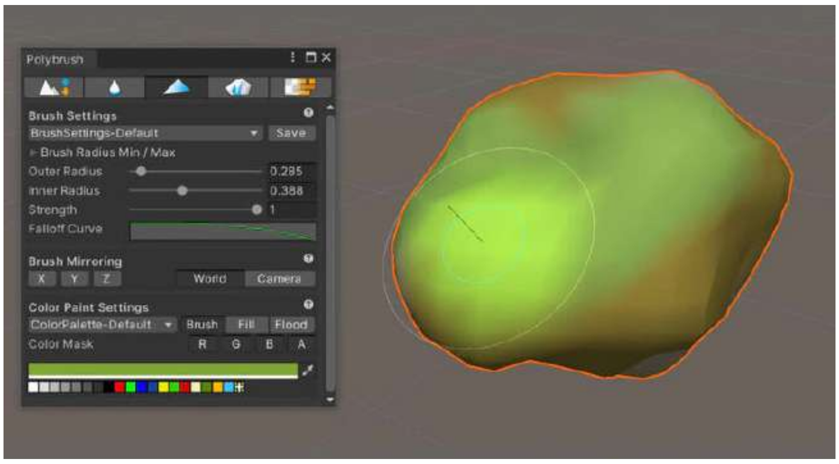


Figure 3.28: Paint Vertex Colors

Scatter Objects:

Add rocks, grass, debris and other geometry details The latest version of Polybrush allows you to scatter objects in the world with highly customizable brushes (Figure 3.29).

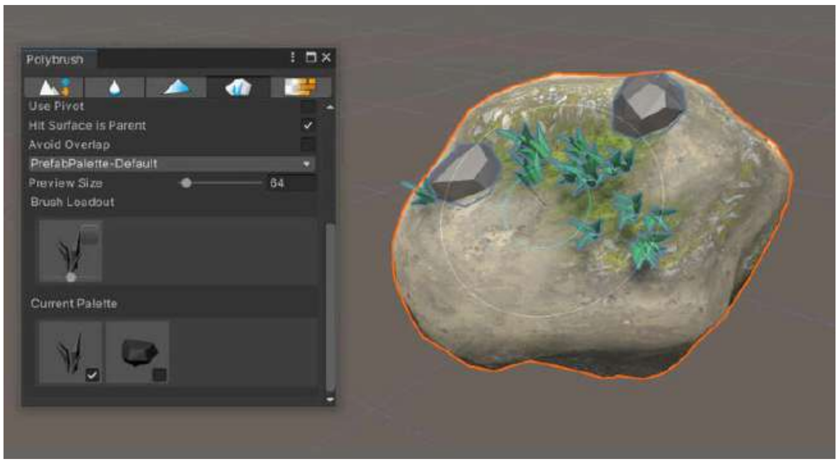


Figure 3.29: Scatter Objects

3.8 C# or C Sharp

C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO). C# was developed by Anders Hejlsberg and his team during the development of .Net Framework. C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

The following reasons make C# a widely used professional language :

- It is a modern, general-purpose programming language
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.

3.9 VISUAL STUDIO 2019

The Visual Studio interactive development environment (IDE) is a creative launching pad that you can use to view and edit nearly any kind of code, and then debug, build, and publish apps for Android, iOS, Windows, the web, and the cloud. There are versions available for Mac and Windows. This topic introduces you to the features of the Visual Studio IDE. We'll walk through some things you can do with Visual Studio and how to install and use it, create a simple project, get pointers on debugging and deploying code, and take a tour of the various tool windows.

With Visual Studio Tools for Unity (VSTU), you can use Visual Studio to write game and editor scripts in C and then use its powerful debugger to find and fix errors. The latest release of VSTU includes syntax coloring for Unity's ShaderLab shader language, better debugger visualizations, and improved code generation for the MonoBehavior wizard. VSTU also brings your Unity project files, console messages, and the ability to start your game into Visual Studio so you can spend less time switching to and from the Unity Editor while writing code.Visual Studio Tools for Unity registers a log callback so it can stream the Unity console to Visual Studio. If you have editor scripts that log information, you can plug them into the same callback to send your messages to Visual Studio.

3.10 Agile Methodology

The Agile methodology is a way to manage a project by breaking it up into several phases. It involves constant collaboration with stakeholders and continuous improvement at every stage. Once the work begins, teams cycle through a process of planning, executing, and evaluating. Continuous collaboration is vital, both with team members and project stakeholders.This project was completed in 2 sprints with 5 modules.

3.10.1 Sprint 1

- Implementing Player Character
- Creation of enemy characters

3.10.2 Sprint 2

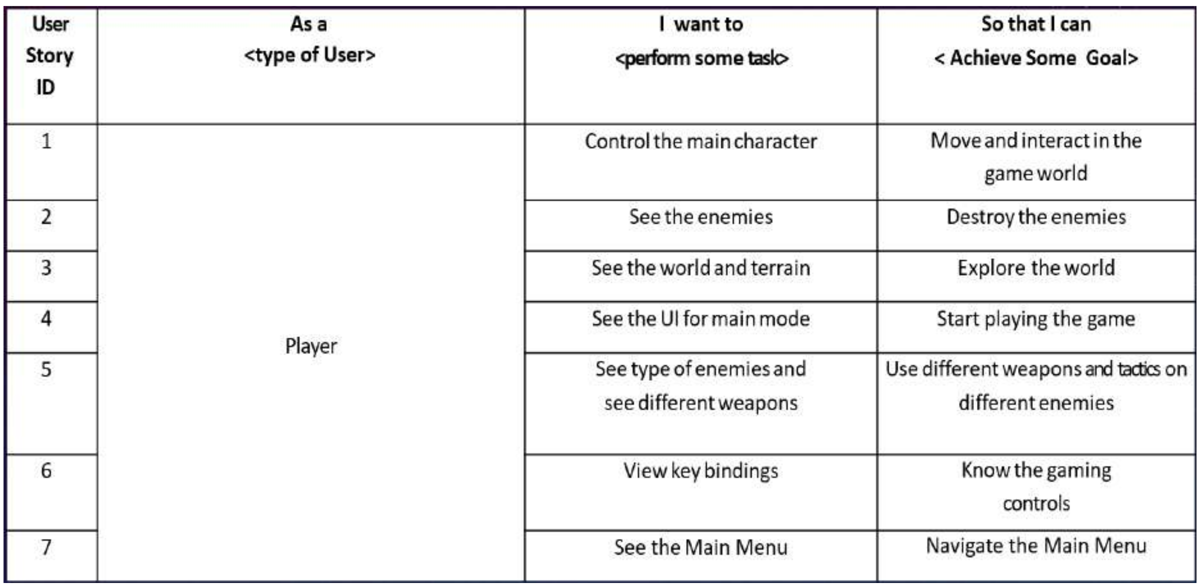
- World and terrain design
- Scripting
- Basic UI

3.10.3 Sprint 3

- Key bindings
- Weapons
- AI for enemy Characters
- Final UI

3.10.4 User Story

A key component of agile software development is putting people first, and user-stories put actual end users at the center of the conversation. Stories use non-technical language to provide context for the development team and their efforts. After reading a user story, the team knows why they are building what they're building and what value it creates. A user story is a tool used in agile software development to capture a description of a software feature from an end user perspective. The user story describes the type of user, what they want and why. A user story helps to create a simplified description of a requirement. User stories are one of the core components of an agile program. They help provide a user-focused framework for daily work which drives collaboration, creativity, and a better product overall. The user story of this system is given below.

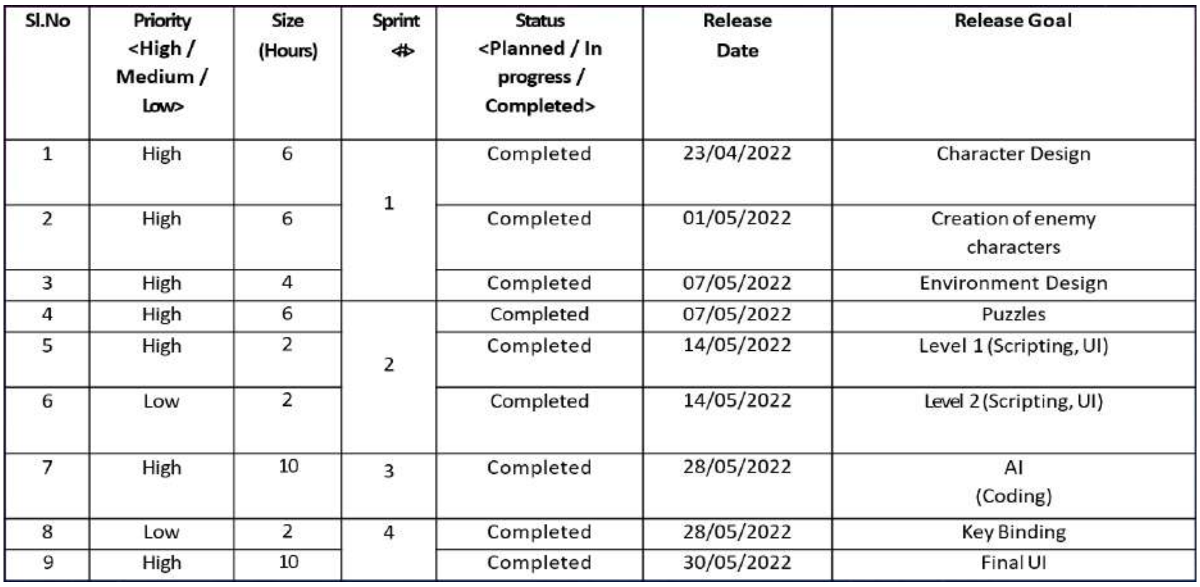


User Story ID	As a <type of User>	I want to <perform some task>	So that I can < Achieve Some Goal>
1	Player	Control the main character	Move and interact in the game world
2		See the enemies	Destroy the enemies
3		See the world and terrain	Explore the world
4		See the UI for main mode	Start playing the game
5		See type of enemies and see different weapons	Use different weapons and tactics on different enemies
6		View key bindings	Know the gaming controls
7		See the Main Menu	Navigate the Main Menu

Figure 3.30: User Story

3.10.5 Product Backlog

A product backlog is a list of the new features, changes to existing features, bug fixes, infrastructure changes or other activities that a team may deliver in order to achieve a specific outcome.The product backlog is the single authoritative source for things that a team works on. That means that nothing gets done that isn't on the product backlog. Conversely, the presence of a product backlog item on a product backlog does not guarantee that it will be delivered. It represents an option the team has for delivering a specific outcome rather than a commitment.It should be cheap and fast to add a product backlog item to the product backlog, and it should be equally as easy to remove a product backlog item that does not result in direct progress to achieving the desired outcome or enable progress toward the outcome. The Scrum Product Backlog is simply a list of all things that needs to be done within the project. It replaces the traditional requirements specification artifacts. These items can have a technical nature or can be user-centric e.g. in the form of user stories.The product backlog of the system is given below figure.

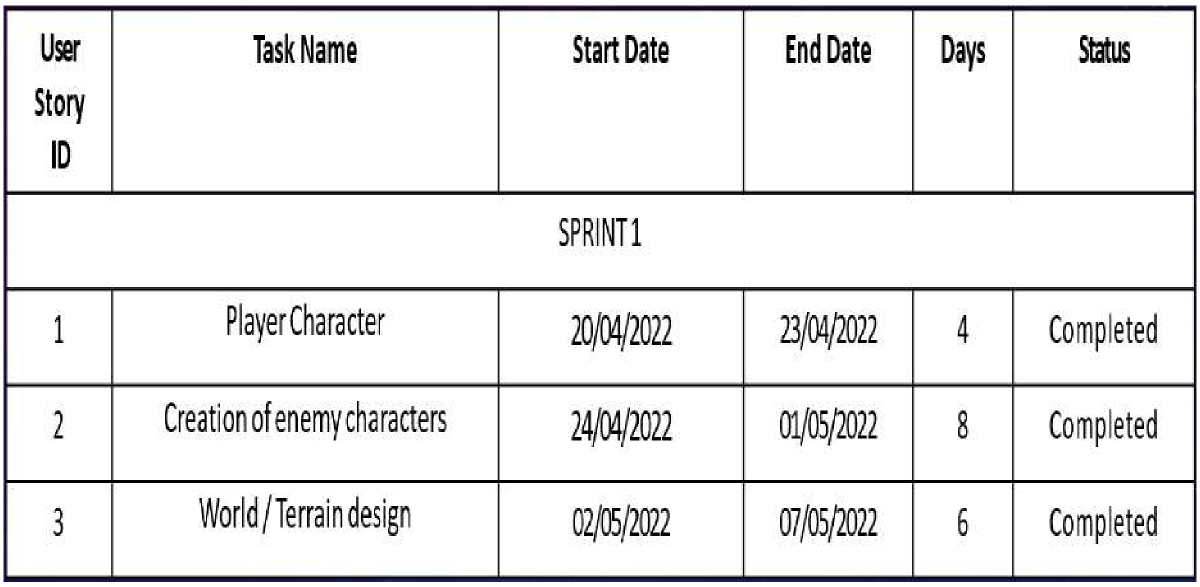


Sl.No	Priority <High / Medium / Low>	Size (Hours)	Sprint <#>	Status <Planned / In progress / Completed>	Release Date	Release Goal
1	High	6	1	Completed	23/04/2022	Character Design
2	High	6		Completed	01/05/2022	Creation of enemy characters
3	High	4		Completed	07/05/2022	Environment Design
4	High	6	2	Completed	07/05/2022	Puzzles
5	High	2		Completed	14/05/2022	Level 1 (Scripting, UI)
6	Low	2		Completed	14/05/2022	Level 2 (Scripting, UI)
7	High	10	3	Completed	28/05/2022	AI (Coding)
8	Low	2	4	Completed	28/05/2022	Key Binding
9	High	10		Completed	30/05/2022	Final UI

Figure 3.31: Product Backlog

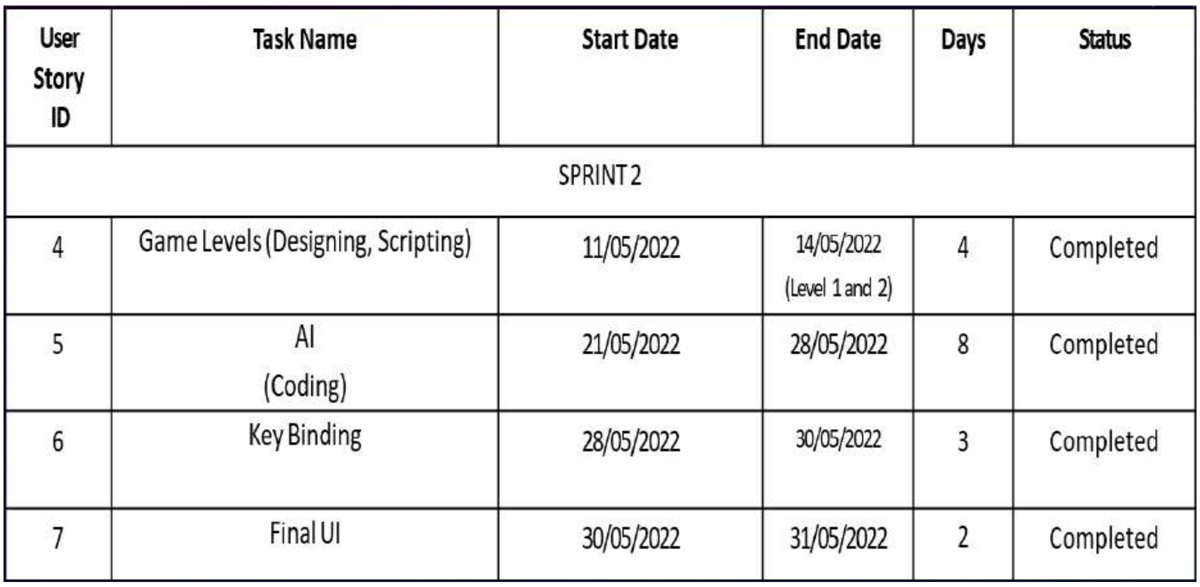
3.10.6 Project Plan

A project plan that has a series of tasks laid out for the entire project, listing task duration, responsibility assignments, and dependencies. Plans are developed in this manner based on the assumption that the Project Manager, hopefully along with the team, can predict up front everything that will need to happen in the project, how long it will take, and who will be able to do it. Project plan is given below figure. The project has two sprints.



User Story ID	Task Name	Start Date	End Date	Days	Status
SPRINT 1					
1	Player Character	20/04/2022	23/04/2022	4	Completed
2	Creation of enemy characters	24/04/2022	01/05/2022	8	Completed
3	World / Terrain design	02/05/2022	07/05/2022	6	Completed

Figure 3.32: Project Plan 1.1

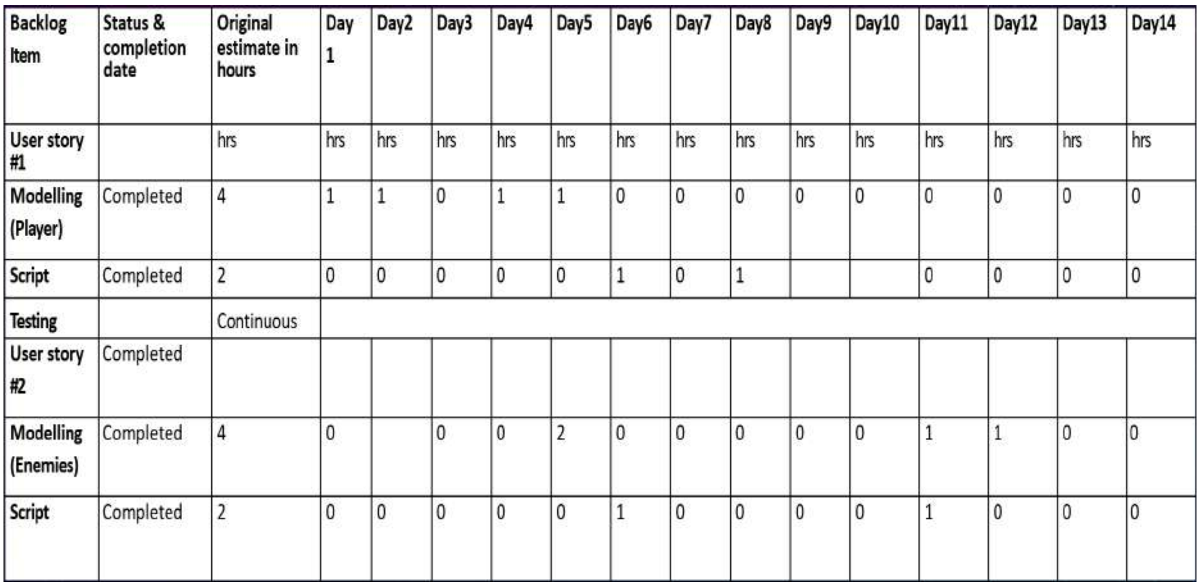


User Story ID	Task Name	Start Date	End Date	Days	Status
SPRINT 2					
4	Game Levels (Designing, Scripting)	11/05/2022	14/05/2022 (Level 1 and 2)	4	Completed
5	AI (Coding)	21/05/2022	28/05/2022	8	Completed
6	Key Binding	28/05/2022	30/05/2022	3	Completed
7	Final UI	30/05/2022	31/05/2022	2	Completed

Figure 3.33: Project Plan 1.2

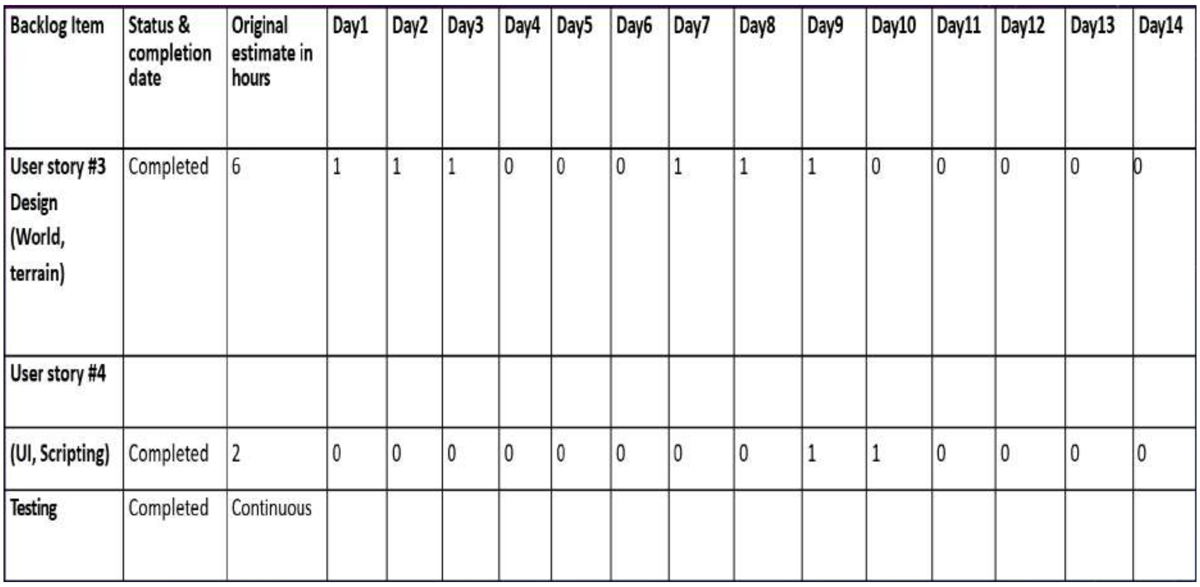
3.10.7 Sprint backlog

The sprint backlog is a list of tasks identified by the Scrum team to be completed during the Scrum sprint. During the sprint planning meeting, the team selects some number of product backlog items, usually in the form of user stories, and identifies the tasks necessary to complete each user story. Most teams also estimate how many hours each task will take someone on the team to complete.



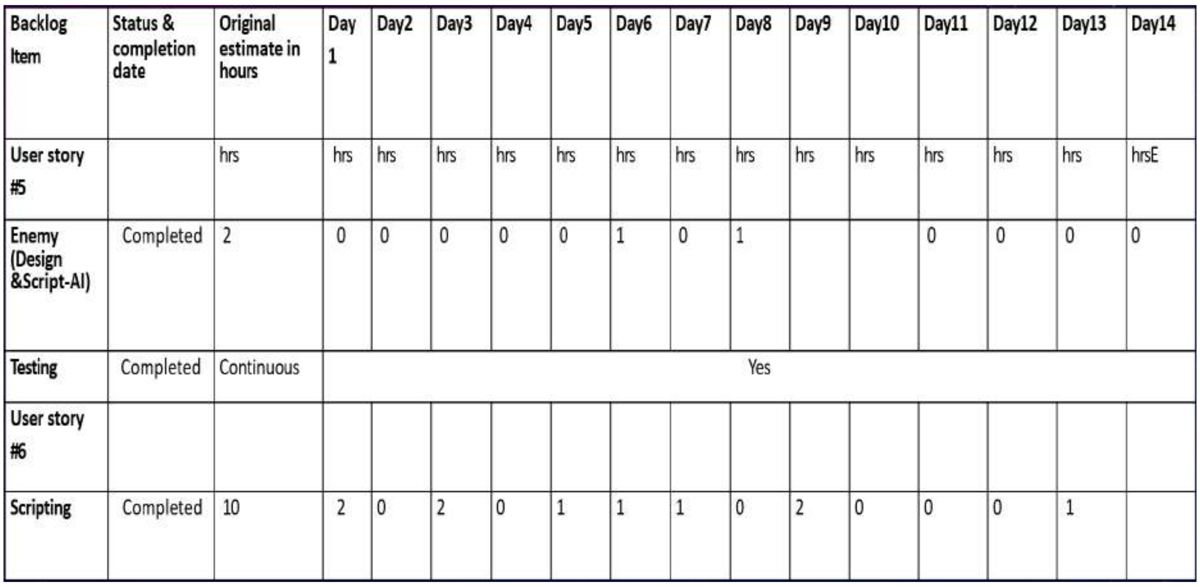
Backlog Item	Status & completion date	Original estimate in hours	Day 1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #1		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs
Modelling (Player)	Completed	4	1	1	0	1	1	0	0	0	0	0	0	0	0	0
Script	Completed	2	0	0	0	0	0	1	0	1			0	0	0	0
Testing		Continuous														
User story #2	Completed															
Modelling (Enemies)	Completed	4	0		0	0	2	0	0	0	0	0	1	1	0	0
Script	Completed	2	0	0	0	0	0	1	0	0	0	0	1	0	0	0

Figure 3.34: Sprint backlog 1.1



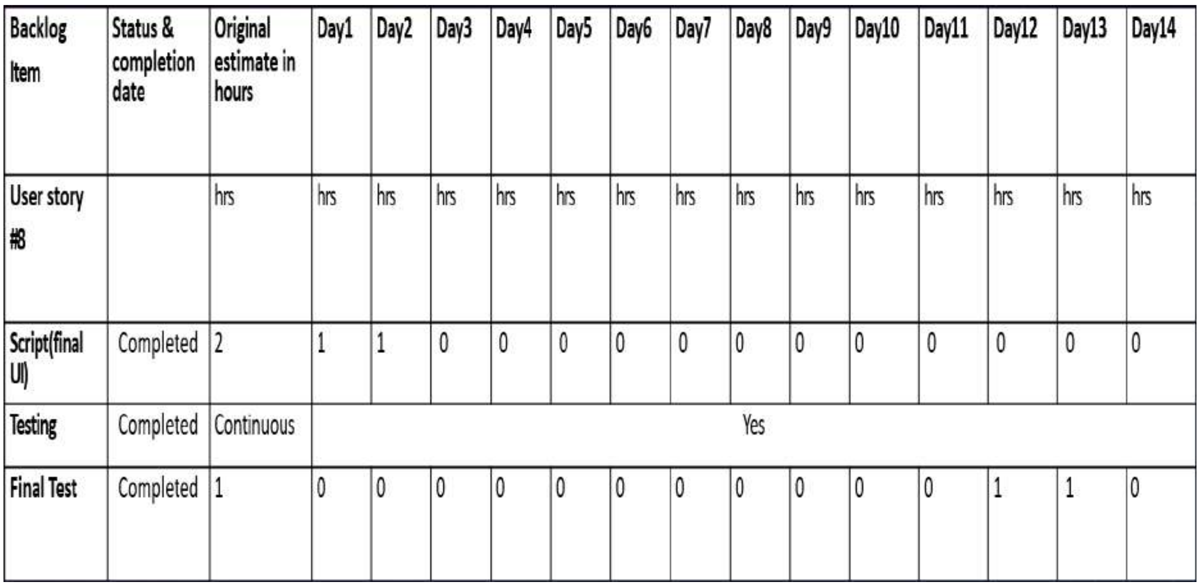
Backlog Item	Status & completion date	Original estimate in hours	Day1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #3 Design (World, terrain)	Completed	6	1	1	1	0	0	0	1	1	1	0	0	0	0	0
User story #4																
(UI, Scripting)	Completed	2	0	0	0	0	0	0	0	0	1	1	0	0	0	0
Testing	Completed	Continuous														

Figure 3.35: Sprint backlog 1.2



Backlog Item	Status & completion date	Original estimate in hours	Day 1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #5		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrsE
Enemy (Design &Script-AI)	Completed	2	0	0	0	0	0	1	0	1			0	0	0	0
Testing	Completed	Continuous								Yes						
User story #6																
Scripting	Completed	10	2	0	2	0	1	1	1	0	2	0	0	0	1	

Figure 3.36: Sprint backlog 1.3

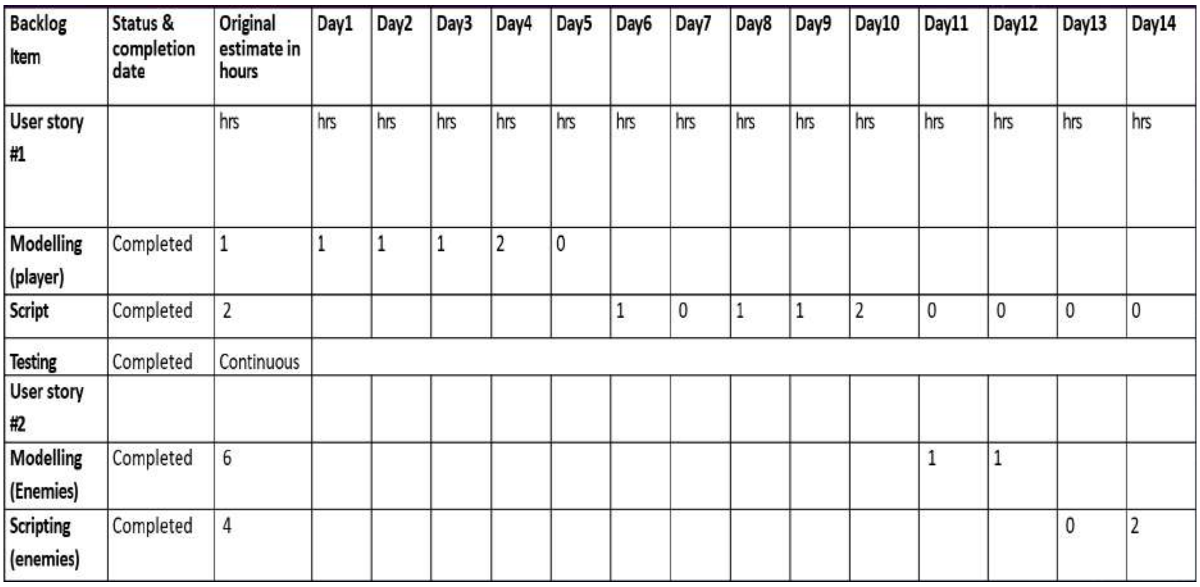


Backlog Item	Status & completion date	Original estimate in hours	Day1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #8		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs
Script(final UI)	Completed	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Testing	Completed	Continuous								Yes						
Final Test	Completed	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Figure 3.37: Sprint backlog 1.4

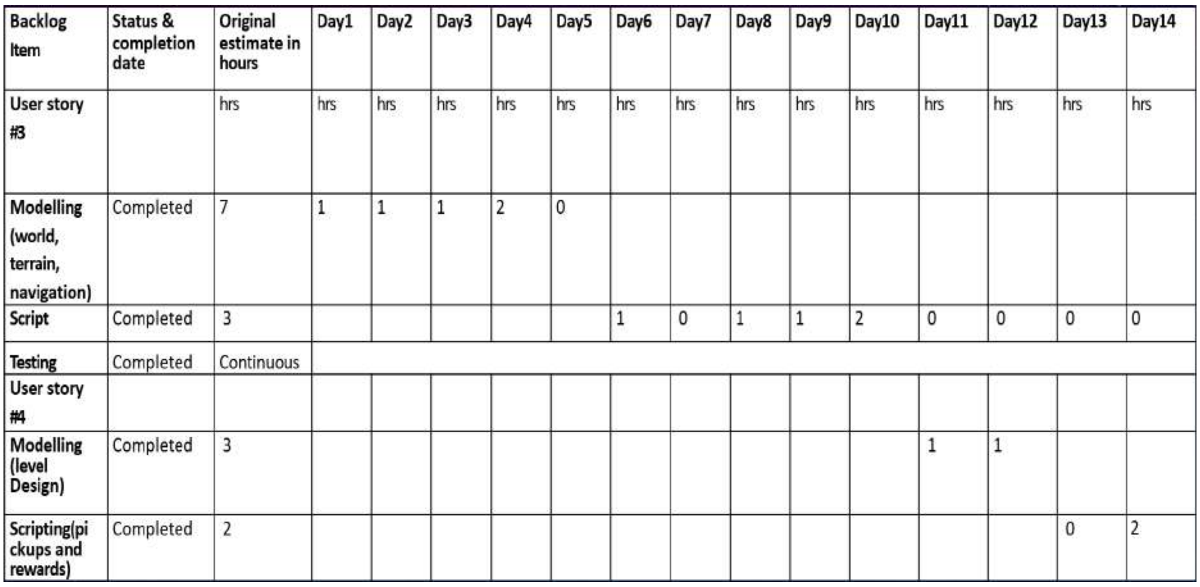
3.10.8 Sprint Actual

Actual sprint backlog is what adequate sprint planning is actually done by project team there may or may not be difference in planned sprint backlog. The detailed sprint backlog (Actual) is given below.



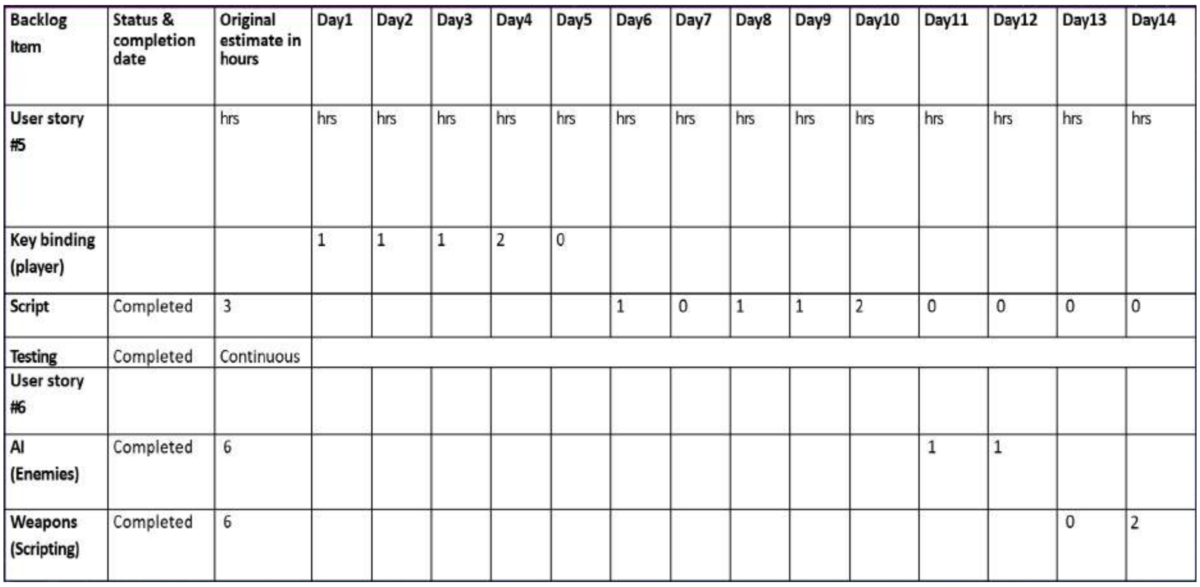
Backlog Item	Status & completion date	Original estimate in hours	Day1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #1		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs
Modelling (player)	Completed	1	1	1	1	2	0									
Script	Completed	2						1	0	1	1	2	0	0	0	0
Testing	Completed	Continuous														
User story #2																
Modelling (Enemies)	Completed	6											1	1		
Scripting (enemies)	Completed	4													0	2

Figure 3.38: Sprint 1 Actual



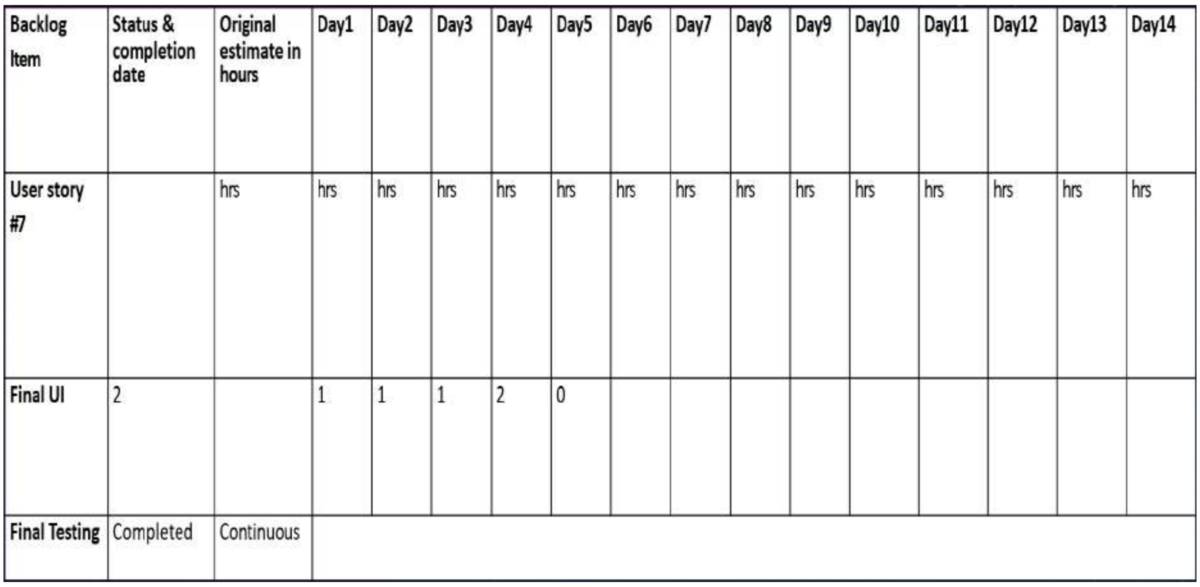
Backlog Item	Status & completion date	Original estimate in hours	Day1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #3		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs
Modelling (world, terrain, navigation)	Completed	7	1	1	1	2	0									
Script	Completed	3						1	0	1	1	2	0	0	0	0
Testing	Completed	Continuous														
User story #4																
Modelling (level Design)	Completed	3											1	1		
Scripting(pickups and rewards)	Completed	2													0	2

Figure 3.39: Sprint 2 Actual



Backlog Item	Status & completion date	Original estimate in hours	Day1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #5		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs
Key binding (player)			1	1	1	2	0									
Script	Completed	3						1	0	1	1	2	0	0	0	0
Testing	Completed	Continuous														
User story #6																
AI (Enemies)	Completed	6											1	1		
Weapons (Scripting)	Completed	6													0	2

Figure 3.40: Sprint 3 Actual 1.1



Backlog Item	Status & completion date	Original estimate in hours	Day1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #7		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs
Final UI	2		1	1	1	2	0									
Final Testing	Completed	Continuous														

Figure 3.41: Sprint 3 Actual 1.2

Chapter 4

Results and Discussions

4.1 Results

When we open the executable file, the game loads up and displays the main menu which shows the title of the game and has the level environment as the background. There are three buttons on the Main Menu, Start, Options and Exit Game(Figure 4.1: Main menu).

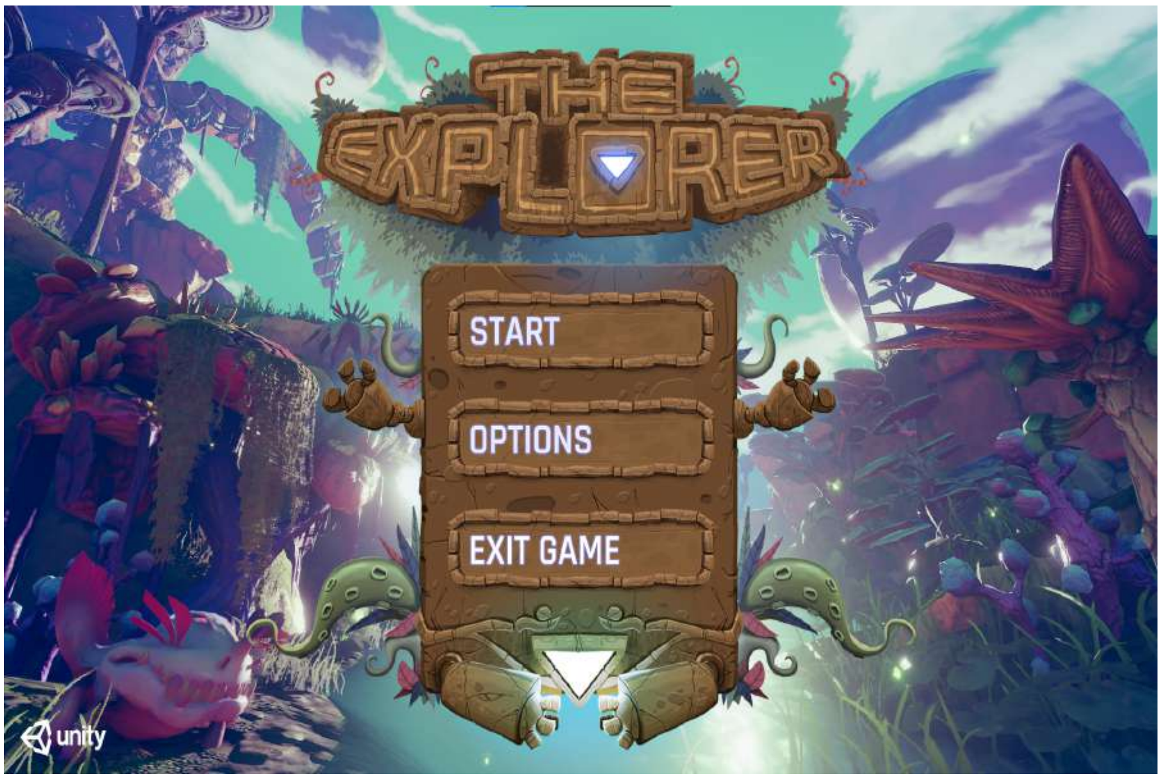
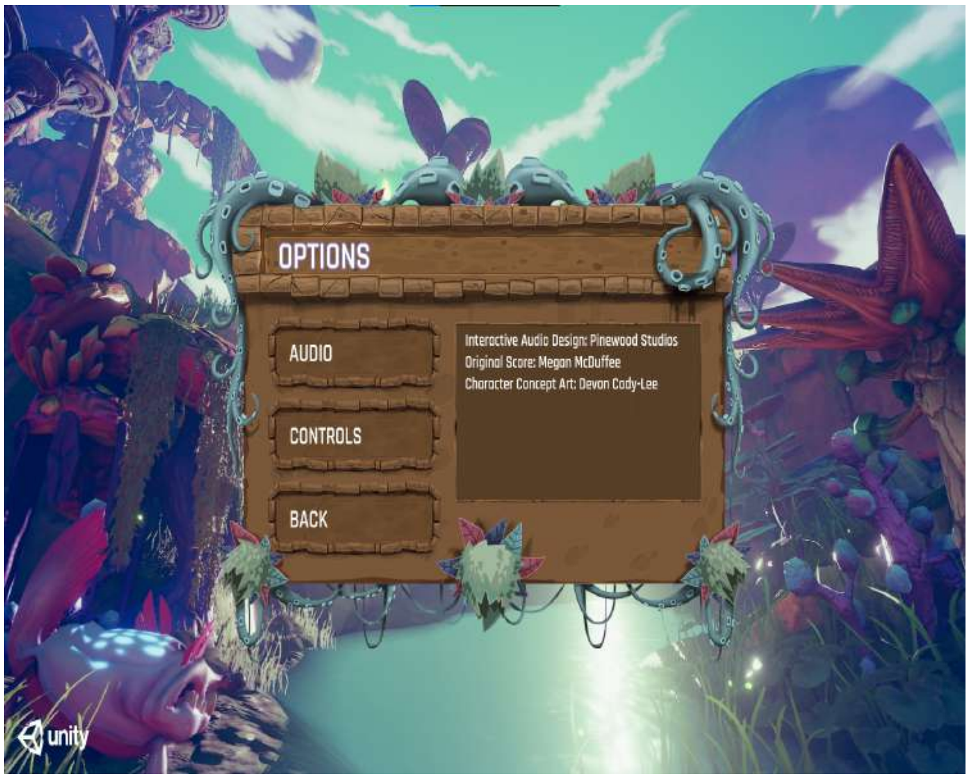


Figure 4.1: Main menu

When the Options is clicked, the player is redirected to the Options Menu(Figure 4.2: Game controls) which displays the Audio controls and gameplay controls that the player must follow to play the game. There is controls for keyboard + mouse combination. By clicking back button the the player is redirected to the Main Menu.



Interactive Audio Design: Pinewood Studios
Original Score: Megan McDuffee
Character Concept Art: Devon Cody-Lee

Figure 4.2: Options

Upon clicking the Start button the first level of the game will Load and the player will be spawned in to the game world. The scene will contain the open world where the player can explore and basic game UI components like player health meter(Figure 4.3:Game Play). The player can control the player using the assigned control buttons and move around the world and use the mouse button kill enemies.

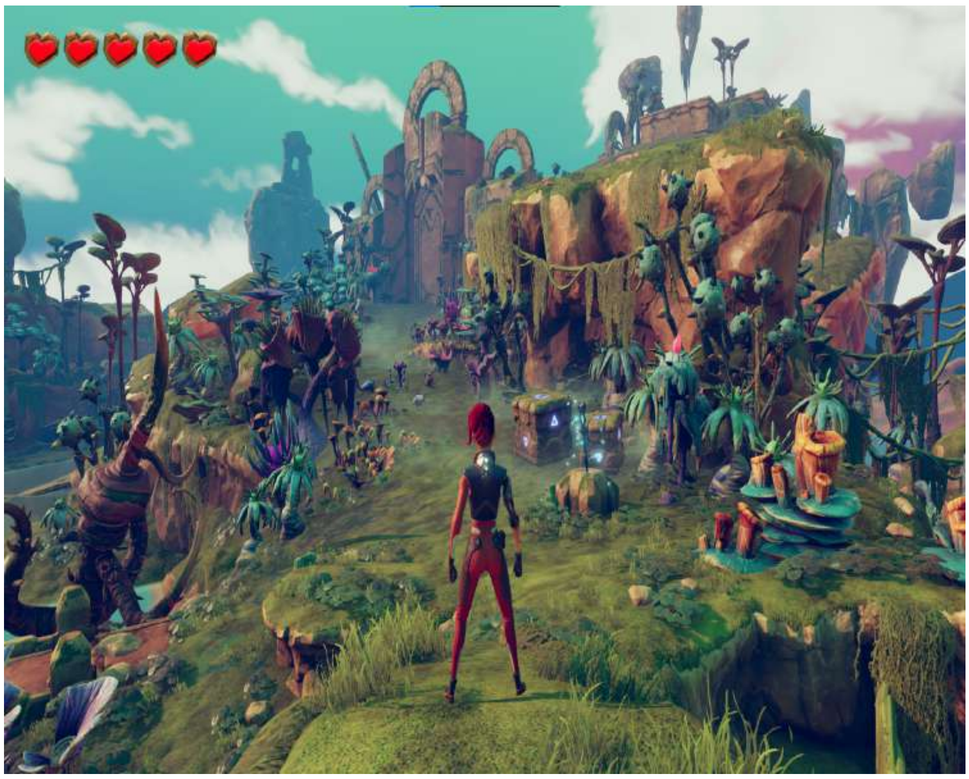


Figure 4.3: Gameplay

The enemies will detect our presence and they will start attack us. One successful attack will cost us one life point which will be indicated on the health bar. The enemy character will start chasing us if we try to run. This is done using the AI implementation on the enemy characters. There are three types of enemy characters in this game, Chombers, Spitters and Grenadiers. Once you are detected then they will start to follow you anywhere you go until u destroy them. This enemy AI is called hunting and is a famous implementation of AI in gaming. All enemies have their own strengths and weaknesses. This game has only a melee weapon, which look like a staff. By clicking right mouse button you can kill enemies with them. The staff is obtained while exploring the open world. Finally, after eliminating all the enemies and defeating the final bosses, our ship will come to us and the game is finished.

Milestones completed:

- Prototype build was completed successfully.
- A playable game with two level was completed successfully.
- UI was designed completely.

- AI implementations was incorporated to the enemies successfully.
- loot items are added successfully.

Chapter 5

Conclusions

The Game was tested by over twenty users, who all reported that they thoroughly enjoyed the game. It was observed that there was approximately equal number of players who were able to kill all the enemies. But for everyone the boss fights were very challenging and fun. This suggests that not only did the game provide an entertaining gaming experience, it also provided a reasonably engaging and awesom gameplay.

In general, it can be concluded that the Unity platform supported efficient development of the action-adventure Game. The Unity platform supports implementing the player's movement on the terrain with the keyboard and mouse , the physics engine, and vector calculation functions, all of which are not available if the implementation was done using traditional AI search techniques.

With these Unity components, We are able to implement the enemy's search for the player with less effort and more efficiently, and the prefab enemy characters can successfully imitate predator behaviours.

5.1 Future Enhancements

- Improved graphics
- Smoother movement and more responsive game controls
- Improved frame per second
- An entertaining story mode

- Improved and solid level design and world design
- More balanced challenge and reward
- More levels.
- New Weapons.
- New enemies
- Multiplayer facility for both online and offline gaming.
- Interactive objects and sub missions
- Cross Platform multiplayer
- Allies

References

[1] **Haas, John K**. ”A history of the unity game engine.” (2014) , [Online]. Available: `https://digital.wpi.edu/downloads/2f75r821k?locale=en`.

[2] **Canossa, Alessandro.**”Interview with Nicholas Francis and Thomas Hagen from Unity Technologies.” In-Game Analytics, pp. 137-142. Springer, London, 2013.

[3] **Becker-Asano, Christian, Felix Ruzzoli, Christoph Hölscher, and Bernhard Nebel.** ”A multi-agent system based on unity 4 for virtual perception and wayfinding.” Transportation Research Procedia 2: 452-455.

[4] **Goldstone, Will.**. Unity 3. x game development essentials. Packt Publishing Ltd, 2011.

[5] **Patil, Pratik P., and Ronald Alvares.**.”Cross-platform Application Development using Unity Game Engine.” Int. J 3, no. 4

[6] **Andrade, António.**””Game engines: a survey.” EAI Endorsed Transactions on Serious Games 2, no. 6

[7] **Borg, Markus, Vahid Garousi, Anas Mahmoud, Thomas Olsson, and Oskar Stalberg.**”Video Game Development in a Rush: A Survey of the Global Game Jam Participants.” IEEE Transactions on Games

[8] **Malete, Thabo N., Kabo Moruti, Tsaone Swaabow Thapelo, and Rodrigo S. Jamisola** a. ”EEG-based Control of a 3D Game Using 14-channel Emotiv Epoc+.” In 2019 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM), pp.463-468. IEEE,2019.

[9] **Unity Real-Time Development Platform**, Available: `https://unity.com/`.

[10] **Unity Learn**, Available: `https://learn.unity.com/`.

[11] **Unity Developer tools and Resources**, Available: `https://unity.com/developer-tools`.

[12] **Unity 2D 3D Game Creation Solutions Services**, Available: `https://unity.com/solutions`.

[13] **Unity Asset Store**, Available: `https://assetstore.unity.com/`.

[14] **C documentation**, Available: `https://docs.microsoft.com/en-us/dotnet/csharp/`.

[15] **Learn to code in Visual Studio**, Available: `https://visualstudio.microsoft.com/vs/getting-started/`.

[16] **Visual Studio documentation**, Available: `https://docs.microsoft.com/en-us/visualstudio/windows/?view=vs-2022`.

[17] **ProBuilder**, Available: `https://unity.com/features/probuilder`.

[18] **Polybrush**, Available: `https://unity.com/features/polybrush`.

Appendix

Source Code

Damageable.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
using Gamekit3D.Message;
using UnityEngine.Serialization;

namespace Gamekit3D
{
   public partial class Damageable : MonoBehaviour
   {

      public int maxHitPoints;
      [Tooltip("Time that this gameObject is invulnerable for, after receiving damage.")]
      public float invulnerabiltyTime;


      [Tooltip("The angle from the which that damageable is hitable. Always in the world XZ plane, with the forward being
           rotate by hitForwardRoation")]
      [Range(0.0f, 360.0f)]
      public float hitAngle = 360.0f;
      [Tooltip("Allow to rotate the world forward vector of the damageable used to define the hitAngle zone")]
      [Range(0.0f, 360.0f)]
      [FormerlySerializedAs("hitForwardRoation")] //SHAME!
      public float hitForwardRotation = 360.0f;

      public bool isInvulnerable { get; set; }
      public int currentHitPoints { get; private set; }

      public UnityEvent OnDeath, OnReceiveDamage, OnHitWhileInvulnerable, OnBecomeVulnerable, OnResetDamage;

      [Tooltip("When this gameObject is damaged, these other gameObjects are notified.")]
      [EnforceType(typeof(Message.IMessageReceiver))]
      public List<MonoBehaviour> onDamageMessageReceivers;

      protected float m_timeSinceLastHit = 0.0f;
      protected Collider m_Collider;

      System.Action schedule;

      void Start()
```

```
{
   ResetDamage();
   m_Collider = GetComponent<Collider>();
}

void Update()
{
   if (isInvulnerable)
   {
      m_timeSinceLastHit += Time.deltaTime;
      if (m_timeSinceLastHit > invulnerabiltyTime)
      {
         m_timeSinceLastHit = 0.0f;
         isInvulnerable = false;
         OnBecomeVulnerable.Invoke();
      }
   }
}

public void ResetDamage()
{
   currentHitPoints = maxHitPoints;
   isInvulnerable = false;
   m_timeSinceLastHit = 0.0f;
   OnResetDamage.Invoke();
}

public void SetColliderState(bool enabled)
{
   m_Collider.enabled = enabled;
}

public void ApplyDamage(DamageMessage data)
{
   if (currentHitPoints <= 0)
   {//ignore damage if already dead. TODO : may have to change that if we want to detect hit on death...
      return;
   }

   if (isInvulnerable)
   {
      OnHitWhileInvulnerable.Invoke();
      return;
   }

   Vector3 forward = transform.forward;
   forward = Quaternion.AngleAxis(hitForwardRotation, transform.up) * forward;

   //we project the direction to damager to the plane formed by the direction of damage
   Vector3 positionToDamager = data.damageSource - transform.position;
   positionToDamager -= transform.up * Vector3.Dot(transform.up, positionToDamager);

   if (Vector3.Angle(forward, positionToDamager) > hitAngle * 0.5f)
      return;

   isInvulnerable = true;
   currentHitPoints -= data.amount;

   if (currentHitPoints <= 0)
      schedule += OnDeath.Invoke; //This avoid race condition when objects kill each other.
   else
      OnReceiveDamage.Invoke();
```

```
            var messageType = currentHitPoints <= 0 ? MessageType.DEAD : MessageType.DAMAGED;

            for (var i = 0; i < onDamageMessageReceivers.Count; ++i)
            {
                var receiver = onDamageMessageReceivers[i] as IMessageReceiver;
                receiver.OnReceiveMessage(messageType, this, data);
            }
        }

        void LateUpdate()
        {
            if (schedule != null)
            {
                schedule();
                schedule = null;
            }
        }

#if UNITY_EDITOR
        private void OnDrawGizmosSelected()
        {
            Vector3 forward = transform.forward;
            forward = Quaternion.AngleAxis(hitForwardRotation, transform.up) * forward;

            if (Event.current.type == EventType.Repaint)
            {
                UnityEditor.Handles.color = Color.blue;
                UnityEditor.Handles.ArrowHandleCap(0, transform.position, Quaternion.LookRotation(forward), 1.0f,
                    EventType.Repaint);
            }


            UnityEditor.Handles.color = new Color(1.0f, 0.0f, 0.0f, 0.5f);
            forward = Quaternion.AngleAxis(-hitAngle * 0.5f, transform.up) * forward;
            UnityEditor.Handles.DrawSolidArc(transform.position, transform.up, forward, hitAngle, 1.0f);
        }
#endif
    }

}
```

Appendix

PlayerInput.cs

```
using UnityEngine;
using System;
using System.Collections;
using Gamekit3D;


public class PlayerInput : MonoBehaviour
{
   public static PlayerInput Instance
   {
      get { return s_Instance; }
   }

   protected static PlayerInput s_Instance;

   [HideInInspector]
   public bool playerControllerInputBlocked;

   protected Vector2 m_Movement;
   protected Vector2 m_Camera;
   protected bool m_Jump;
   protected bool m_Attack;
   protected bool m_Pause;
   protected bool m_ExternalInputBlocked;

   public Vector2 MoveInput
   {
      get
      {
         if(playerControllerInputBlocked || m_ExternalInputBlocked)
            return Vector2.zero;
         return m_Movement;
      }
   }

   public Vector2 CameraInput
   {
      get
      {
         if(playerControllerInputBlocked || m_ExternalInputBlocked)
            return Vector2.zero;
         return m_Camera;
      }
   }

   public bool JumpInput
   {
      get { return m_Jump && !playerControllerInputBlocked && !m_ExternalInputBlocked; }
   }

   public bool Attack
   {
      get { return m_Attack && !playerControllerInputBlocked && !m_ExternalInputBlocked; }
   }

   public bool Pause
   {
      get { return m_Pause; }
   }

   WaitForSeconds m_AttackInputWait;
   Coroutine m_AttackWaitCoroutine;

   const float k_AttackInputDuration = 0.03f;

   void Awake()
   {
      m_AttackInputWait = new WaitForSeconds(k_AttackInputDuration);

      if (s_Instance == null)
         s_Instance = this;
      else if (s_Instance != this)
         throw new UnityException("There cannot be more than one PlayerInput script. The instances are " + s_Instance.name
              + " and " + name + ".");
   }


   void Update()
   {
      m_Movement.Set(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"));
      m_Camera.Set(Input.GetAxis("Mouse X"), Input.GetAxis("Mouse Y"));
      m_Jump = Input.GetButton("Jump");

      if (Input.GetButtonDown("Fire1"))
      {
         if (m_AttackWaitCoroutine != null)
            StopCoroutine(m_AttackWaitCoroutine);

         m_AttackWaitCoroutine = StartCoroutine(AttackWait());
      }

      m_Pause = Input.GetButtonDown ("Pause");
   }

   IEnumerator AttackWait()
   {
      m_Attack = true;

      yield return m_AttackInputWait;

      m_Attack = false;
   }

   public bool HaveControl()
   {
      return !m_ExternalInputBlocked;
   }

   public void ReleaseControl()
   {
      m_ExternalInputBlocked = true;
   }

   public void GainControl()
   {
      m_ExternalInputBlocked = false;
   }
}
```