, H446 Coursework

Name: Ryan Roice
Candidate Number:
Centre Number:

Contents

Analysis

An outline of the problem:

In 2003, the first instalment of the Call of Duty franchise was developed by Infinity Ward and published by Activision. The game's great success inspired the first Call of Duty Zombies game, developed by Ideaworks Game Studio and also published by Activision. Since then there have been 7 new COD zombie's games that each build up on the game's storyline. I have decided to focus on the game that was released in November of 2015, Call of Duty: Black Ops III.

My project will be a three dimensional, first-person shooter survival game in which I aim to iterate upon the existing Call of Duty Zombies game. The game requires you to kill zombies that move towards you and attack. The player will be able to kill the zombies by shooting it with a gun until its health completely diminishes. Each kill will have a chance of an item(e.g a coin or some sort of currency) being dropped which will allow the player to upgrade their weapons and perks allowing them to kill zombies more easily. However, if the player is hit by a zombie, they will take damage and will result in a deduction of the player's health. Once the player's health reaches zero, the player dies and will respawn. However, if the player dies 3 times, the game is over. A certain number of enemies must be killed in each round. Furthermore, the game will get more difficult after every round and a point system will be implemented to track the progress of the player. I will also include co-op compatibility for the game.

How the project will be solved by computational methods:

Thinking Abstractly:

Abstracting a problem involves removing unnecessary details while focusing on the most important features for the problem. This will help me plan my project and efficiently work through it as I can identify what is and isn't necessary for the game.

Abstraction for the game:

- The game will be in first person which means that I would not need to add any character animations for sprinting, walking, carrying the gun, etc.
- I will avoid using complex mechanics that make the game too realistic as the result is not worth the effort and time it takes to produce, e.g screen shake, gun recoil, aiming down sight with the gun, blood splatters.
- I will only be using crucial sound effects that are not too complicated so that the game is more immersive, e.g gunshots, footsteps,

- The hitboxes for the game will be very simplistic involving basic shapes that are stacked upon each other as pixel perfect hitboxes are unnecessary and will make the game very demanding on the computer.
- Adding a game menu that allows the user to change in-game settings(e.g. game difficulty, adjusting game volume, etc.) and access other game options such as a shop or the inventory

Decomposition:

This involves breaking down the problem into smaller, modular components that are more manageable. Once these specific components are identified, they can be solved with relative ease, therefore simplifying the problem and making it efficient. This will be a salient part of the development process for my game as I will break down the overall game concept into its core elements such as game mechanics, characters and objectives. Since each of these components are isolated, testing and debugging errors in the code will also be much easier as I would recognise what part of the code is causing an issue quicker, hence making the whole process more efficient. Similarly, as I am using Unity to create my game, I will be having multiple scenes(each scene represents a distinct part of the game) that will all have to be put together. Decomposition will help me manage these scenes efficiently, allowing me to design and test them individually, enhancing the organisation of the whole development process.

Pattern Recognition:

This involves finding patterns among small, decomposed problems, which allow complex problems to be solved more easily. Due to the nature of my project, areas of code will constantly be repeated throughout the program. After identifying these repeated patterns, I will make efficient use of subroutines to replace the repetitive code. For example, both my character and enemy model will most likely have similar movement capabilities such as regular directional movement and also sprinting. Therefore, I will be able to reuse elements of code from my character model into my enemy model. This is much more efficient than having to rewrite code due to the time I would save. Furthermore, I will know that the code I reused is reliable and functional since it had worked previously.

Pipelining:

Pipelining is a crucial aspect of a game for many reasons, primarily related to optimising the performance and responsiveness of the game. This is because pipelining allows multiple sequential instructions to be performed concurrently. This means pipelining can be a useful computational method for accomplishing multiple things. For example, input processing can utilise pipelining to make sure that player inputs are quickly acted upon to prevent any noticeable lag or input delay.

Algorithmic thinking:
This is the final component of computational thinking. It is the process of defining a solution through step-by-step instructions. This is an important aspect to consider as I will need to take into account the efficiency, maintainability and readability of the code which I will write. Therefore using suitable variable and function names alongside descriptive comments for complicated parts of the program will immensely help in the readability of the code. This will allow possible future developers to quickly and effectively understand the code, leading to a much faster development. Furthermore, the modular nature of Object-Oriented programming allows other developers to develop systems more quickly and effectively as code is already known to be functional.

Conclusion
Computational thinking will be crucial when creating my game as it provides a structured and problem-solving oriented approach to development. The previous examples demonstrate how my game can be solved through different computational methods, therefore a computer program would be an appropriate course of action for efficiently solving the problem.

Stakeholders

Since my game is going to be used for general entertainment, I will have quite a wide range of ages for my target audience. However, due to the realism of the game and the complex controls required to play the game, I think that it will be quite unsuitable for younger audiences. Therefore an appropriate age range for my target audience would be between 16 - 40 years old.

In order for the game to be created to a good standard, I will need to have stakeholders who can guide me during the design and development processes for the game. The stakeholders will also be able to use the system extensively to critique the game's features or to report any major bugs/issues they find. This allows me to create a game that would be enjoyable and suitable for their needs.

Stakeholder 1

Name - Samuel Mandoza Age - 17 Gender - Male

I believe people like Samuel will be the main demographic for my target audience. These are generally teenagers who have substantial free time and are well interested and experienced in survival games. Samuel has been playing similar games for years and according to him, finding a game that is unique and brings that sense of exhilaration again is nearly impossible these days. Therefore, he will be my main companion during this project as I will keep regular contact with him and continue to inquire with him on how I should develop the game.

Stakeholder 2

Name- Donte O'Neill Age - 36 Gender - Male

Donte has keen interests in survival games and has had similar experiences from well known titles such as Call of Duty: World at War and Call of Duty: Black Ops II. He works full time as a financial advisor at a bank and has a family of 4(2 of which are kids of ages 3 and 9 years old). So he generally has a tight schedule and will only have a limited amount of time to play games. Therefore, I believe my game would appeal to him, as the concept of my game is very simple and does not require much previous knowledge. Furthermore, I would implement the ability to pause and save the game. This would be incredibly helpful for people who don't have a lot of time on their hands as they can save their hard-earned progress which will keep them motivated to play again.

Stakeholder 3

Name - Lexie Holt Age - 17 Gender - Female

Lexie has never really played survival games before. Although, she has played iconic games like Minecraft and Roblox which may contain some aspects of survival. She has never gone out of her way to enjoy the thrill and excitement of staying alive as long as possible. She also has a significant amount of free time, similar to Samuel, however she doesn't regularly play video games unless all her friends are playing. She generally enjoys watching TV shows on Netflix and regularly plays sports to pass time. This is because she thinks that games can get tiring and mundane very quickly. However, I believe that my game would still appeal to her as I will include co-op compatibility allowing her to play with her friends. Furthermore, I could also add a plot to the game and update the storyline regularly throughout the game to keep her entertained.

Stakeholder 4

Name - Maya Wilson Age - 28 Gender - Female

Maya loves indulging into survival games from time to time, however she struggles to find her passion for games recently as she spends most of her time working on her own games. This is because Maya is a game developer at a small indie company that generally produces puzzle/problem solving games. Similarly to Samuel, she has also played a handful of survival games and rarely finds a game that she can get hooked on. However, I believe that with her expertise in game design and development I would be able to create a thrilling survival game that keeps players hooked.

Research

Primary research:

Client Interview

I decided to have an interview with my client, Samuel Mandoza, as this will allow me to get a stronger insight into the desired features of the game and whether they are a necessity or not. I chose to do an interview over other primary data collection methods such as questionnaires, observations and focus groups as these research methods generally require large groups of people which I believe will be time consuming to find. Furthermore, they focus on a variety of different answers from people whereas my game has quite a niche target audience, which makes detailed, well informed answers more beneficial.

Interview plan:

There will be 4 main categories of questions for the game. These will include: What is the theme for the game? How realistic should the game be? Is a two-player co-op/multiplayer mode wanted? What are some features you would want in the game?

Interview Script:

What is the theme for the game?

What will the map for the game look like?
I want the game to have a space-like look to it. For example a starry sky with planets and galaxies as the background for the map. For the terrain of the map, maybe a large map with rocky/sandy hills. I would also like to have some extra features like a body of water, some sort of vegetation and buildings. Although, these will all have to match the aesthetics and theme of the map.

What would the name of the game be?
To match the theme of the game as a space shooter game, I think I will call the game “Galaxy Gunner” or perhaps “Space Extermination”. I think a simple name would be the most desirable as those tend to stick in people’s heads. I actually really like the way “Galaxy Gunner” sounds, so I will go with this being the name for the game.

How realistic should the game be?

Would you want your game to be 2D or 3D?

I would like the game to be 3 dimensional.

Do you think sound effects are necessary?
Yes I believe sound effects would really elevate the gaming experience which I think is necessary for a good game. For example footsteps, hostile enemy sounds, weapon sounds,etc alongside their respective visual animations and effects would definitely amplify the immersiveness of the game.

Do you think visual effects are necessary?
Yes I also believe that visual effects are necessary however, too many effects can be overwhelming and could possibly cause a lot of visual clutter which would ruin the gaming experience. I think the only visual effects required are ones that can provide information to the player. For example, when the player takes damage, the screen has a red tint for a few seconds, or when the player successfully hits the enemy a number can pop up above the enemy to represent the damage done.

What input would be used for the game?
I would like to use a keyboard and mouse to play the game as this is the most appropriate input for a computer. Even though the haptic vibrations from a controller can make the game more immersive, it would be inconvenient to use it with a computer.

Is a two-player co-op mode wanted?

Do you think a two player co-op mode is necessary?
Yes, I believe that playing with a friend makes the experience marginally better. Therefore, having the option to choose to play with a friend would be extremely entertaining.

How will the second player play the game?
I think that the second player can be a teammate to the first player allowing them to work together. This makes having another player beneficial as it would be easier to kill the enemies. I am personally not a big fan of split screen as it ruins the immersion of the game.

What are some features you would want in the game?

What features do you think would make the game interesting?
I think an emphasis on game mechanics and movement would make the game a lot more interesting. For example, being able to quickly dash away or having special abilities that can make killing enemies easier. Of course, these things would need cooldowns to prevent the game from becoming too easy. Although these features may be too hard to implement in the limited time available.

How will you reward the player for killing enemies?

The enemies could drop items that could be used at the shop to purchase other items or upgrade the player's statistics or weapons. This gives the player an incentive to keep playing as he will get better equipment which makes the game easier.

Can you expand on how the shop would function?
The shop will either be a physical location or a virtual interface within the game. The player would use their points/currency to make purchases. These purchases could be delivered to the player as airdrops from the sky. We can also make a fake spaceship that quickly flies over the map to imitate an actual airdrop.

Review of the interview:

This interview was carried out to get a better understanding of the initial requirements/features as well as the motive for the game. From this interview, I have understood that the game will have a strong space theme to it. I will need to take this into account when designing the game, especially the user interfaces and the map for the game.

The realism for the game also seems to be an important factor for the client. Although some aspects that affect the realism of the game such as animations and movement ability are important and can add a lot to the game, things like sound and visual effects would not add much to the functionality of the game. Therefore, these things would have a lower priority when developing the game.

The client would also like to have some sort of multiplayer feature, allowing the player to play with someone else as a teammate. I believe this would be quite challenging to implement, especially understanding the network behind it.

Some of the extra features such as extra movement techniques and special abilities would be a great addition to the game; I believe that it can make the game a lot more unique. Furthermore, the shop feature can open a lot of possibilities when considering the future of the game. It will provide much more progression and engagement, as well as possible monetization of the game.

Secondary Research:

I am going to be focusing on the zombies gamemode from Call of Duty. The main objective of this game is to survive as long as possible against the zombies.

Call Of Duty: Zombies

Initially, Zombies wasn't meant to be a featured game mode in the Call of Duty franchise. It was just a secret mini-game with only a single map called 'Nacht der Untoten'. This was part of the game Call of Duty: World at War which was released in November 2008. At this point, there were no dedicated characters or storylines, however, due to the unprecedented success of

Zombies, Treyarch included a new Zombies map as part of a DLC pack, called “Verruckt”. This was the first appearance of a core character known as Tank Dempsey. As time passed, two more additional maps were brought to World at War Zombies, which were great fan favourites. The game had then become more advanced, with more features and mechanics, and the storyline for the game was finally being built up at this time.

The first game of the Black Ops series was released in 2010. This was game changing. There were 7 new maps that were added, multiple new weapons and different types of zombies with just the release of Black Ops 1. There were 3 more Black Ops games that were all huge successes, each game developing the storyline further providing heaps of content for the players.

Within the game, zombies will spawn around the map and can damage the player through melee attacks. The player is required to shoot the zombies and upon killing, the player is rewarded points for each zombie. These points can be used for a variety of different things, such as purchasing new weapons and ammo, upgrading their weapons, purchasing perks, opening up additional areas on the map, etc.

After a certain number of zombies are killed, the player progresses to the next round. The only difference between rounds is that waves of zombies are more frequent and the zombies also increase in health. This makes the game progressively more difficult and as there is no limit to what round you can reach, rounds can get incredibly difficult to survive. However, this game allows you to play cooperatively with up to 3 other friends to ease the difficulty and to also make the game more enjoyable.

The game also contains impressive visual graphics and realism with their assets. However, this greatly contributes to the staggering system storage requirements of nearly 100 gigabytes, which is absurd. The game is also very complicated to play due to the multitude of easter eggs and quests which require tedious steps and precise execution to do. However new players that don’t have much experience in the COD zombies franchise can still play competently as the main objective of the game is quite simple.

A number will pop up briefly when the player has gained a certain amount of points.

The total ammunition and the bullets in the gun are displayed separately here.

The player's score alongside their avatar is displayed here.

Any active bonuses are displayed at the bottom of the screen

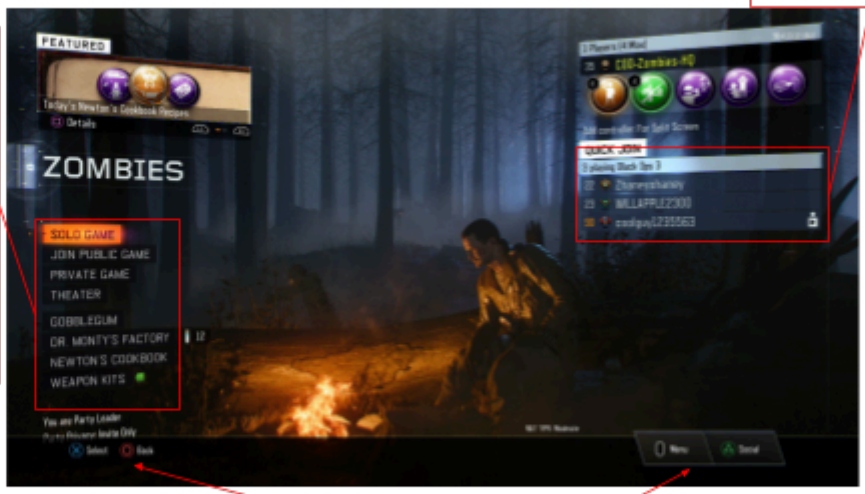
The game informs the player when new controls are introduced.

A timer is creatively displayed for when the player returns to their normal state.

A green tint visual effect is shown to tell the player they are in a different state.

Different players in the same game session are clearly displayed here.

Main menu contains many different options that are clearly labelled for the player. All the options are also stacked together on this side of the screen



Some options require specific buttons on the input device to access, which are clearly shown.

Power-ups dropped by zombies have an aura alongside an icon denoting its ability.

The first 5 rounds are counted and displayed as tally marks

Any existing perk purchased by the player is displayed here with a unique icon and timer.

Vampire Survivors:

Vampire Survivors was developed by the Italian developer Luca “poncle” Galante in 2020 using the Phaser Framework. The inspiration behind Luca’s successful game was Magic Survival, a mobile game that also consisted of a character automatically attacking enemies. It took him nearly a year to develop the first functional iteration of the game; he then proceeded to release it in early access on a website called itch.io. However the game didn’t receive much engagement. As a result, he decided to release the game on Steam - a well known game distribution service - by the end of 2021. Thanks to the influence of well known Youtubers, his game had seen an exponential rise in players due to it being discovered by more people.

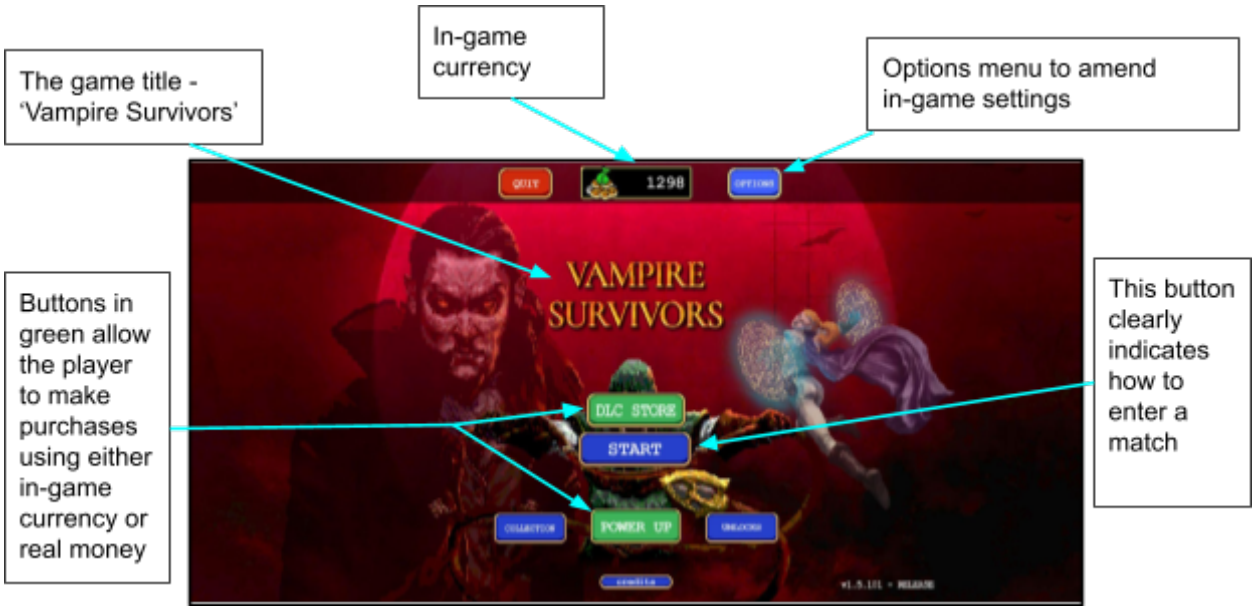
In the end Luca had won multiple awards for his game and had his all-time peak concurrent players of 77,061 in the February of 2022. Due to the success of the game, he had expanded it

to multiple other platforms including Nintendo switch, Xbox and mobile. As of September 2023, the game's peak players is 17,446 and is slightly declining. However, Vampire Survivors have a strong community of players that relish the game which was Luca's main objective when he started developing the game.

Vampire survivors is a 2D top-down survival game that has a retro-style. The main aim of the game is similar to COD zombies - to survive as long as possible. However, this game has a stronger emphasis on upgradable perks and abilities. After every level, the player is granted a choice from 3 perks which will aid them in their survival. These perks get progressively stronger as the rounds pass. The game also has different characters that each have their own unique abilities. The multiple combinations of perks and characters allows this game to have many different play styles.

The game has a multitude of different enemy monsters, each with varying health and strength. The enemies will also attack the player when they are within range of them which will cause the player's health to decrease. Killing an enemy will drop an experience gem and after a certain number of gems, the player will reach the next level. Stronger enemies will drop gems that are worth more experience points.

However the game is only single-player, preventing you from playing with friends. This makes the game less enjoyable and also more difficult to play. The game also has very limited interaction with the user, as they cannot control when they can attack as all attacks are automated. This makes the game unstimulating as the player has less control over what they can do.



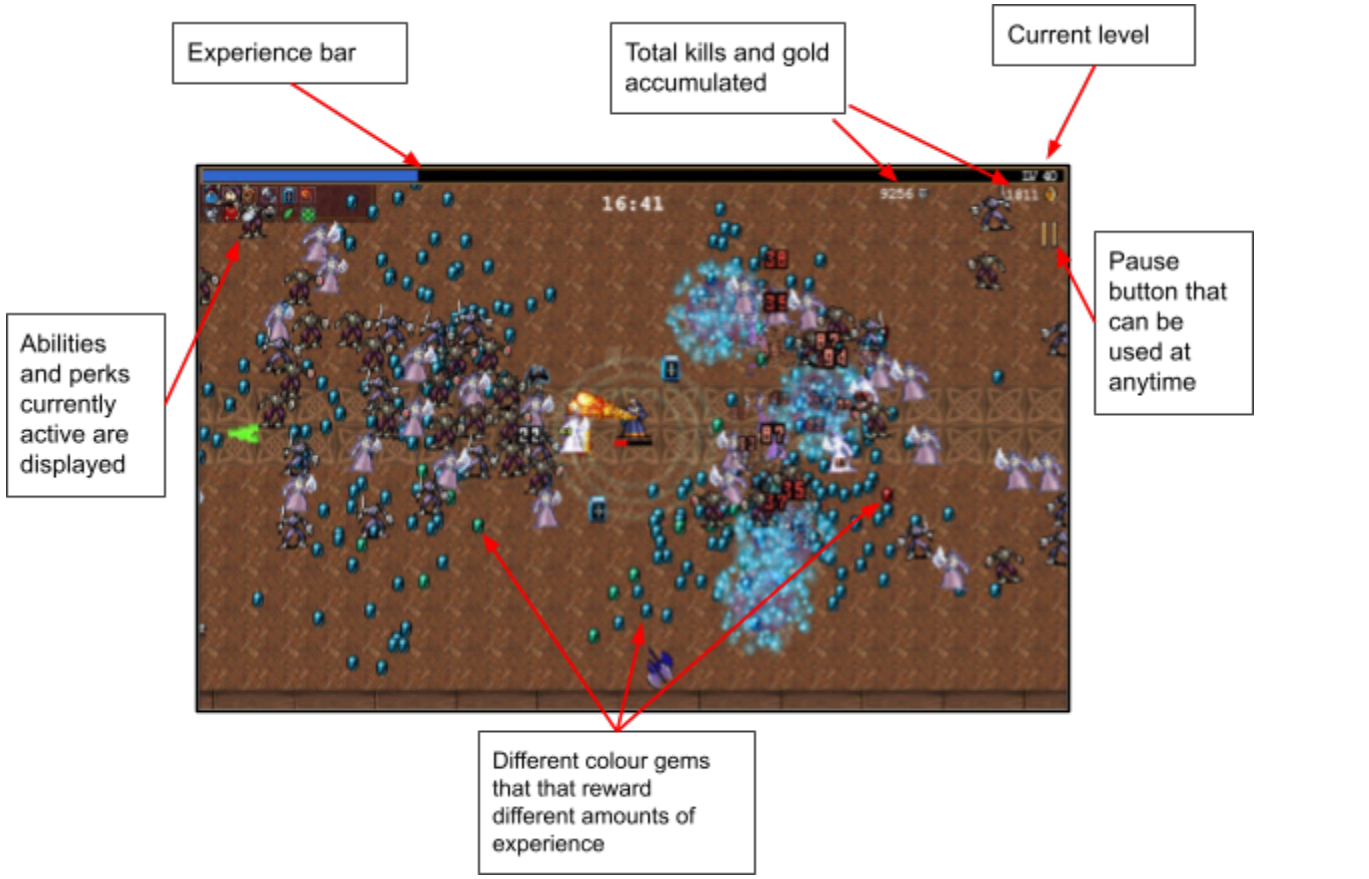
Different characters can be selected once they are unlocked with gold

The starting statistics of the selected character

Brief description of the selected character, stating their unique abilities and attributes.


Total time survived
The player is highlighted red when taken damage from an enemy
The health bar will decrease upon any damage taken
Large variety and quantity of enemies late in the game


There are multiple maps available, each with their own in-depth description.
Each map has their own unique properties displayed here
More maps get unlocked the further you progress into the game
Stage Selection


Experience bar
Total kills and gold accumulated
Current level
16:41
Abilities and perks currently active are displayed
Pause button that can be used at anytime
Different colour gems that that reward different amounts of experience


After each level the player is given a choice from 3 abilities/perks
Level Up!
Current statistics of the player

Once the player loses all their health, a large 'GAME OVER' text appears

Quit button to leave the match and return to main menu

The results screen gives the player a summary of their overall performance from their most recent match.

The fundamental game statistics are displayed here.

This displays the number of achievements unlocked and also describes each achievement.

The Forest:

The Forest is a survival horror video game developed and published by Endnight Games. It was initially released in 2018 and is the prequel to the game “Sons of the Forest”.

The game involves surviving on a remote island with the main objective of finding a way back home. However, the island is inhabited by hostile mutants and deadly animals which pose a constant threat to the player. The game is also extremely realistic (other than the malformed people that run around erratically) as it contains many naturalistic animations and mechanics. For example, players will need to be wary of their hunger, thirst, energy levels, illnesses and injuries, etc. This realistic approach makes the game highly immersive.

To survive, players must explore their surroundings, gather resources, and construct shelter, weapons, and tools. Crafting also plays a significant role in the game, allowing players to create weapons for defense, traps for hunting, and structures for shelter. This again adds to the realism of the game.

The enemies in The Forest are extremely aggressive and come in various forms, ranging from cannibals to grotesque mutants each with their own unique abilities. This adds an element of unpredictability and danger to the game, making the game more exhilarating. Players will have to fend off against these enemies while exploring through impenetrable forests and caves with deep underground tunnels, all while unravelling the strange mysteries surrounding them. There is a story-driven aspect to the game, with a series of notes, tapes, and visual cues that provide context and narrative progression.

The Forest can be played in both single-player and multiplayer modes. In multiplayer, players can collaborate with friends to survive together, share resources, and tackle the challenges of the island as a team. Multiplayer adds a cooperative element to the gameplay. However, this can also introduce more challenges as the number and strength of enemies increase based on the number of players. The forest also has a save game feature, however unlike usual games where the player may have to access a user interface to save their game, the forest requires the player to build a shelter. This means that the player will have to go out to scavenge resources to be able to craft a shelter to save their game progress. This is a very interesting and unique method to implement a great feature.

Clear blood indicator on the whole screen to quickly notify the player that they have taken damage.

The compact player stats HUD here is used to show the energy(dark blue arc), stamina (light blue arc), Health(red arc), as well as the thirst (water drop) and hunger(stomach) of the player.

The game provides instructions to the player in critical conditions. For example, the player is currently hungry, here the game is advising the player to find something to eat.

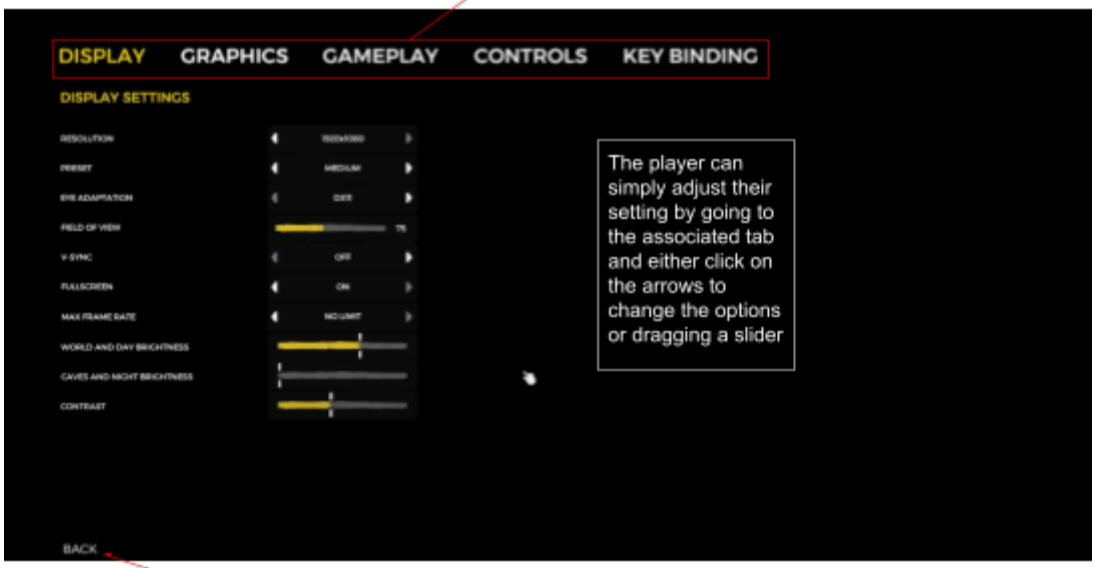
Items in the player's inventory are scattered across the ground and can be individually picked up

An icon for a cog becomes filled the more items the player adds from the correct recipe. Full filled cog mean the item can be crafted

Hovering over any items provide a brief explanation of what they are. For example we can see that the item highlighted is a cloth and is a crafting ingredient.

Items that have been picked up from the edges of the screen are brought to the middle mat to tbe crafted. There is also an option to remove all items placed on the crafting mat.

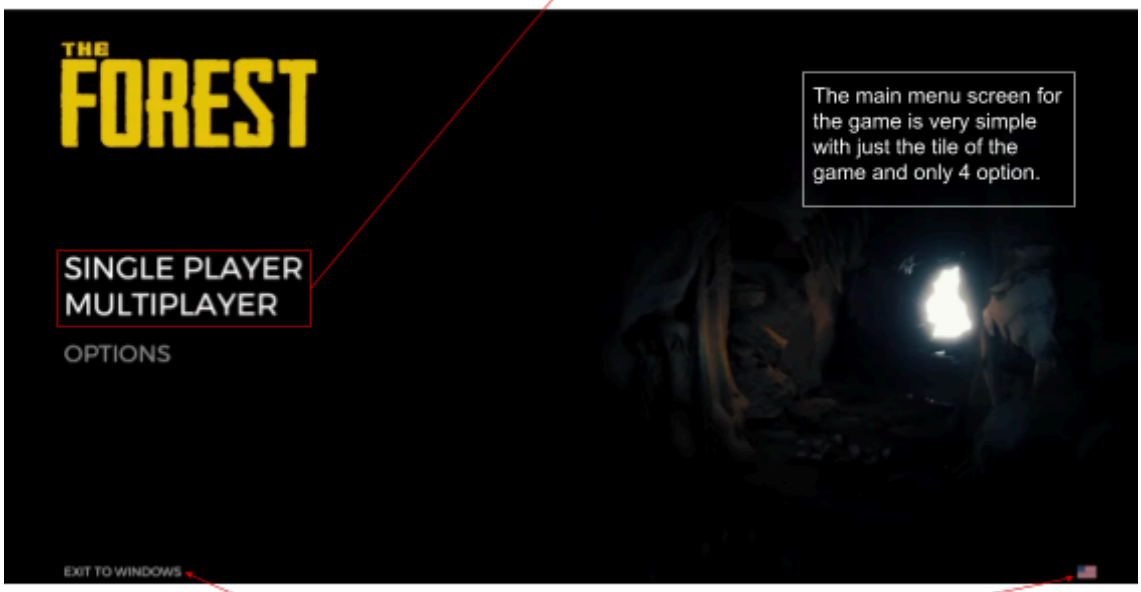
The different categories of the settings are clearly labelled and separated as tabs



The player can simply adjust their setting by going to the associated tab and either click on the arrows to change the options or dragging a slider

A back button is also displayed allowing the player to return to the previous screen they were on

The single player and multiplayer options are displayed in bigger and brighter text as this is what the player will mainly be clicking on.

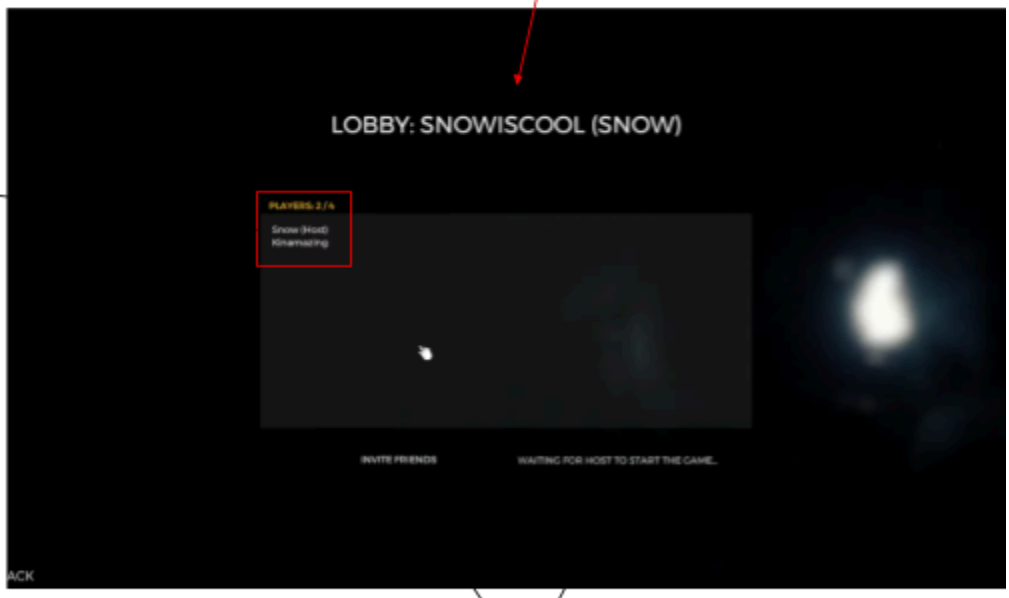


The main menu screen for the game is very simple with just the tile of the game and only 4 option.

An "exit to windows" button and the server location are displayed in small at the bottom of the screen

The name of the lobby is clearly displayed at the top

The total player out of 4 are displayed as well as each of their usernames



There are two options at the bottom of the screen. The player can invite friends or the ycan ready up, in this case they are already readied up and is waiting for the host.

The loading screen for the game is very minimalistic

There are also many hints that are cycled through while the game is loading. This can help guide the player once they are in the game.

The "The Forest" logo gets slowly filled with yellow, the same as the accent in the rest of the game.

Secondary research review:

In conclusion, these 3 games have provided insightful information about key features in successful games. This will allow me to make informed decisions about what features I should implement within my project. For example, the co-op compatibility of Call of Duty makes the game much more enjoyable as people can share their experience with their friends. The simplicity of vampire survivors allows for a wider target audience of players and a more relaxing gameplay experience. The Forest's thrilling story-telling and attention to detail lead to a very realistic and engaging gameplay experience. These are only some of the many features which I will be excited to possibly include within my game.

Features

Essential Features:

No.	Features	Justification
1	A menu	This allows the player to choose which mode they would like to play(either single player or co-op mode). The setting can also be accessed and changed at this time.
2	Sound effects	It makes the game more unique and exciting. It can also be helpful as it informs the player when anything happens during the game.
3	Player has health bar	This makes the game challenging as the player loses if their health completely depletes. It also gives the player a visual indicator of how much health they have left.
4	Player has multiple lives	This provides the user the ability to respawn a limited amount of times which would make the game much less stressful compared to only having one life.
5	The game must have an ammo counter	It gives a visual indicator of how much ammunition the player has left.
6	A shop	The player can make purchases using items they obtain within the game to give them advantages such as upgraded weapons, more health, perks, etc.
7	Score Leaderboard	Gives the player a reason to keep playing the

		game so that they can beat their high score.
8	Enemies get stronger as rounds pass	This is necessary to make the game more challenging and less repetitive.
9	A “Game Over” screen	This is a clear indicator to the player to tell them that their game has ended.
10	Damage indicator numbers	This helps the player clearly recognise when he has hit the enemy, it also informs them of the damage they have done.
11	Co-op mode	This makes the game more enjoyable as it allows friends to play together as teammates.
12	A Crosshair	This allows the player to accurately aim at the enemy.

Limitations:

Time Limitations			
Requirement No.	Requirements	Justification	Why is this a limitation?
R1	The gun will have recoil	Even though it adds realism to the game, it will be difficult to code and isn’t necessary for the game to function.	Due to the many visual and mechanical effects required, it will be difficult to implement within the time limit.
R2	Different parts of the game will have specific music tracks	This would provide a better atmosphere in the game. It can also create different emotions for the player allowing him to be more immersed into the game.	It will be time consuming to produce my own music especially since I am not familiar with any music production software.
R3	The player will have multiple extra movement abilities and techniques such as sliding and grappling.	These could improve the enjoyability of the game as the player is more in control, however it will be extremely hard to implement within the time limits I am given.	It will be difficult to implement all these game mechanics smoothly within such a short period of time. I also am developing the game on my own and have to spend time on my other A-Level courses.
Hardware Limitations			

Requirement No.	Requirements	Justification	Why is this a limitation?
R1	The player will be able to play the game online with other people.	This makes the game more interesting as the player can interact and play with new people.	I would need to have access to a server to facilitate this feature. I am unlikely to be able to do this.
Software Limitations			
Requirement No.	Requirements	Justification	Why is this a limitation?
R1	The game will have many realistic and detailed graphics and animations	This would make the game more immersive and also attract more people to play the game.	This would require software such as “Autodesk Maya” and “3Ds Max” which I don’t have access to.

Software requirements to make the game:

Requirement No.	Requirements	Justification
R1	Operating System Version (Windows 7 (SP1+), Windows 10 and Windows 11, 64-bit versions only)	These are the minimum hardware requirements needed to create the game on Unity.
R2	Unity game engine	Unity is an ideal game engine due to the wide variety of prebuilt assets which will greatly save time.
R3	VisualStudio Code IDE	This is the standard IDE used to run C# on Unity.

Software requirements to play the game:

Requirement No.	Requirements	Justification
R1	Windows or mac OS	A well developed operating system will be required to interact with the hardware required for the game.

R2	Graphics Card drivers	This is necessary for the graphics card to run optimally
R3	Keyboard drivers	This is necessary for the user's keyboard to work properly in the game
R4	Mouse drivers	This is necessary for the user's keyboard to work properly in the game
R5	DirectX (for Windows) or OpenGL(for macOS or linux)	This required for the unity to render the game's graphics properly
R6	Sufficient storage space	This is required for the game to be downloaded, otherwise the game can't be played

Hardware requirements to make the game:

Requirement No.	Requirements	Justification
R1	CPU (X64 architecture with SSE2 instruction set support)	These are the minimum hardware requirements needed to create the game on Unity.
R2	Graphics API (DX10, DX11, and DX12-capable GPUs)	These are the minimum hardware requirements needed to create the game on Unity.
R3	Hardware vendor officially supported drivers	These are the minimum hardware requirements needed to create the game on Unity.

Hardware requirements to play the game:

Requirement No.	Requirement	Justification
R1	Monitor	This is required to display the game to the player.
R2	Mouse	This allows the player to interact with the game by clicking. It also allows them to change the direction they are facing by moving the mouse.

R3	Keyboard	This allows the player to input the controls of the game by pressing keys.
R4	Headphones/Speakers	This outputs the game audio such as sound effects
R5	4 GB of RAM	This is the minimum amount of memory required to play this game.
R6	Intel Core i5 processor	This is the minimum spec processor required to run the game.
R8	NVIDIA® GeForce® GTX 470 Graphics card	This is the minimum spec graphics card required to render and play the game.

Success Criteria:

Criteria No.	Success Criteria	Justification	☑	Reference
1	Display the Main menu and settings for audio and visual effects.	This allows the player to start the game at his own accord. The settings for audio and visual effects allows the player to make changes to make the game more enjoyable and personal.	□	Vampire Survivors
2	Display the current score in the game.	The current score tells the player how well they are doing in the game. The all-time high score tells the player his best score at the game and gives him a goal - to beat his own high score.	□	Vampire survivors
3	Display a leaderboard with the different scores achieved by different people.	The leaderboard tells the player his best score at the game and gives him a goal - to beat his own high score.	□	Vampire Survivors
4	Display the health bar of the players and enemies	This allows the player to keep track of how much health he has until he dies. The enemies health bar tells	□	COD Zombies

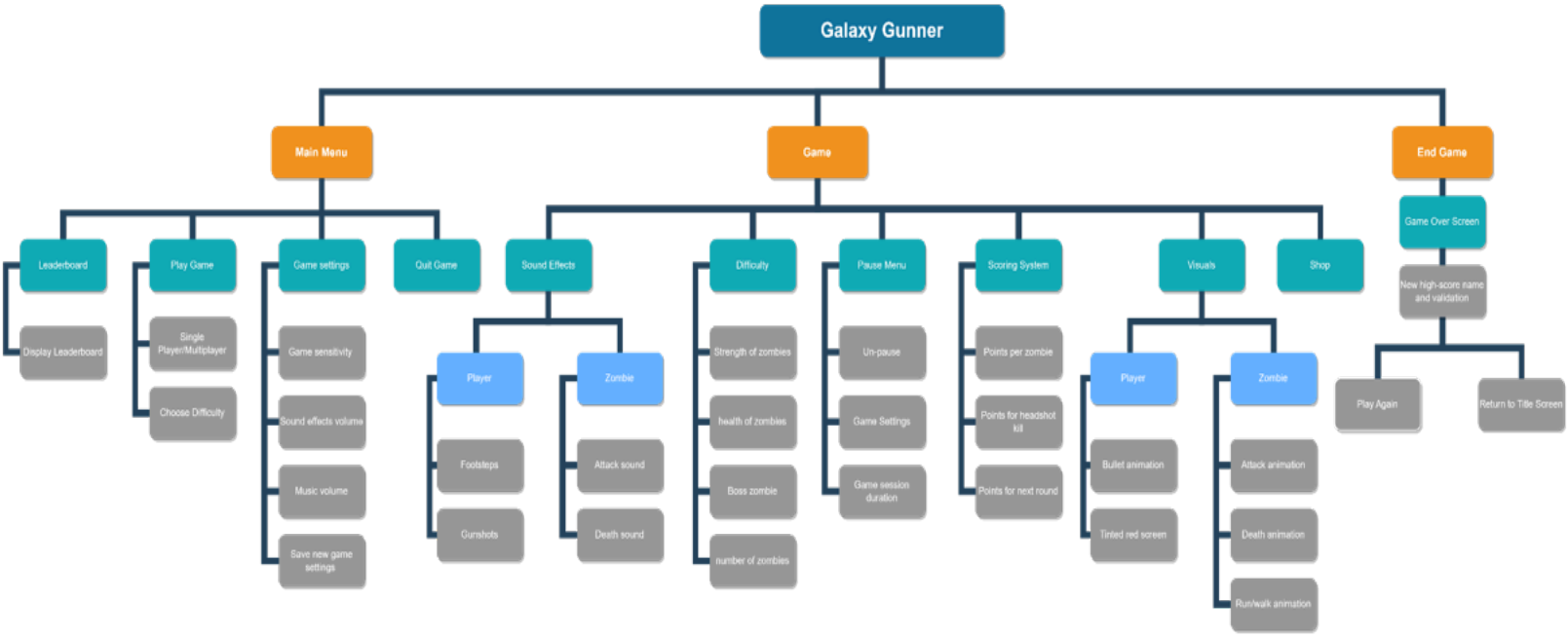
		the player how much more damage is required to kill it.		
5	A terrain for the game	This allows the player and enemies to move around properly. It also adds a lot to the game compared to an empty plane being used for the ground	□	COD zombies
6	The player can look around freely	This allows the player to be aware of their surroundings	□	The Forest
7	Player can move in all 4 directions as well as jump upwards	This allows the player to move away from enemies and to traverse the map. This feature is absolutely necessary for the game.	□	COD Zombies
8	Player can sprint to move faster	This allows the player to move faster in dangerous situations to avoid dying	□	The Forest
9	The player can shoot their gun at the enemy	This allows the player to defend themselves from the enemies and is a key element for the player's survival	□	COD Zombies
10	The player deals more damage with their weapon if they hit the zombie in the head.	This allows the player to get rewarded if they are skilled enough to hit the head of the zombie. It also makes the game more realistic as the head is more vulnerable than the rest of the body.	□	COD Zombies
11	The player's score increases by 10 points when an enemy is killed	This is done to reward the player for killing the enemy, the points will then be added onto their score	□	COD Zombies
12	The player starts with 100 health	This causes the player to be wary of getting attacked by an enemy as once they lose all their health, they will lose a life.	□	COD Zombies
13	The player has 3 lives	This will give the user the ability to respawn after dying a maximum of 3 times. This makes the game more forgiving as a single life will be too difficult.	□	
14	Display the number of lives the player has.	The player will need to know how many lives they have left before	□	

		the game is over. Icons can effectively be used to display this information.		
15	The player is given spawn protection upon respawning.	This is important as the player will just die again if they are immediately swarmed and attacked by the remaining zombies from when they previously died, which can be extremely frustrating.	□	COD Zombies
16	Attacking animation for the enemy	This will make the game more realistic and immersive which leads to a better gaming experience.	□	Interview
17	Dying animation for the enemy	This allows the player to recognise when an enemy has died. It also makes the game realistic	□	Interview
18	Chasing animation for the enemy	This will make the game more realistic and immersive which leads to a better gaming experience.	□	Interview
19	Sound effect for gunshots	This will make the game more realistic and immersive which leads to a better gaming experience.	□	COD Zombies
20	Increasingly larger waves of enemies are spawned randomly across the map.	This makes the game challenging as the player has to fight more enemies. It also makes the game more interesting as the user wouldn't be able to predict where the enemies will spawn from.	□	COD Zombies
21	The enemy can attack and kill the player	This is necessary for a survival game as the player must avoid taking too much damage from the enemy as they would die otherwise	□	The Forest
22	Spawn a Boss enemy with extreme health and damage every 5 rounds	This makes the game more unique and less repetitive.	□	Vampire survivors
23	Ability to pause the game	This is done for the convenience of the player as they can stop playing the game without endangering themselves.	□	Vampire Survivors
24	Ability to save the game	This allows the player to stop playing the game and retain his	□	The Forest

		progress for the next time he plays.		
25	Player can use the in-game shop to upgrade weapon and physical attributes	This gives the player something to work towards as they will be able to get stronger weapons. It also provides more personalisation and engagement.	□	COD Zombies
26	Display a game over screen when the game ends	This is a clear indicator for the player that their game is over. The player can either play again or return to the main menu after this.	□	Vampire Survivors
27	The screen will have a red tint for 3 seconds after the player takes damage.	This visual effect would be extremely beneficial for the player to realise they are taking damage. This is much better than having to constantly look at the health bar.	□	Interview
28	The player regenerates their health after not taking damage for 10 seconds	This means that the player can recover from a predicament if they are successful. Therefore, the player is more likely to continue playing.	□	COD Zombies
29	The zombie despawns 5 seconds after it falls to the ground once killed.	This allows the game to perform better as the dead zombie models wouldn't be occupying space in memory	□	The Forest
30	The player can play with another person as a teammate.	This makes the game more enjoyable as the players can share the experience they had	□	COD Zombies

Design

Decomposition Diagram:



Why have I taken this approach to help design my problem?

I have used a decomposition diagram to abstractly display all the integral mechanics and systems of the game. This allows me to break down the game into smaller modules that I can

individually focus on. It will also act as a guide while I programme the game, allowing me to work through the game efficiently and in the correct order.

Explanation of each module

Main menu:

Leaderboard

Display Leaderboard - This option will allow the player to view the leaderboard with each name and their respective score achieved.

Play Game

Choose Difficulty - This allows the player to choose between the easy, normal and hard difficulties before starting the game.

Game settings

Game sensitivity - This allows the player to adjust the sensitivity of the mouse they are using. Once a new sensitivity is saved by the player, the variable for the sensitivity will be updated.

Sound effects volume - The player can change the volume of the sound effects in the game to suit their liking. This is important as each person has their own preference as to how loud the sound effects should be.

Music volume - Similarly to sound effects, each person would like to change the volume of the music to their liking. However, unlike sound effects, music doesn't provide much benefit to the performance of the person playing.

Save new game settings - This is necessary so that the data for the new settings the player has applied is actually saved in the respective game variables.

Quit game

Game:

Sound Effects

Player

Footsteps - Whenever the player moves around or jumps a footstep sound will be played.

Gunshots - When the player clicks or holds the left click on the mouse, a gunshot sound will be played for every bullet shot.

Zombie

Attack sound - When the zombie is within range to attack the player a grunt sound is played alongside an attacking animation.

Death sound - When the zombies has lost all of its health points it will let out a howl sound alongside the death animation.

Difficulty

Strength of zombies - The starting value of the variable that holds the damage inflicted by the zombies is either increased or decreased based on the difficulty chosen by the player. This variable also increases by 2 for every round that the player has passed.

Health of zombies - The starting value of the variable that holds the health of the zombies is either increased or decreased based on the difficulty the player has chosen. This variable also increases by 10 for every round the player has passed.

Boss zombie - Every 5 rounds the player passes, there will be a boss zombie that spawns. These will be much bigger and stronger than a regular zombie. However it will also be slower than a regular zombie.

Number of zombies -The starting value of the variable for the number of zombies stays the same no matter what difficulty is chosen. However, the variable increases by 2 for every round the player has passed.

Pause Menu

Un-pause - When the player presses the “esc” key, the game will change from the paused state to the playing state.

Game Settings - The player will be able to change the settings for his game(sensitivity, audio) in the paused state. These settings will be applied once the player returns back to the playing state.

Game session duration - This displays how long the player has played for.

Scoring system

Points per zombie - For each kill on a zombie, 10 points are added to the variable holding the player's score. This will change further down the game as zombies will be worth more points.

Points for headshot kill - The player gets extra points when they kill a zombie with a headshot. The extra points are determined by multiplying the points per zombie by 1.25. Any decimal number is rounded up to the nearest whole number.

Points for each round - The player must achieve a certain number of points per round to advance to the next round. The amount of points to get to the next round equates to the total number of zombies spawned for that round multiplied by the points gained for each zombie killed. Therefore the player will have to kill all the zombies to advance to the next round.

Visuals

Player

Bullet visual effect - When the player left clicks a visual effect will be added to represent the travel of the projectile.

Tinted red screen - Whenever the player's health decreases , a light red tint will be added to the screen.

Zombie

Attack animation - When the zombie is in the attacking state, an animation will be played alongside the respective sound effect. If the player is in contact with the zombie's animation he will take damage.

Death animation - When the zombie's health is completely depleted, an animation of the zombie falling will be played alongside the respective sound effect. The zombie will then despawn from the map after 5 seconds.

Run and walk animation - When the zombie is in the walk state the animation of the zombie walking will be continuously looped. The zombie will only enter the run state once aggravated by the player.

Shop

This will be a physical shop on the map within the game. The player will be able to make transactions using their score from killing zombies. They will also be able to upgrade their equipment such as their weapons and even their physical attributes for example their speed, overall strength, vigour or endurance.

End Game:

Game over screen

When the player's health depletes to zero and has no lives left the game is over. The game will switch from the playing state to the game over state. This is indicated with a clear "GAME OVER" text displayed on the screen.

New high-score name and validation - This checks if the player's final score of the game that just ended is higher than the lowest score on the leaderboard. If it is, then it checks whether a suitable name is entered and then the score with the respective name is added to the leaderboard.

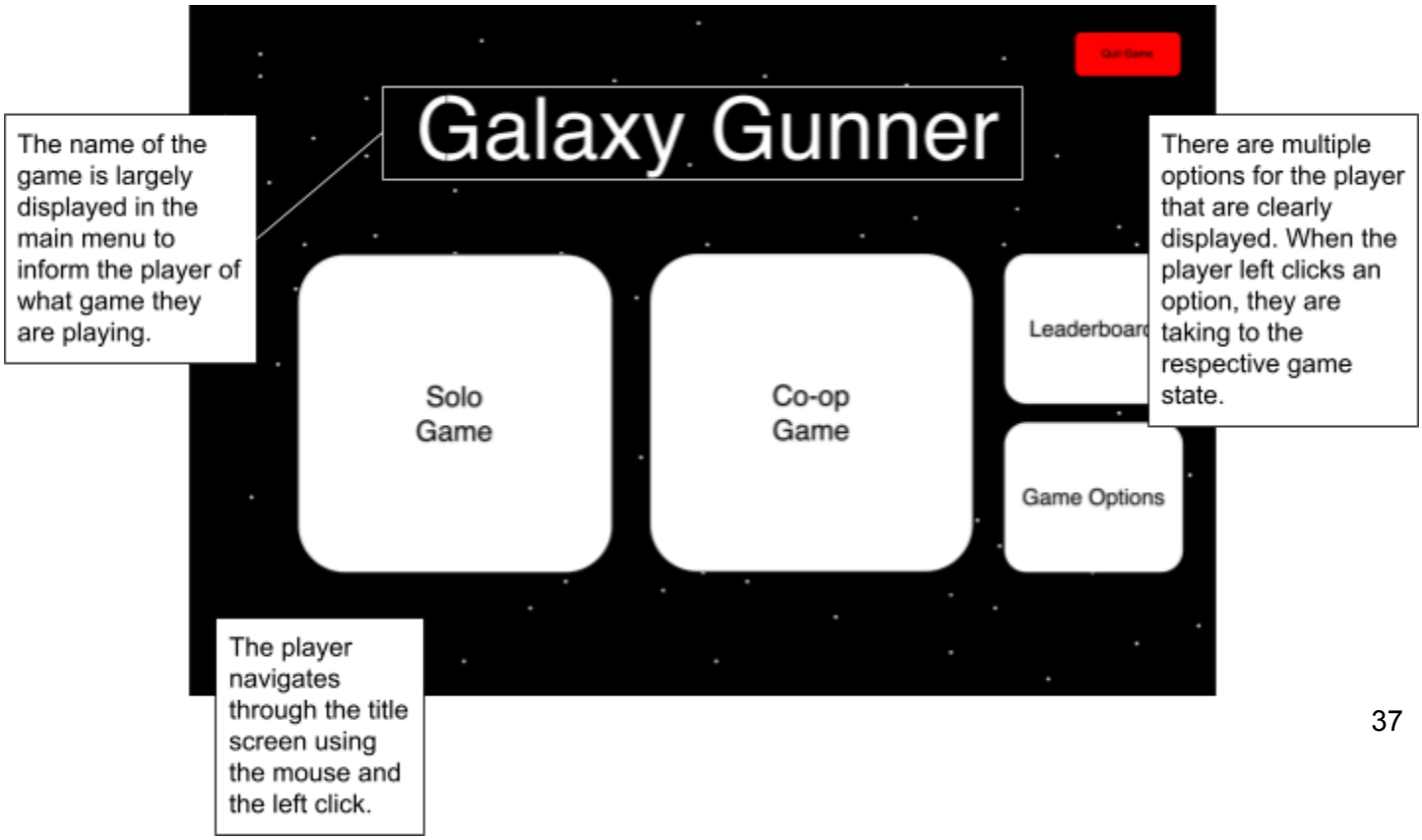
Play again - When the player clicks this button the game will restart with the same settings and difficulty. Therefore the game will return to the playing state from the game over state.

Return to Main menu- When the player clicks this button he will be directed back to the Main menu of the game and the game state will switch from the game over state to main menu state.

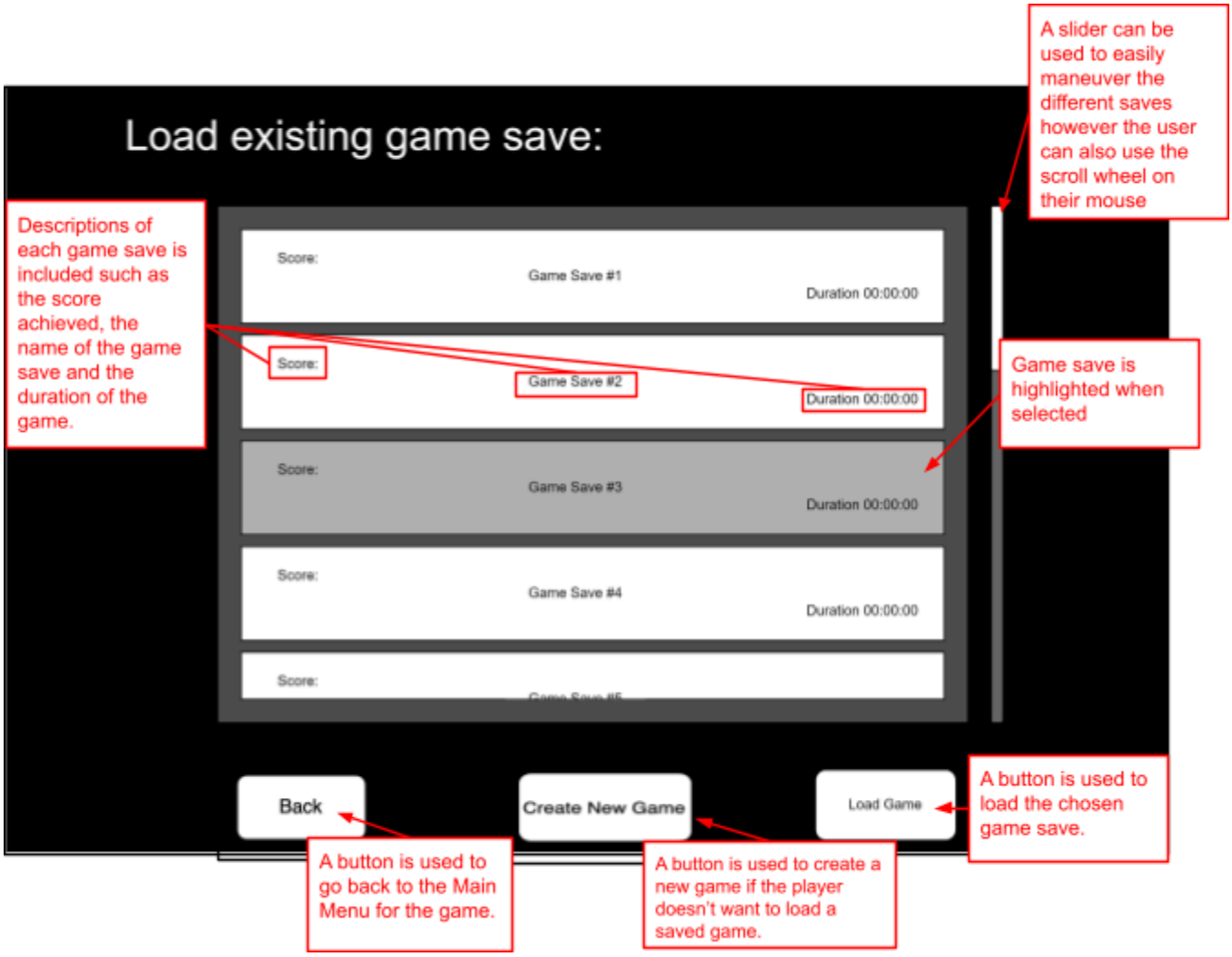
Interface Designs

In this section, I will present a vague design of the different user interfaces that would be incorporated into the game.

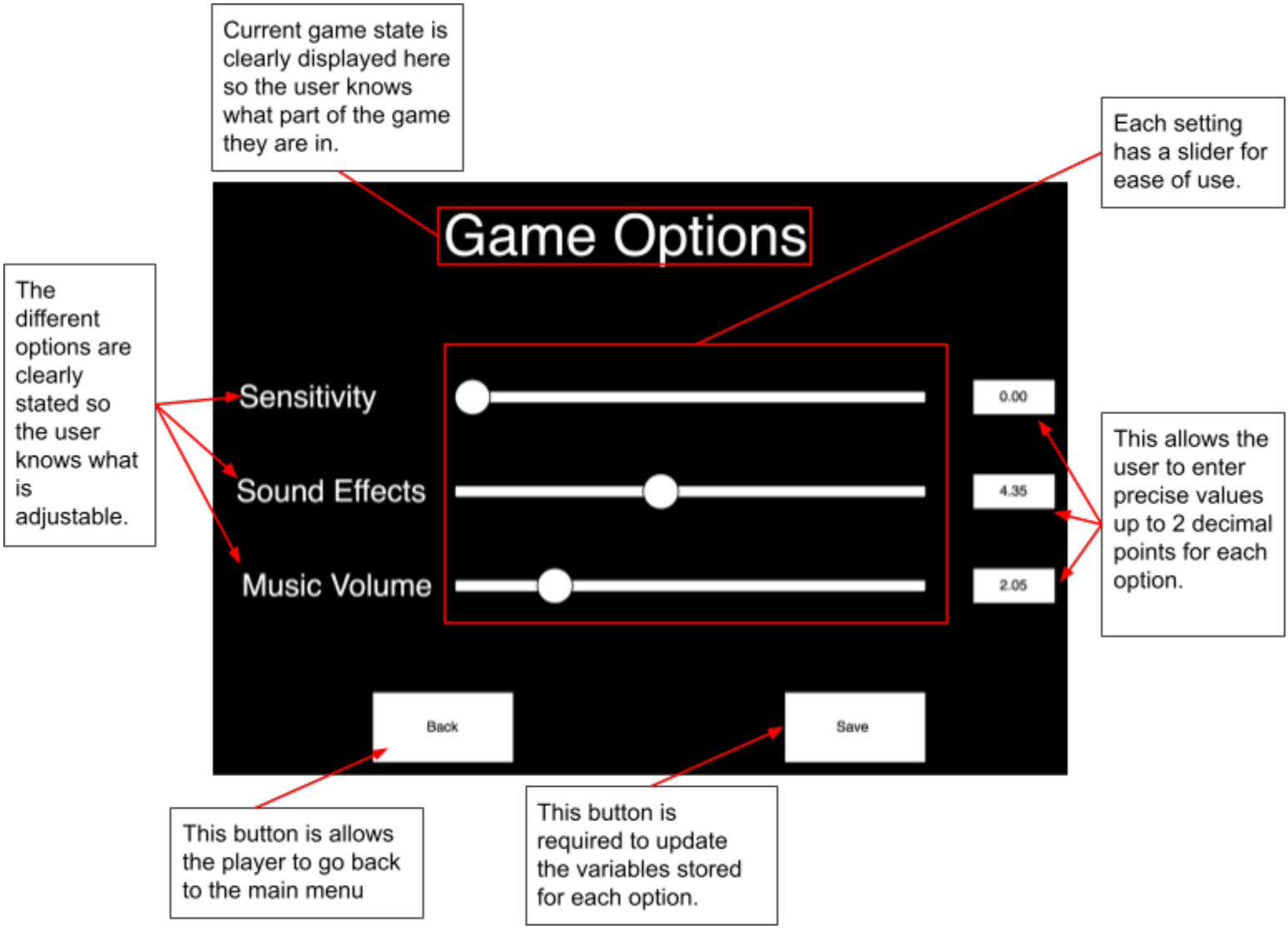
Main menu:



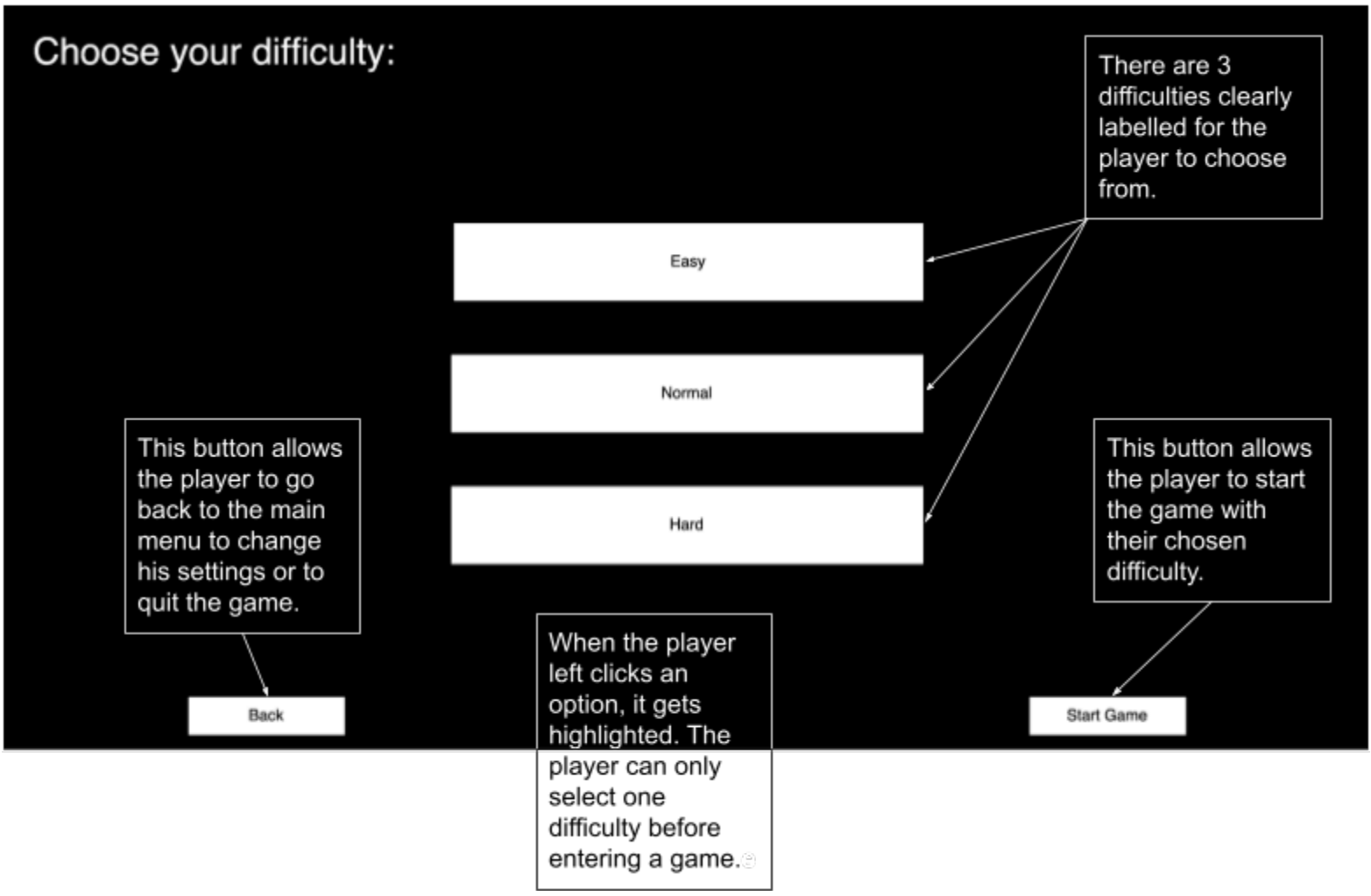
Loading a saved game:



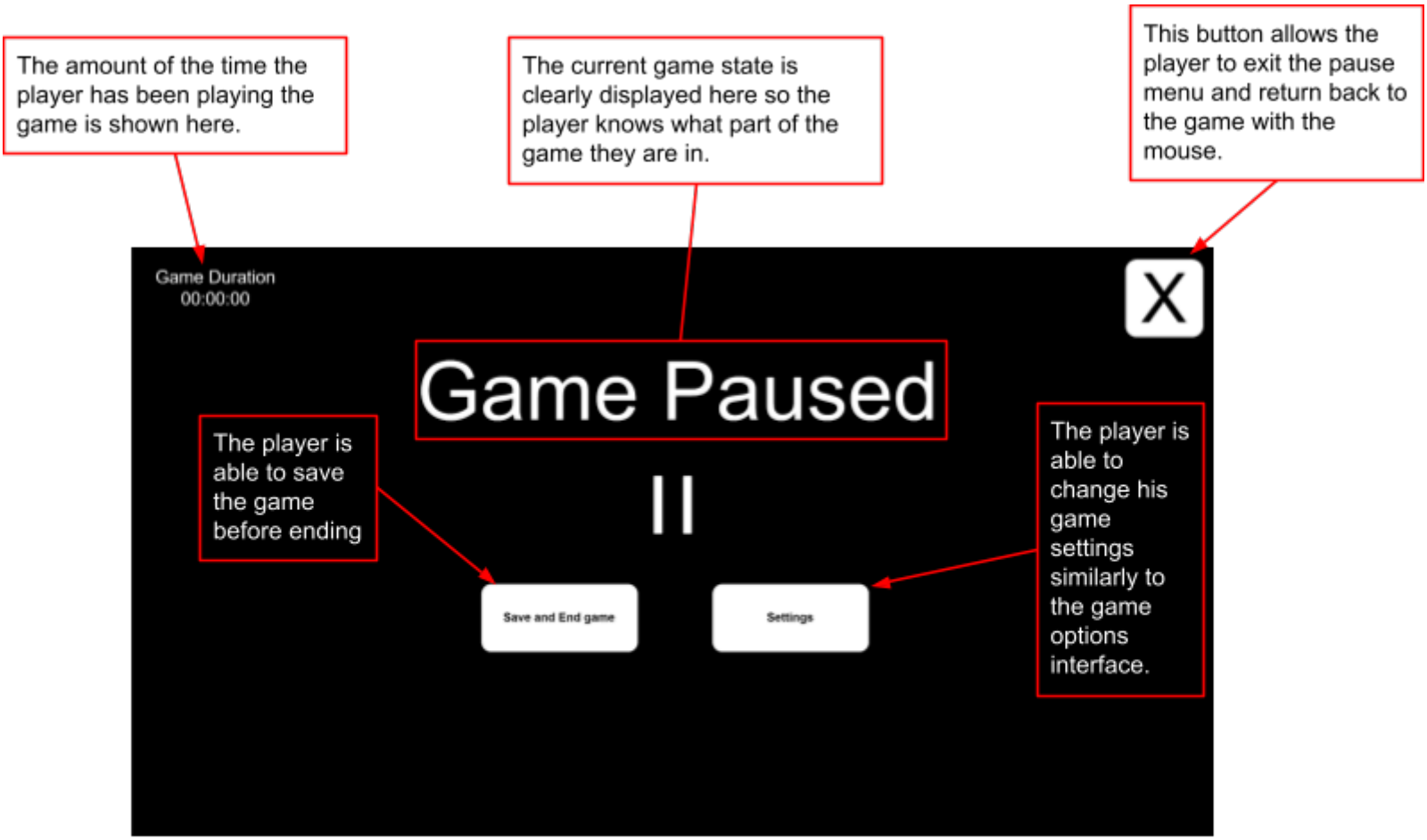
Game settings:



Difficulty options:

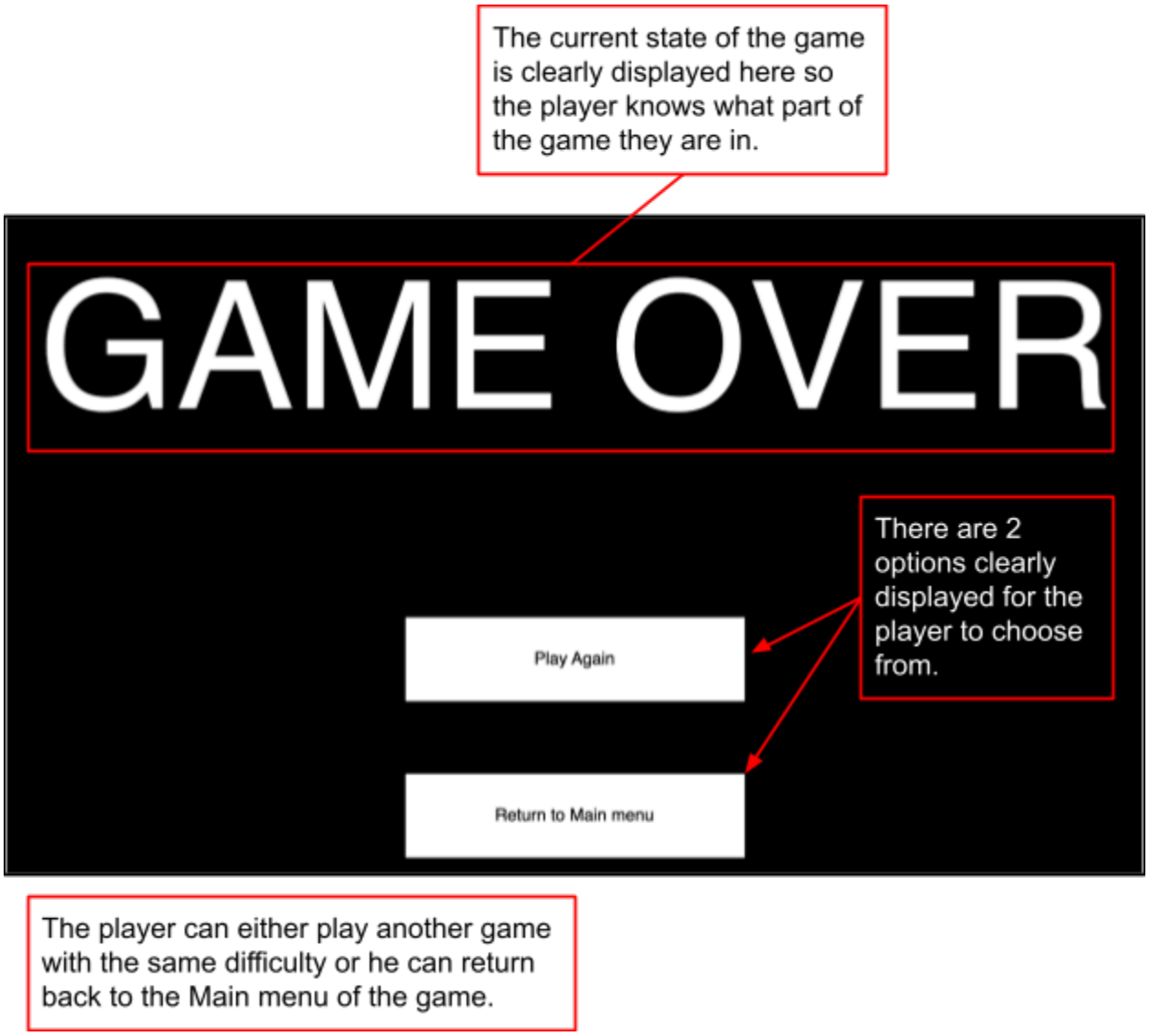


Pause Menu:

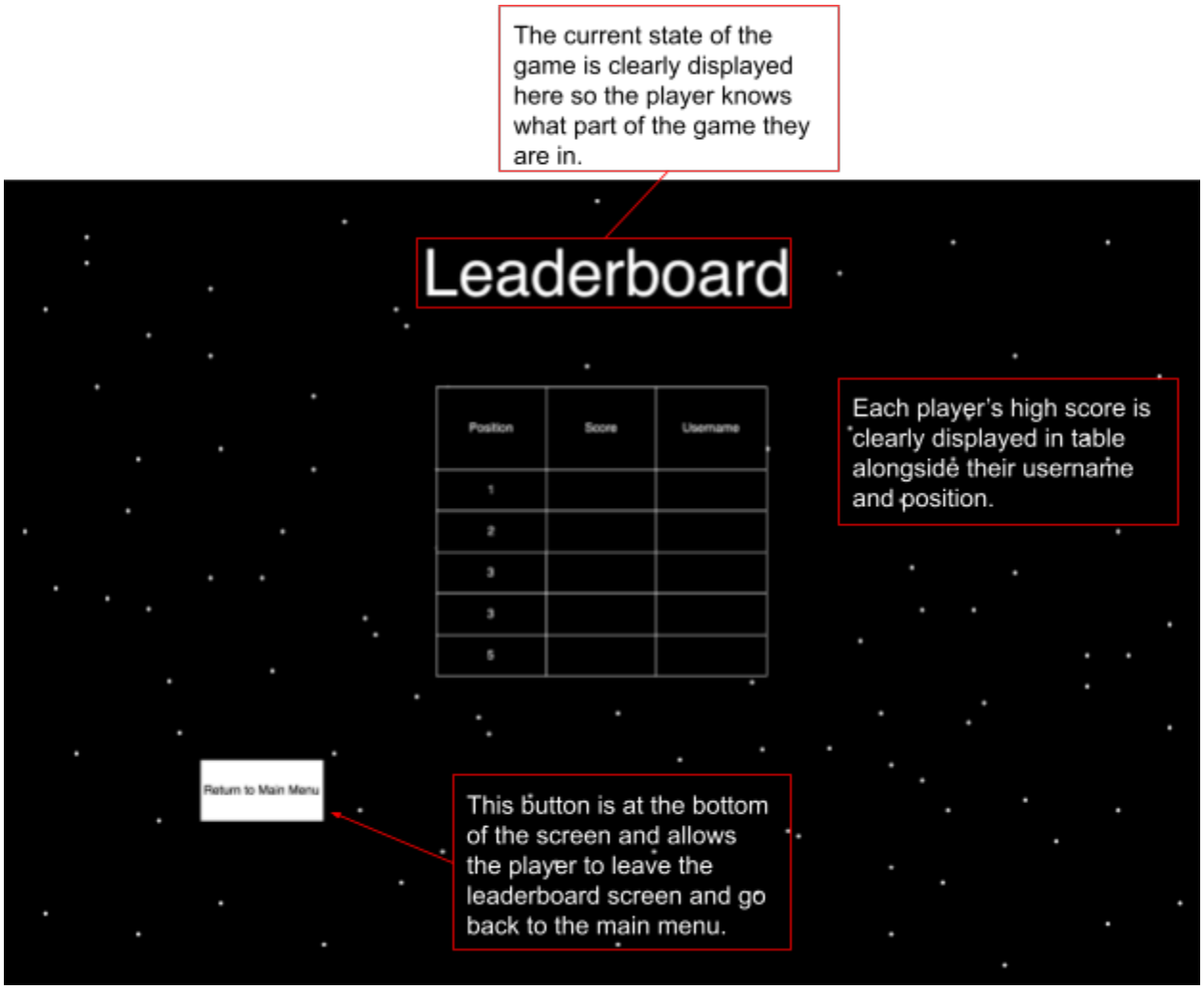


Having game settings in the pause menu allows the player to test out their setting right after they have been changed. This allows the player to adjust the setting until they are comfortable.

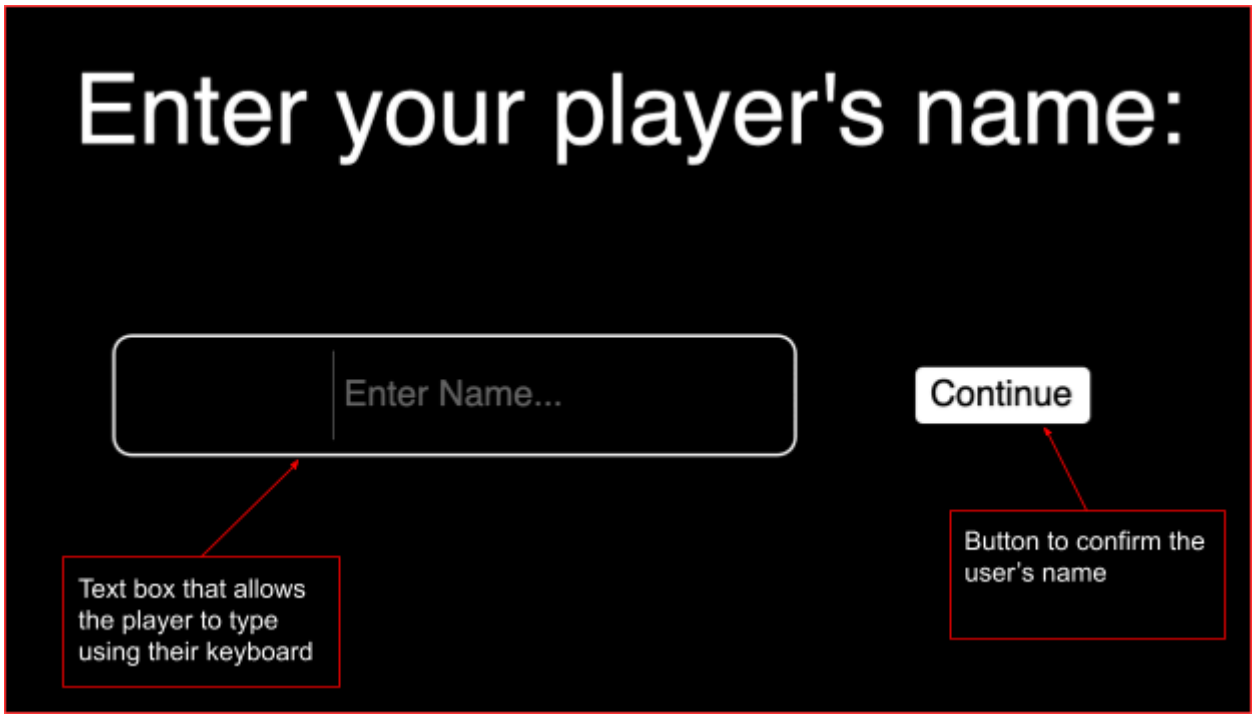
Game over screen:



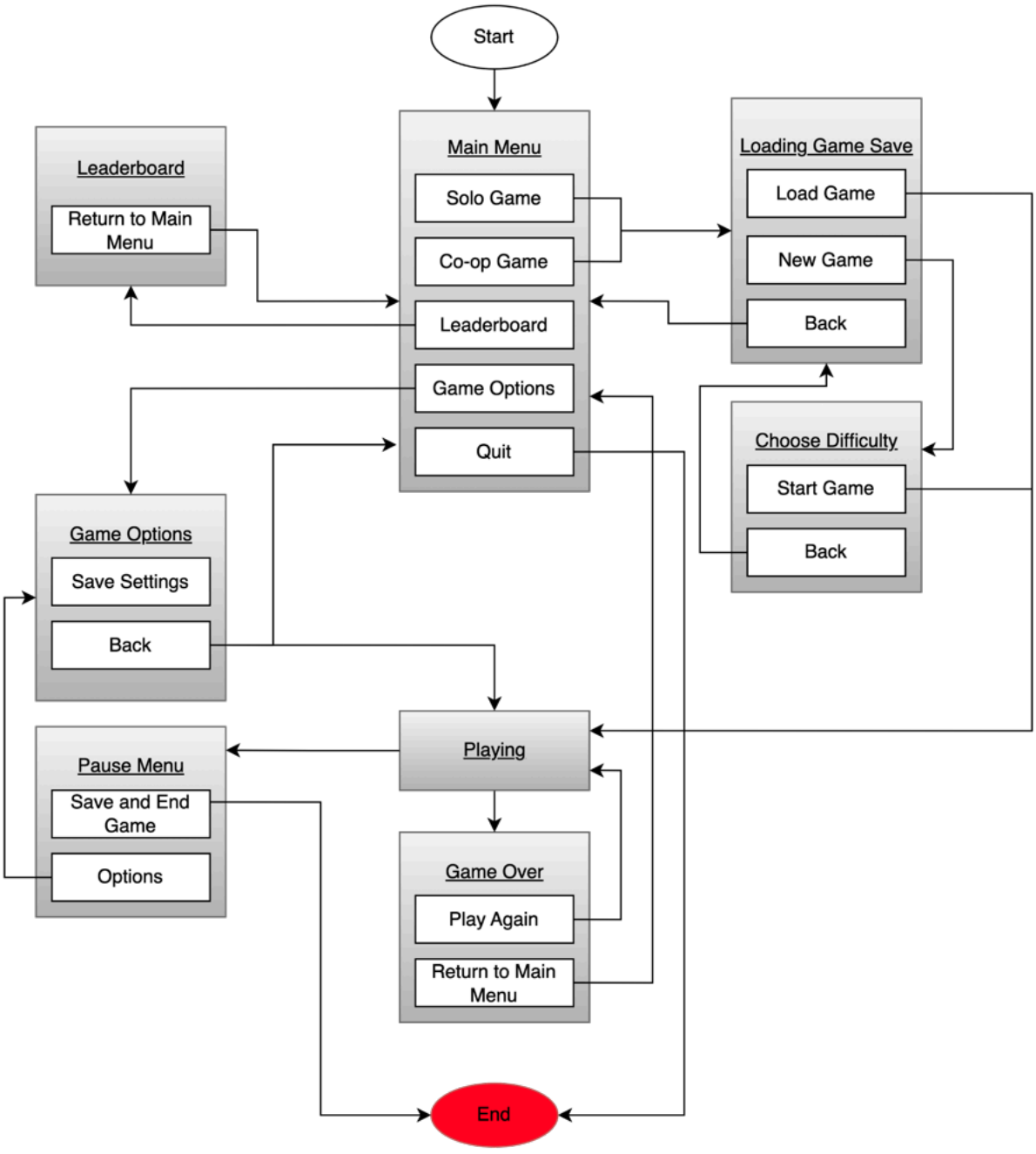
Leaderboard screen:



Player identification screen:



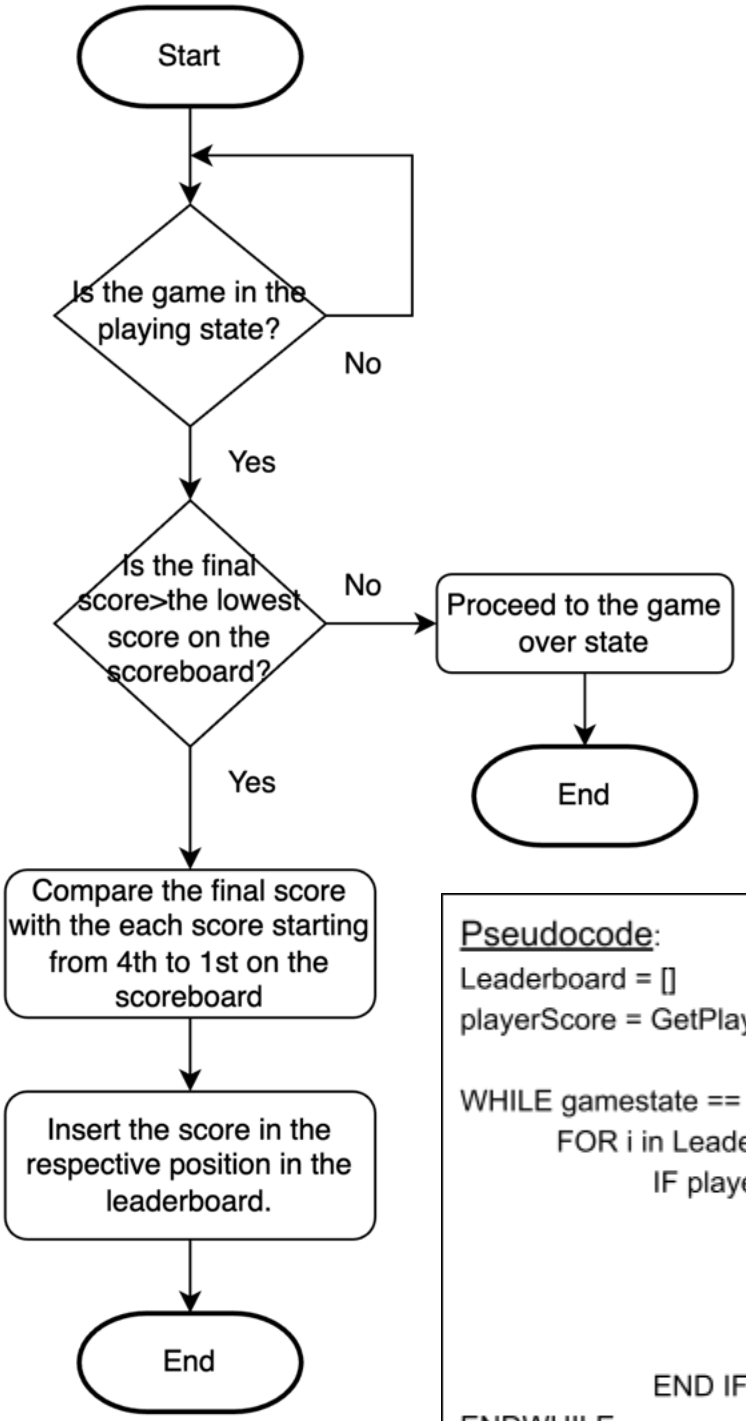
User Interface flow diagram:



This diagram is used to visualise how all the different user interfaces will interact with each other. It also provides a structure for the game which will help during the development process. Each grey box represents a state/interface within the game while the smaller white boxes inside the grey boxes represent buttons that lead to a different interface/state.

Flowcharts and Pseudocode:

1. New High Score (Flowchart):

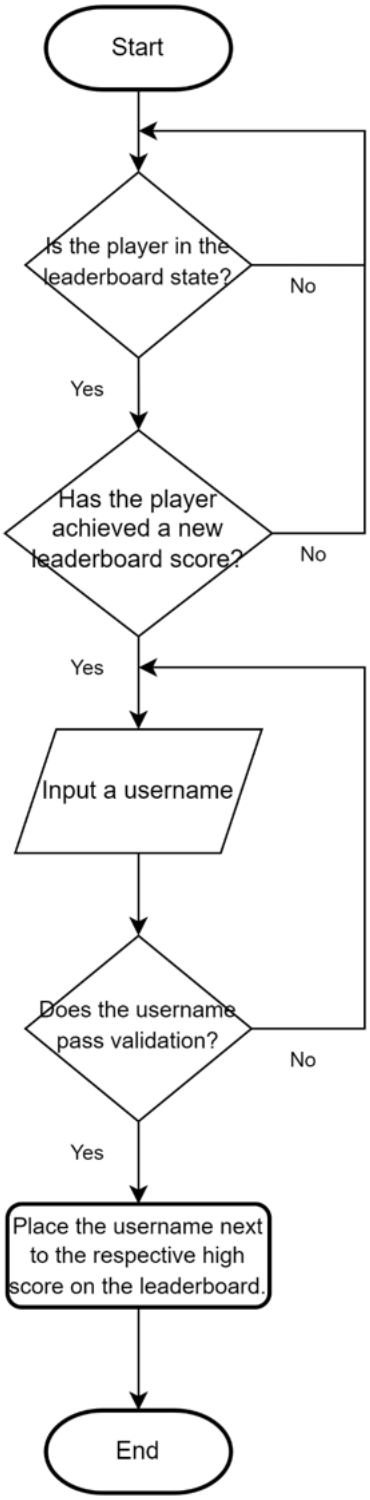


This flowchart is used to illustrate how a new score that is greater than the top 5 scores achieved will be added to the leaderboard. This is done by comparing the player's new score to the lowest score on the leaderboard. The player's score is then compared to the rest of the leaderboard to find the correct position. This will most likely be implemented through Unity's Leaderboard software development kit.

Pseudocode:

```
Leaderboard = []
playerScore = GetPlayerScore()

WHILE gamestate == Playing:
        FOR i in Leaderboard:
                IF playerScore >= Leaderboard[i]
                        Leaderboard.insert(i, playerScore)
                        IF length(Leaderboard) > maxLength
                                Leaderboard.remove(maxLength-1)
                        END IF
                END IF
ENDWHILE
```

2. High score name and validation(Flowchart):



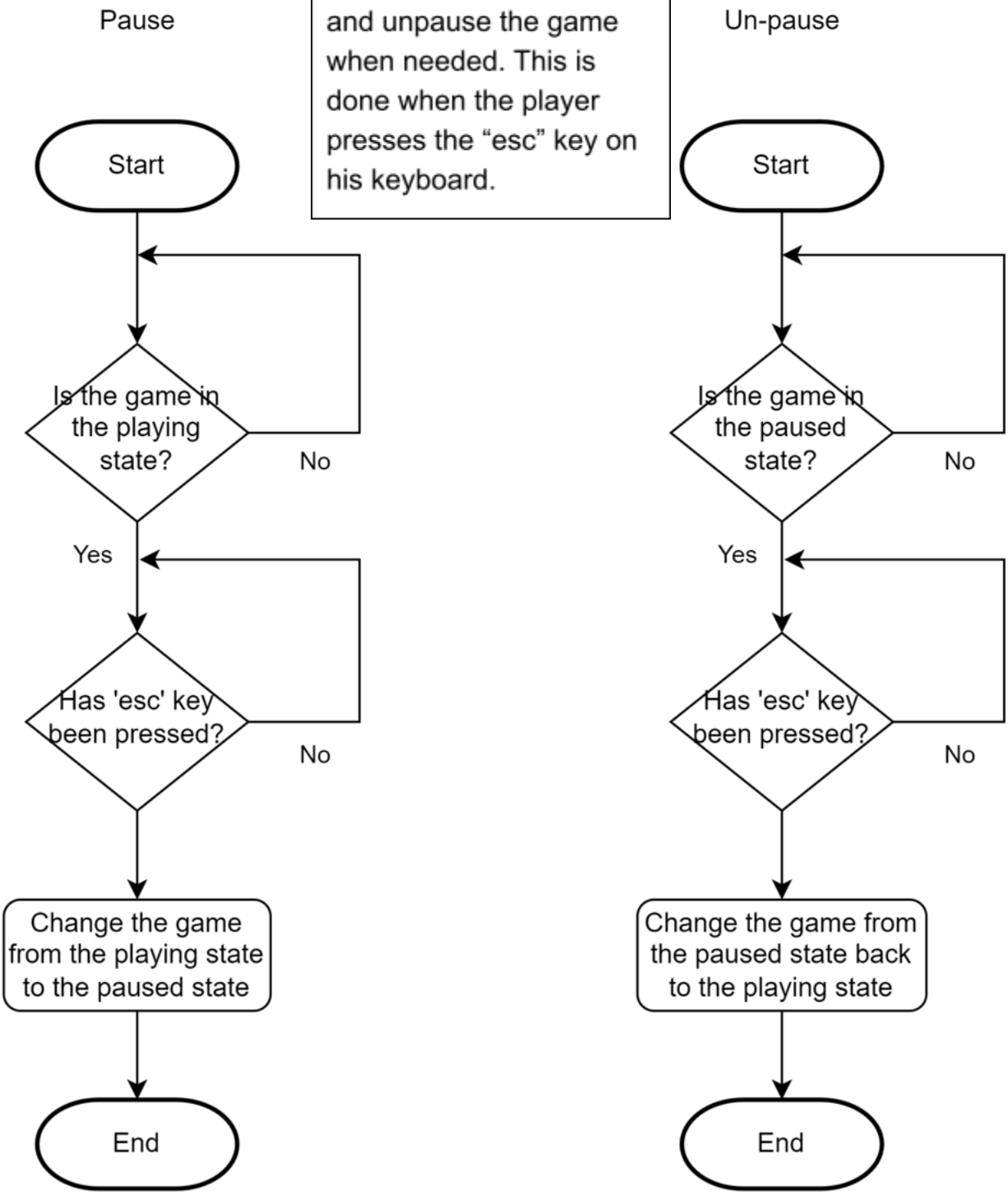
This flowchart is used to validate the username that the player enters after getting a new high score.This is only done if the player gets a new highscore. It will also place the player's username in their respective position on the leaderboard.

Pseudocode:

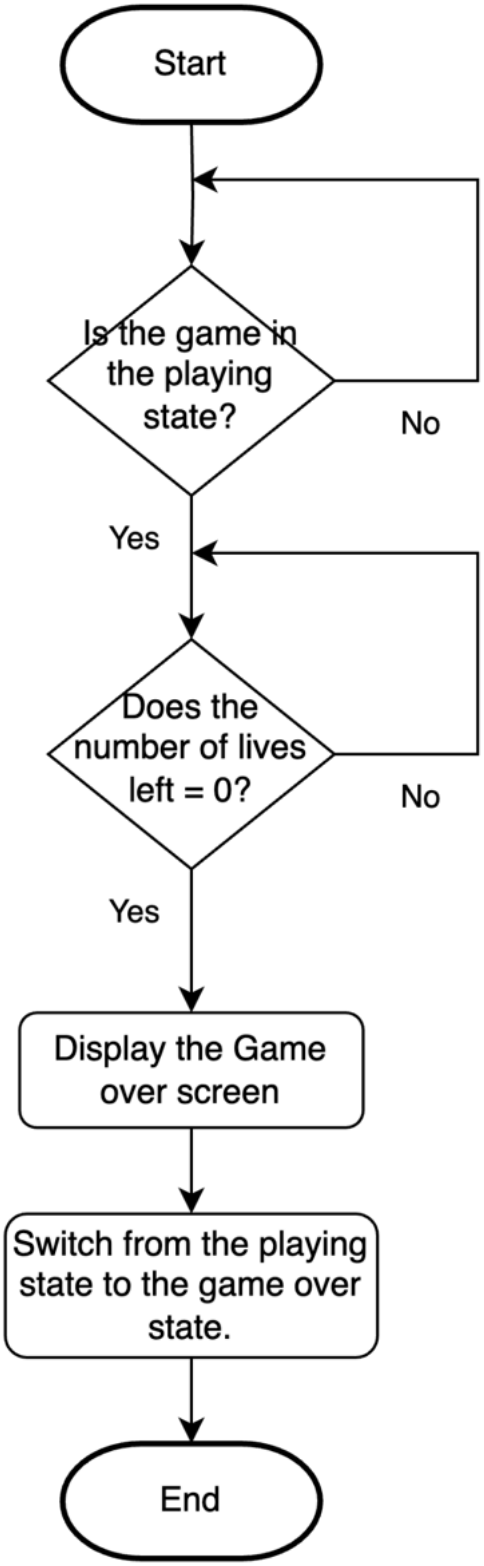
```
WHILE gamestate = "Leaderboard":
        highscoreName == input("Enter your name:")
        isValid = ValidateName(highscoreName)
        If !isValid:
                print("You have entered an invalid name.
Try again without symbols")
END WHILE


Function ValidateName(playerName):
IF playerName.length() = 0:
        return  false
Else:
         FOR i in playerName():
                IF i is a symbol:
                        return  false
                END IF
        END FOR
END IF

Return true
```

3. Game pause:

These flowcharts depict how the player can pause and unpause the game when needed. This is done when the player presses the “esc” key on his keyboard.



4. Game Over:

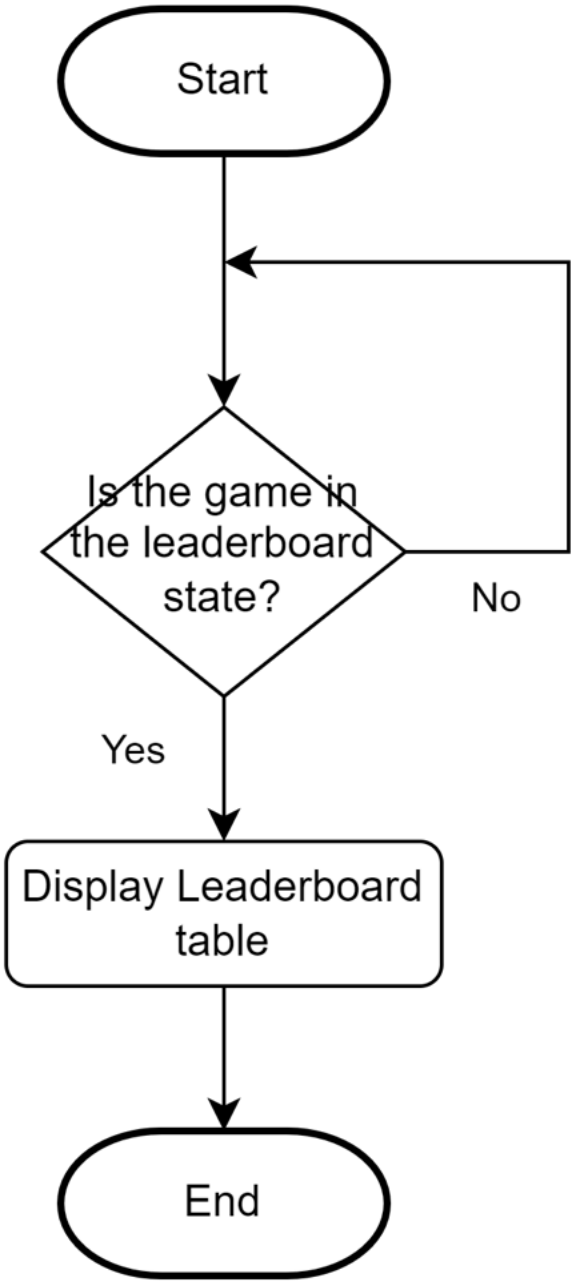


This flowchart ensures that the game ends by tracking the player's remaining lives. Once the player loses all their lives, the game will end.

Pseudocode:

```
WHILE gamestate == "Playing":
        IF livesLeft <=0:
                Display(GameOverScreen)
                Gamestate == "GameOver"
        ENDIF
ENDWHILE
```

5. Displaying the leaderboard:

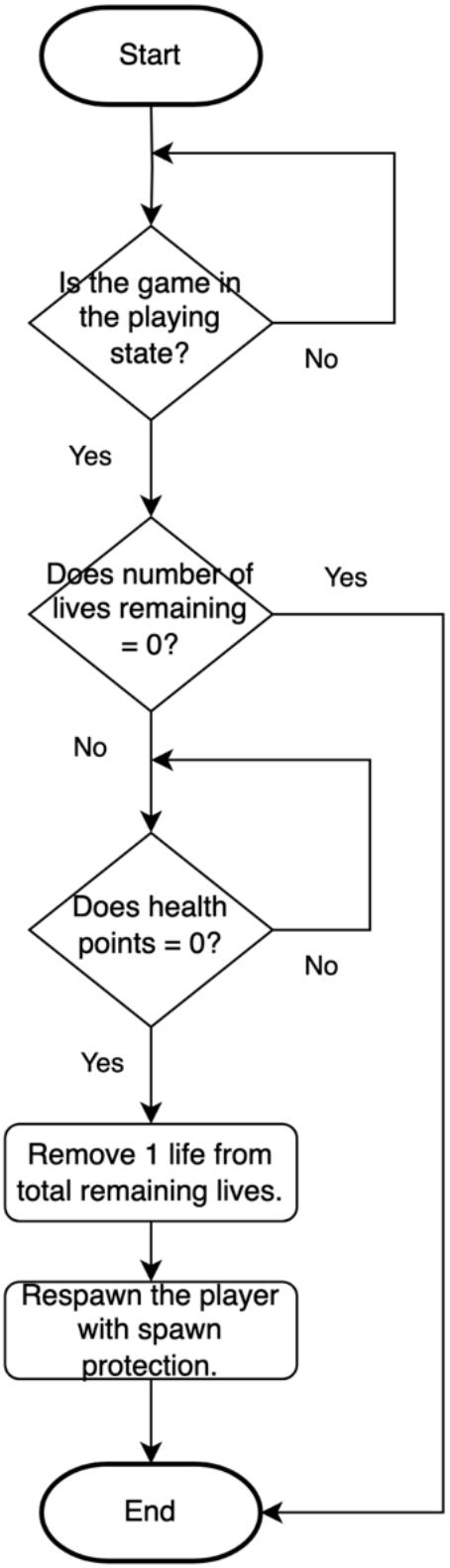


This simple flow chart displays the leaderboard when the player wants to look at the leaderboard. This is done when the player clicks on the leaderboard button in the main menu.

Pseudocode:

```
IF gamestate = "Leaderboard":
        Display(LeaderboardScreen)
END IF
```

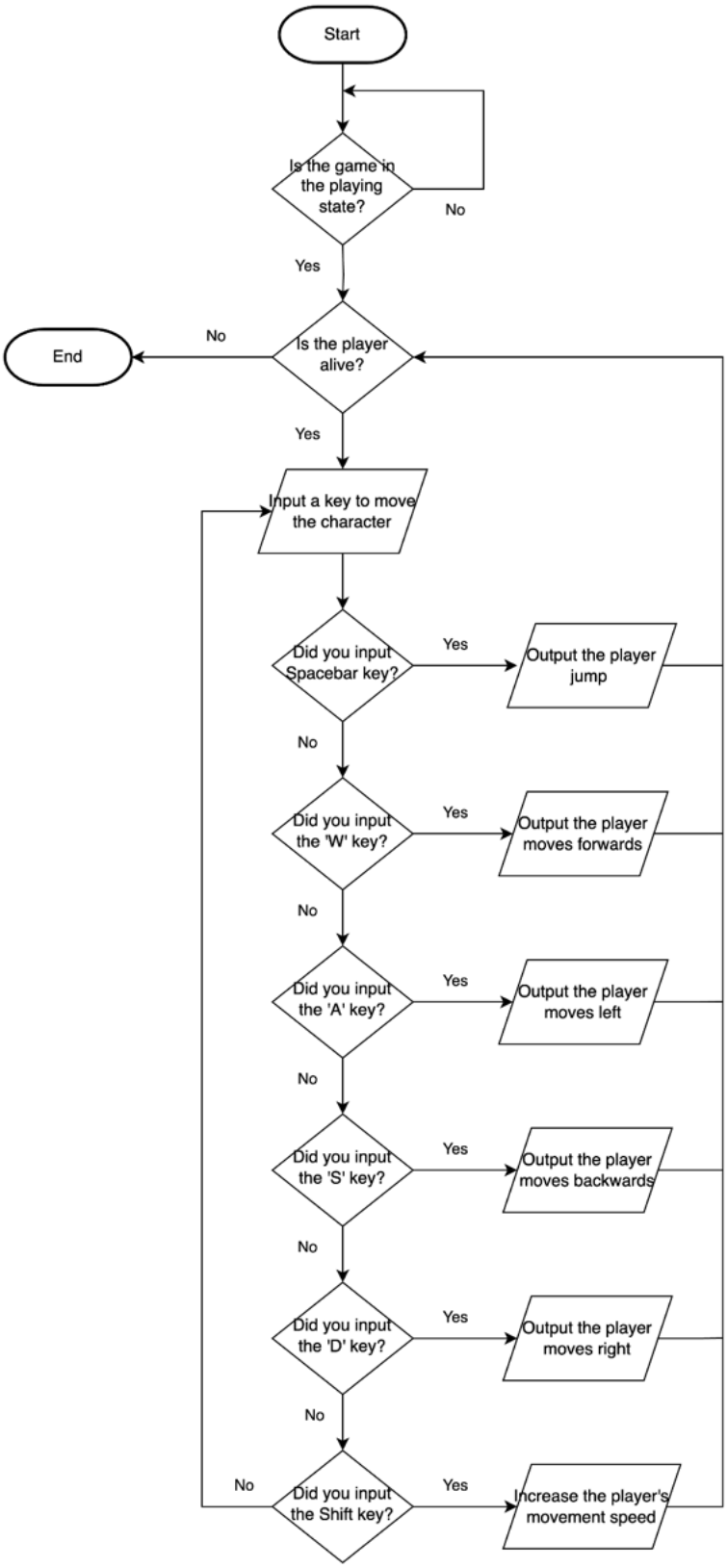
6. Life removal system:



This flowchart controls the player's health and lives remaining. If the player's health reaches 0, they will die and a life will be taken off their total life left. They will then be respawned with spawn protection. This is done until the player has no more lives left.

```
IF gamestate = "Playing":
    WHILE lives != 0:
        IF health = 0:
            lives -=1
            health = maxhealth
        END IF
    END WHILE
```

7. Character Movement:

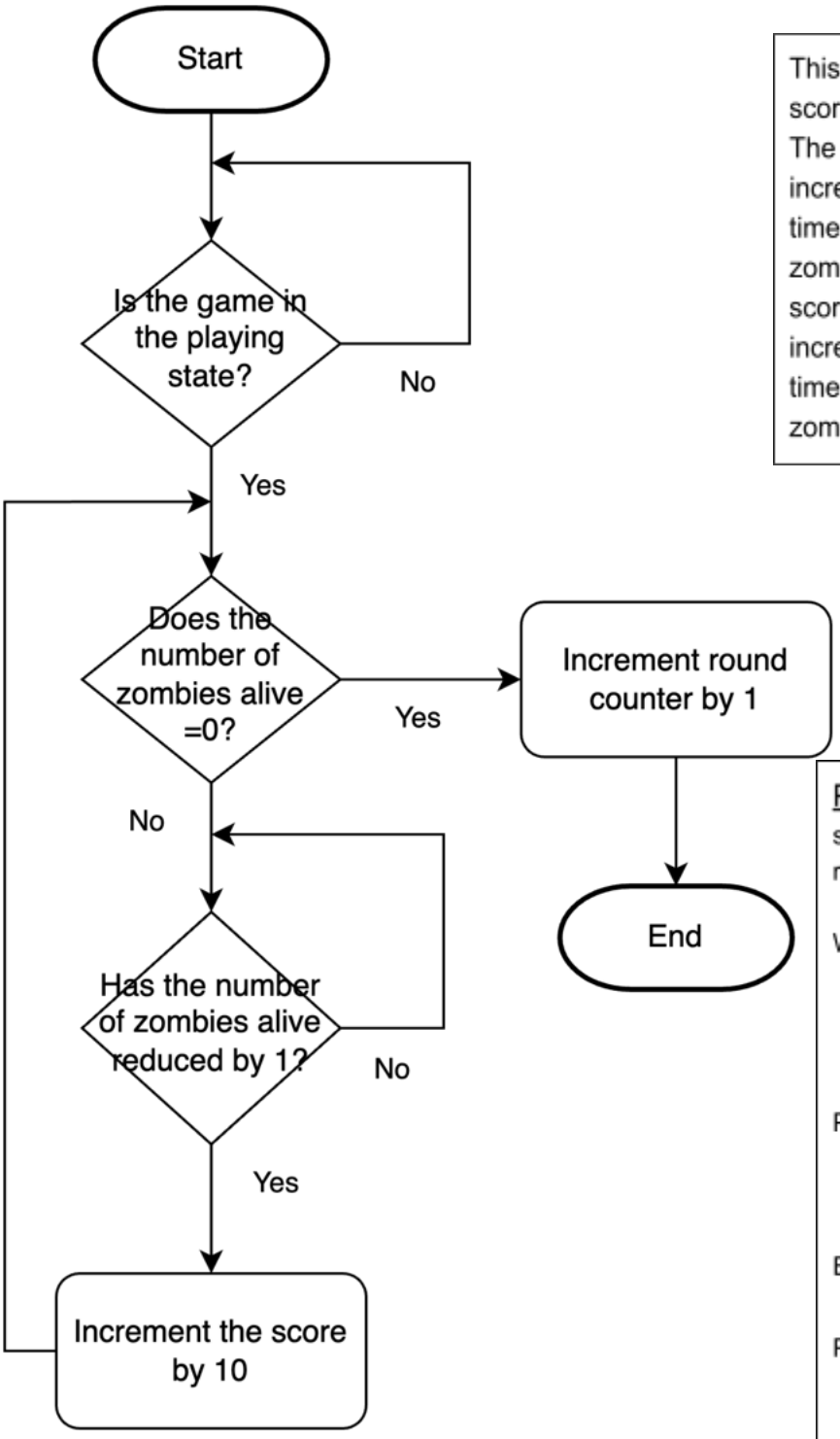


This flowchart simply manages how the movement system for the game is going to function. Depending on the player's input, the respective action is performed.

Pseudocode:

```
Speed = 20
jumpForce = 5

movementDirection = vector3(0, 0, 0)

IF input.key("W"):
    movementDirection += forwardsDirection
ENDIF

IF input.key("A"):
    movementDirection -= rightDirection
ENDIF

IF input.key("S"):
    movementDirection -= forwardsDirection
ENDIF

IF input.key("D"):
    movementDirection += rightDirection
ENDIF


IF input.key(Spacebar):
    rb.addforce(vector3.up * jumpForce)
ENDIF
```

8. Score and Round counter:



This flowchart manages the score system for the game. The round counter increments by 1 for each time the player kills all the zombies on the map. The score counter will also increment by 10 for each time the player kills a zombie.

Pseudocode:

```
score = 0;
round =1;

WHILE gamestate = Playing:
        score.Display()
        round.Display()


PROCEDURE scoreGain(scoreValue)

        score += scoreValue

END  PROCEDURE

PROCEDURE roundGain()

        round +=1

END PROCEDURE
```

Validation:

Input validation is a check performed to ensure that user inputs are entered correctly and allows the program to function as intended. This means if the user inputs do not meet the required validation, they will receive a message telling them to change their input so that it is valid for the program.

There are multiple types of validation that can be used depending on what is the intended goal. The types of input validation include:

- Length Check: This input validation check ensures that the user inputs data that doesn't fall short or exceeds a certain threshold. This usually involves the user inputting data with a certain amount of characters. It is fundamental as it prevents issues while maintaining the data integrity and can lead to excessively long inputs and outputs.

- Presence Check: This input validation check ensures that the user has not left any essential fields blank by checking if the user input data or not. If the user does not input anything they will receive a clear message informing them that they will have to input data of some sort within the field. This check is crucial for making sure that data is collected and processed correctly.

- Type Check: This input validation check ensures that the data the user input is of the correct data type. For example, if the user is entering a username for their player, there could be validation that forces the user to only use characters for their username. Otherwise the user will be warned to change their username to bypass the validation.

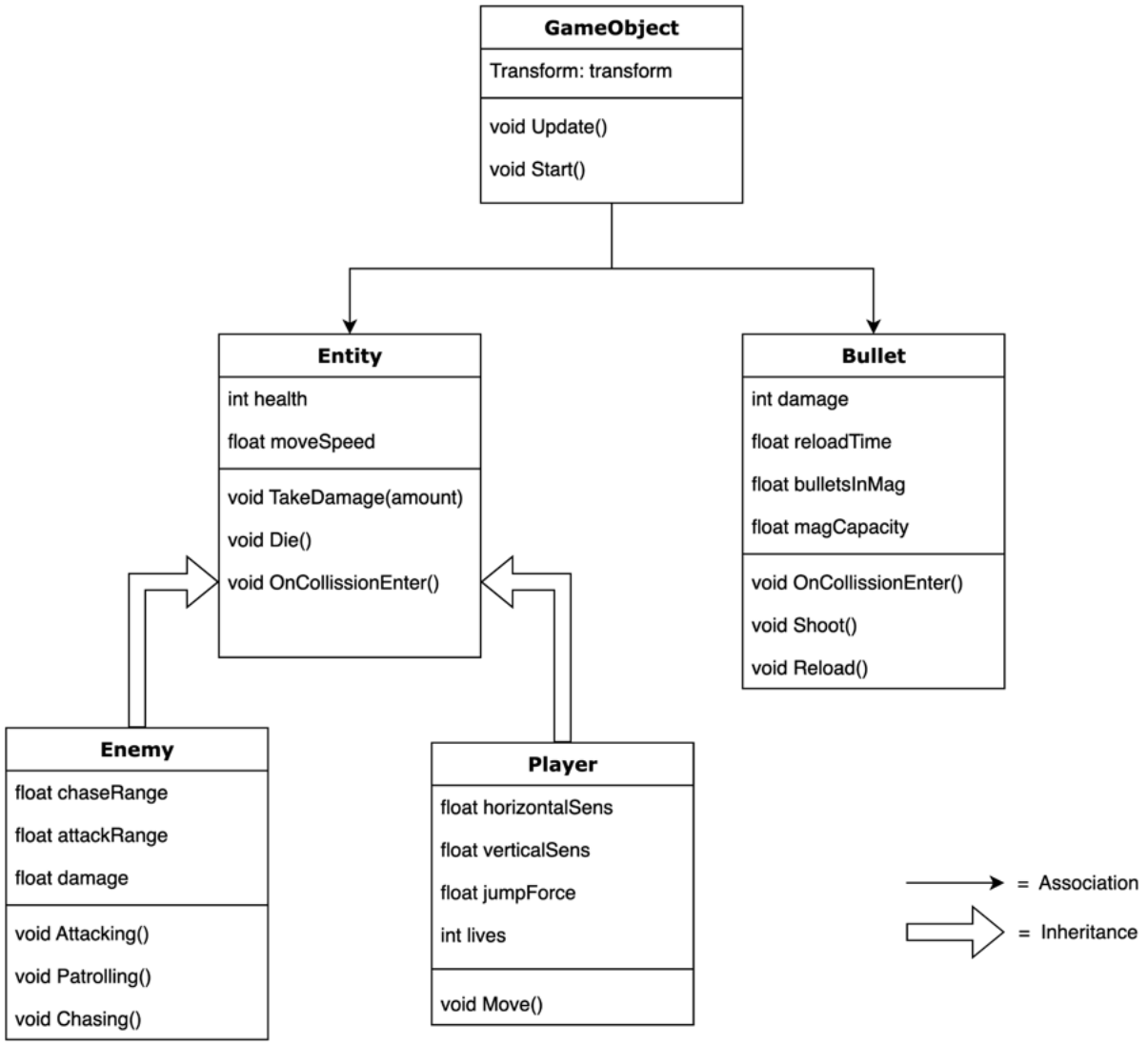
- Range Check: This input validation check is similar to the length check as they both involve the size of the data that the user inputs. The range check ensures that data entered by the user is inbetween a certain range of characters. This prevents possible outliers within the data which could possibly disrupt the functionality of the system

These are important for my project because I will require the player to input data in some circumstances. This may include their username on the leaderboard for the name of the game save file they create. I will most likely be using multiple types of validation checks for each field where the user inputs data. For example, when the user is creating a new game save file, I would implement a presence check and a range check. Whereas, when the user is creating a username for the leaderboard, I would additionally implement a type check and require the player to not use any integers. This is because it would be confusing if the user's name was just

numbers and so are the highscores. Therefore, the user's name will only be able to have characters and symbols.

Another type of validation I could do is to make sure the player doesn't leave the playable area of the map. Although this isn't directly validating the user's text input, It is still fundamental to the usability of the game. This could be done by making an invisible barrier around the edge of the playable area, which prevents the player from being stuck in an unplayable position, possibly resulting in the player losing all their progress in the game.

UML Class Diagram:



Test strategies:

Testing will be a necessary part of the development process. By using a range of different test strategies I will be able to determine if the client's system is fully functional and suitable for use. Therefore the end users will be happy with the system as there are no game-breaking bugs or issues which would prevent them being able to enjoy the game. Furthermore, I will be able to verify if I have met the success criteria previously made.

Some of the test strategies I will be using for this project is listed and explained below:

Alpha testing - For this test strategy I will be enquiring my stakeholders to partake in an alpha test session. The session will take place once I feel that the game is developed to the point it is playable and will last around two weeks. The main objective of this test is to find any problematic bugs or issues occurring which concern the functionality of the game. This is important as I will be able to fix any major issues for the game before It is released.

Beta testing - Similar to alpha testing, I will require people to test out the system. However, beta testing will be open to a much larger group of people compared to alpha testing. The main objective of beta testing is to test the robustness and reliability of the system. Beta test users will be chosen through a certain criteria to ensure that they are a suitable candidate for testing the game. Due to the large number of people testing the game, the beta test session would only last a week.

Blackbox testing - This is a method which will be used by the testers when they are alpha and beta testing. While testing the game, they will have to analyse the functionality of the system without any knowledge of the internal design. They would then record any issues which were found and send them to me, which I will then solve. For example, the player might respawn and take damage even when they still have spawn protection. The user would then include this bug in his record of issues in the game.

Whitebox testing - This method is likely to be used throughout the development of the game by the developer. I would need to make sure that the game behaves correctly and adheres to the success criteria. This means that certain actions must have expected outputs, if they don't, I would need to fix the issue. These issues are generally logic or syntax errors which need to be fixed by changing the source code of the game.

Data Dictionary

Method	Name	Data type	Explanation	Justification
Main Menu				

Function	LoadGame	N/A	This function switches from the main menu state to the game state	This allows the player to start playing the game from the main menu
Function	LoadSettings	N/A	This function switches from the main menu state to the settings state	This allows the player to go to their settings from the main menu
Function	LoadLeaderboard	N/A	This function switches from the main menu state to the leaderboard state	This allows the player to look at the leaderboard from the main menu
Function	ExitGame	N/A	This function quits the game application	This allows the player to quit the game from the main menu
Leaderboard				
Variable	name	string	This variable holds player's name	This allows the highscores to be identified with whoever achieved them.
Variable	score	float	This variable holds the player's score	This allows the score the player gets in the game to be displayed on the leaderboard
List	playerScores	float	This list stores the sequence of the highscores in order	This allows the leaderboard to be in the correct order
Function	AddScore	N/A	This function adds any new highscores to the leaderboard	This allows the player's new highscore to be added to the leaderboard
Function	UpdateLeaderboard	N/A	This function updates the contents of the leaderboard	This allows the player to view their new highscore on the leaderboard.
Pause Menu				
Function	Paused	N/A	This function pauses the game and opens the pause menu	Allows the player to stop playing the game and adjust any settings

				without leaving the game.
Function	Resume	N/A	This function closes the pause menu and unpauses the game	Allows the player to return back to his game
Function	LoadSettings	N/A	This functions loads the settings interface	Allows the player to change and save any setting while still in the game
Function	QuitGame	N/A	This function saves and then quits the game the player is in	Allows the player to go back to the quit the game and save his progress
Game Options				
Variable	sensitivity	float	This variable holds the sensitivity of the mouse	Allows the player to adjust their sensitivity
Variable	soundEffects	float	This variable holds the volume of the sound effects in the game	Allows the player to adjust the volume of the sound effects
Variable	musicVolume	float	This variable holds the volume of any background music in the game	Allows the player to adjust the background music for the game
Save Game				
Variable	score	integer	Stores the current score of the player	Allows the player's progress to be saved
Variable	health	integer	Stores the current health of the player	Allows the player's progress to be saved
Variable	lives	integer	Stores the current lives of the player	Allows the player's progress to be saved
Array	position	float	Stores the current X, Y and Z positions of the player	Allows the player's progress to be saved
Function	SavePlayer	N/A	This functions saves all the stats of the player in a file	This allows the player to save his game data to play later

Function	LoadPlayer	N/A	This function loads all the previously stored stats of the player from the file	This allows the player to continue playing a previous game without losing any progress
Game Over				
Function	RestartButton	N/A	This function switches from the game over screen back to the playing state	This allows the player to start playing again after clicking the respective button
Function	ExitButton	N/A	This function switches from the game over screen to the main menu screen	This allows the plate to return to the main menu clicking the respective button
Player				
Variable	speed	float	This variable holds the current movement speed of the player.	This is allows the player's movement speed to be manipulated
Variable	jumpHeight	float	This variable holds the height the player jumps to	This determines how high the player can jump
Variable	horizontalInput	float	This will hold any horizontal input from the mouse	Allows the player's input from mouse to look around in the game
Variable	verticalInput	float	This will hold any vertical input from the mouse	Allows the player's input from mouse to look around in the game
Component	playerRb	Rigidbody	This component is used to control the physics of the player	This allows the player to move around the map and jump
Function	TakeDamage	N/A	This function will decrease the player's health by an amount when hit by the enemy	This puts the player at risk as too much damage will lead to death
Variable	Lives	float	This will hold the total number or lives of the player	This makes the game more challenging for the player

Function	Die	N/A	This function controls what happens when the player dies	Allows the player to lose lives, if they lose too many lives, the game will end
Variable	Health	float	This holds the health of the player	This makes the game more challenging for the player
Sprite	filledHeart	.png	This is a pixel art image of a filled heart	This acts as indicator for the player's lives left
Sprite	emptyHeart	.png	This is a pixel art image of an empty heart	This acts as indicator for the player's lives left
First Person Camera				
Variable	SensX	float	This holds the speed the player looks horizontally	The speed can be changed to suit the player's liking
Variable	SensY	float	This holds the speed the player looks vertically	The speed can be changed to suit the player's liking
Variable	xRotation	float	This variable is used to track the vertical rotation of the camera	Allows the the player too look around based on the rotation of the camera
Variable	yRotation	float	This variable is used to track the horizontal rotation of the camera	Allows the the player too look around based on the rotation of the camera
Component	cameraPosition	Transform	This holds the position the camera will be in when the game starts	This allows the camera to always be at the correct position
Enemy				
Component	enemy	NavMeshAgent	This component creates a NavMesh which the enemy uses to navigate on the map	This allows the enemy to move around on the map
Component	enemyAnimator	Animator	This component holds and controls how	This allows the enemy to perform the correct

			animations are played for the enemy	animations at the correct times
Variable	SightRange	boolean	This boolean checks whether the player is in the enemy's sight	This allows enemy to chase the player at the correct time
Variable	AttackRange	boolean	This boolean checks whether the player is close enough to be attacked by the enemy	This allows the enemy to attack the player at the correct time
Function	Patrolling	N/A	This function holds the logic for the enemy when the player is nowhere near	This allows the enemy to be realistic instead of just standing still
Variable	randomZ	integer	This is a random value which will be used as a Z coordinate	This gives the enemy a position to go to instead of standing still in the patrol state
Variable	randomX	integer	This is a random value which will be used as an X coordinate	This gives the enemy a position to go to instead of standing still in the patrol state
Function	ChasePlayer	N/A	This function holds the logic for the enemy chasing the player	This allows the enemy to chase after the player
Function	AttackPlayer	N/A	This function holds the logic for the enemy attacking the player	This allows the enemy to attack the player
Variable	timeBetweenAttacks	float	This holds the time in between each attack by the enemy	This gives the player a chance to escape when getting attacked by the enemy.
Gun				
Variable	bulletsLeft	integer	This holds the number of bullets left in the magazine	Allows the player to keep track of how many bullets they are using
Variable	magazineSize	integer	The holds the maximum number of bullets in the magazine	Allows the player to know how many bullets he can shoot before needing to reload

Function	Shoot	N/A	This function is used to shoot bullets from the gun	Allows the player to defend themselves from the enemies attacking them
Variable	timeBetweenShots	float	This holds the time between each shot when the player is shooting	This is so the gun is realistic as bullets are shot one after the other
Variable	shootForce	float	This acts as the force applied on the bullet	This allows the bullet to be shot after they are instantiated
Variable	direction	Vector3	This holds the direction the bullets will travel in	This allows the bullet to be shot after they are instantiated
Game Object	Bullet	N/A	This is the bullet object which will be getting instantiated when shot	This allows the player to shoot
Function	Reload	N/A	This function reloads the magazine for the player's gun	This allows the player to shoot more than one magazine at the enemies
Variable	reloadTime	float	This is the time taken for the player to reload the gun	This makes the game more challenging as the player will have to wait to be able to shoot again
Sprite	Crosshair	.png	This is a sprite of a crosshair	This allows the player to aim where their bullets are going

Post Development Test Plan:

Success Criteria	Test No.	Description	Test Data	Expected Outcome
Main menu with game title and 4 options: Play game, Co-OP, leaderboard and game options.	1	Check if the main menu loads when the game opens.	N/A	Main menu loads with the game title and 3 options.
	1b	Check if player can go to leaderboard screen	Left mouse click on “Leaderboard” button	Game goes to the leaderboard screen
	1c	Check if player can go to start a game	Left mouse click on “Play” button	A new game is started
	1d	Check if player can go to Co-op screen	Left mouse click on “Co-op” button	Game goes to Co-op screen
	1e	Check if player can go to game options screen	Left mouse click on “Game Options” button	Game goes to options screen
Display the current player’s score in the game.	2	Check if score box is displayed on-screen	N/A	Score box is displayed on the top right of the screen
The highest scores achieved in a game are displayed on the leaderboard	3	Check if top 5 highest scores are appear on the leaderboard	N/A	The top 5 highest scores are output onto the leaderboard
	3b	Check if a new high score is added to the leaderboard	Valid: Finish the game with a score higher than the top 5 Invalid: Finish game with score lower than top 5	The new high score is inserted into the designated spot on the leaderboard.
Display the health bar of the player and the enemies	4	Check if the player’s health	N/A	The player’s health bar is displayed at the

		bar is displayed on-screen		bottom of the screen
	4b	Check if the enemies' health bars are displayed on-screen	N/A	Each enemies' respective health bar is displayed above each of their heads
Terrain map for the game	5	Check if terrain loads in at the start of the game	N/A	The terrain is fully loaded into the game
Player can look around freely	6	Check if player can look upwards	Move mouse up	The player looks up
	6b	Check if player can look downwards	Move mouse down	The player looks down
	6c	Check if player can look left	Move mouse left	The player looks left
	6d	Check if player can look right	Move mouse right	The player looks right
The player can move in all 4 directions as well as jump up into the air	7	Check if player can move forwards	Valid: press "W" key Invalid: press "L" key	Valid: The player moves forwards Invalid: Nothing happens
	7b	Check if player can move backwards	Valid: press "S" key Invalid: press "P" key	Valid: The player moves backwards Invalid: Nothing happens
	7c	Check if player can move to the right	Valid: press "D" key Invalid: press "U" key	Valid: The player moves to the right Invalid: Nothing happens
	7d	Check if player can move to the left	Valid: press "A" key	Valid: The player moves to the left

			Invalid: press “E” key	Invalid: Nothing happens
	7e	Check if player can jump up	Valid: press spacebar key Invalid: press “W” key	Valid: The player jumps up Invalid: player moves forward
The player can sprint to move faster	8	Check if player can sprint	Valid: hold “Shift” + “W”, “A”, “S” or “D” key Invalid: press “O” key	Valid: The player moves faster Invalid: the player does not move faster
The player can shoot	9	Check if the player can shoot	Valid: Left mouse click while in game	The player shoots their gun
	9b	Check if bullets damages the enemy	Left mouse click at enemy	The enemy’s health will decrease when hit with a bullet
	9c	Check if the player can reload their gun	“R” key is pressed	Valid: the gun is reloaded
	9d	Check if bullet deals extra damage when it hits the enemy’s head	Left mouse click at enemy’s head	The bullet is destroyed when it collides with the enemy
	9e	Check if bullet disappears when it hits with the enemy	Left mouse click at enemy	The bullet is destroyed when it collides with the ground
The player deals more damage with their weapon if they hit the zombie in the head.	9	Check if bullet disappears when it hits with the ground	Left mouse click at ground	The enemy’s health will decrease more than when a bullet hits the rest of the body

The player's score is increased by 10 points when an enemy is killed.	11	Checks if the player gains points for killing an enemy.	Valid: left mouse click on the enemy until enemy health<=0. Invalid: left mouse click at the ground	Valid: 10 points are added to the player's score variable. Invalid: The player's score variable remains the same.
	11b	Check if round is incremented after killing all the zombies on the map	Kill all the zombies on the map	Round counter is incremented by 1
The player starts with 100 health	12	Check the player starts with 100 health	N/A	When the game starts, the health variable is set to 100
The player starts with 3 lives	13	Check if the player starts the game with 3 lives	N/A	When the game starts, the life variable is set to 3.
Display the number of lives the player has.	14	Check if the number of lives are displayed	N/A	3 heart sprites will appear at the top of the screen
The player is given spawn protection upon respawning.	15	Check if the player is giving spawn protection after respawning	N/A	The player will be invincible from damage for a short period of time after respawning.
Animations are played for certain actions	16	Check if the enemy performs the chasing animation	Valid: Player moves into sight range of the enemy Invalid: Player is out of sight range of the enemy	Valid: The enemy performs a chasing animation Invalid: No animation is performed
	16b	Check if the enemy performs	Valid: Left mouse click on zombie	Valid: The enemy performs

		the dying animation	until health <=0 Invalid: Left mouse click on zombie when bullets in magazine variable = 0	the dying animation Invalid: No animation is performed
	16c	Check if the enemy performs the attacking animation	Valid: The player moves within range of melee attack from enemy Invalid: The player is out of range of melee attack from enemy	Valid: The enemy performs the attacking animation Invalid: No animation is performed
Sound effects are played for certain actions	17	Check if footstep audio is played when player moves around	Valid: “W”, “A”, “S” or “D” key is pressed/held down Invalid: “L” key is pressed/held down	Valid: The player makes footsteps sound when they move Invalid: No sound is played
	17b	Check if gunshot audio is played when player shoots	Left mouse click with gun	The gun will have a gunshot sound when shot
Increasingly larger waves of enemies are spawned randomly across the map.	18	Check if the enemies spawn randomly across the map	N/A	The enemies are spawned randomly on the map
	18b	Check if the waves of enemies increase in size after each round	N/A	A wave of enemies are spawned, larger than the previous wave
Enemies can attack and hurt the player	19	Check if enemy can attack the player	N/A	The player will lose health after getting hit by the enemy

	19b	Check if enemy can kill the player	N/A	The player will lose a life and the heart will become empty
Spawn a Boss enemy with extreme health and damage every 5 rounds	20	Check if a boss spawns every 5 rounds	Valid: Player is on a round that is a multiple of 5 Invalid: Player is on a round that is not a multiple of 5	Valid: A stronger enemy with more health is spawned amongst the rest Invalid: No boss is spawned
The player can pause their game	21	Check if the player can pause the game	“Esc” key is pressed	The game should pause
	21b	Check if player can unpause the game	“Esc” key is pressed when game is paused	The game should resume in the same state as it was when it was paused
The player can save their progress in a game and continue playing later.	22	Check if the player can save the game currently being played	Left mouse click on the “Quit and Save game” option in the pause menu in-game	The game should quit to the main menu and a new game save is stored
	22b	Check if old game save can be loaded	Choose respective game save and left mouse click on “start game” option	The game should load with correct game data and player progress
	22c	Check if the player can save multiple different games	N/A	The new game save should store separately from the rest
The player can use the in-game shop to purchase items to help them play	23	Checks if the player can access the shop	Valid: press the “G” key on the keyboard Invalid:	The shop should load with all the correct items as well as their price

			Press the “L” key on the keyboard	Invalid: Shop doesn't load
	23b	Check if player receives purchased item	Valid:Left mouse click on desired item Invalid: Right mouse click on desired item	Valid: The player instantly receives the item Invalid: The player does not receive the item
Game over screen when the game ends allowing them to either start a new game or return back to the main menu	24	Check if game over screen is displayed	N/A	The game over screen is displayed
	24b	Check if the player can start a new game from this screen	Left mouse click on “Play Again” button	The game starts again with the same chosen difficulty
	24c	Check if the player can return to main menu from this screen	Left mouse click on “Main Menu” button	The game goes to the main menu screen
The player can play with another person as a teammate	25	Check the player is able to join another person’s game	N/A	The player joins the other person’s game
The screen will have a red tint for 3 seconds after the player takes damage.	26	Check if red tint appears on screen when hit by enemy	Get attacked by enemy	A red tint will appear on the screen to indicate being hit by the enemy

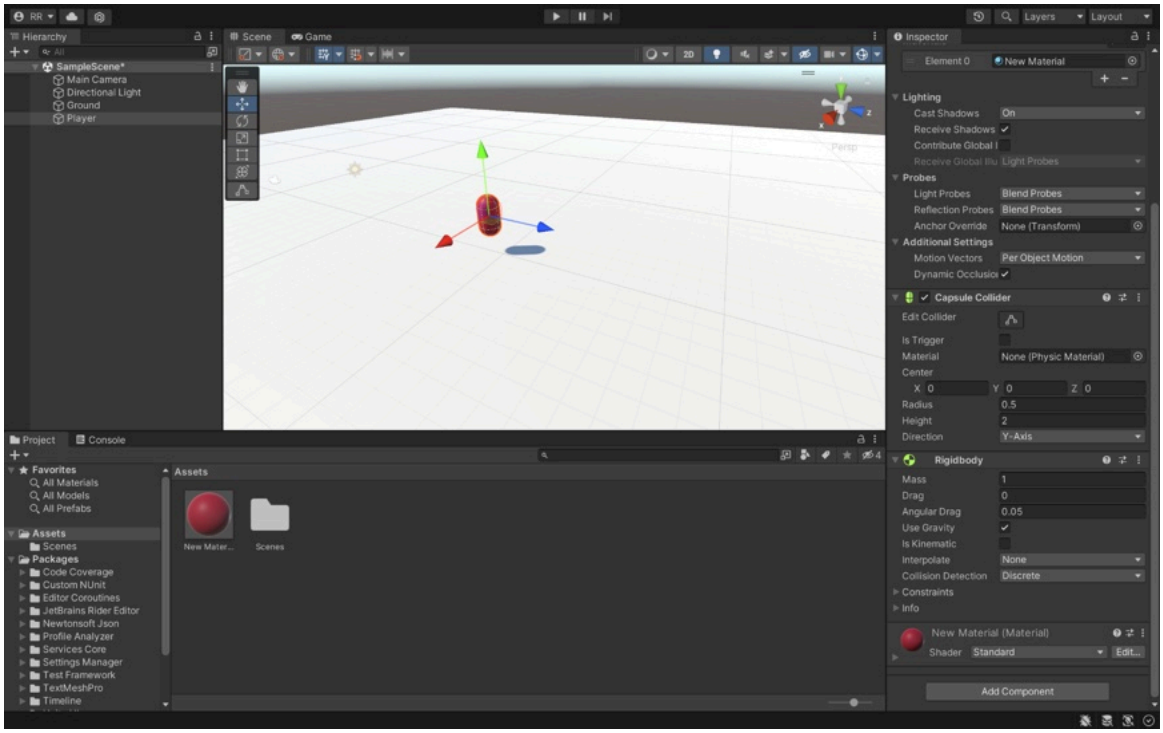
Developing the coded solution

In this stage, I will begin the development of my project by coding and documenting each section of the program, with the aid of my decomposition diagram. I will be coding in C# on the Visual Studio Code IDE and using the Unity game engine to create my game. Any error fixing necessary will be described and explained in red.

Player:

Movement**:**

I first started by adding a simple 3D plane object to temporarily represent the ground and a 3D capsule object to represent the player for my game. Both the plane and player object have a 3D collider component to prevent the player from falling through the ground. I have also applied a rigidbody component to the player, allowing it to have Unity's basic physics system, which is needed when coding the player movement.



Now I will begin programming the script for the player movement. To do this I decided to use Unity's Transform class, this will allow me to manipulate the player's position on the X and Z axis within the game.

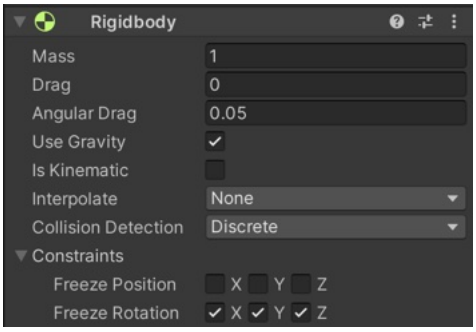
PlayerMovement.cs

Users > ryanroice > A level computing nea > Assets > PlayerMovement.cs

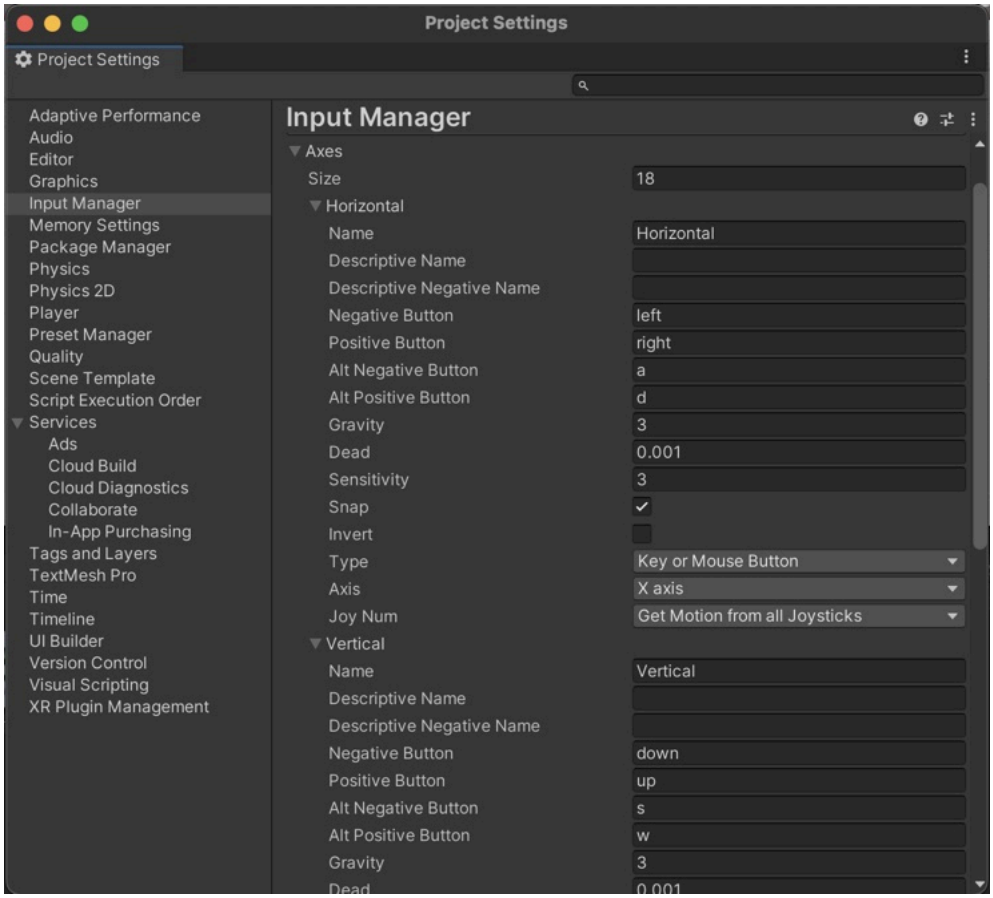
```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class PlayerMovement : MonoBehaviour
6   {
7       public float speed = 5.0f; // Determines the speed of the player
8       private float horizontalInput;
9       private float forwardInput;
10
11      // Start is called before the first frame update
12      void Start()
13      {
14
15      }
16
17      // Update is called once per frame
18      void Update()
19      {
20          //Get player input for horizontal and forward movement
21          horizontalInput = Input.GetAxis("Horizontal");
22          forwardInput = Input.GetAxis("Vertical");
23
24          // Move the player forwards or horizontally based on input
25          transform.Translate(Vector3.forward * Time.deltaTime * speed * forwardInput);
26          transform.Translate(Vector3.right * Time.deltaTime * speed * horizontalInput);
27      }
28  }
29
```

The “Input.GetAxis” method retrieves the player input for moving either horizontally or vertically. Depending on the button pressed the variables “horizontalInput” and “forwardInput” will hold either a negative or positive value.

The Translate method will move the player’s position on the x or z axis by the values held in these variables. The speed variable acts as a multiplier to the distance travelled by the player. ‘’Time.deltaTime” makes the frame rate independent, this ensures a smooth and consistent performance regardless of the maximum framerate the system can handle.



However, I encountered an issue where the player would topple over randomly whilst moving. I figured by freezing the rotation of the object in the rigidbody component, it would keep the player upright. This quickly solved the problem.



The input manager shows both the horizontal and vertical inputs which can be taken. They each have a negative and positive button.

Test	Input	Justification	Expected Outcome	Actual Outcome
Can player move forward	Press “W” key on keyboard	This allows the player to move around	The player will move forwards	The player moves forwards
Can player move backward	Press “S” key on keyboard	This allows the player to move around	The player will move backwards	The player moves backwards
Can player move to the left	Press “A” key on keyboard	This allows the player to move around	The player will move to the left	The player moves to the left
Can player move to the right	Press “D” key on keyboard	This allows the player to move around	The player will move to the right	The player moves to the right

Jumping:

PlayerMovement.cs

Users > ryanroice > A level computing nea > Assets > PlayerMovement.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class PlayerMovement : MonoBehaviour
6   {
7       public float speed = 5.0f; // Determines the speed of the player
8       public float jumpForce = 5.0f; //Determines the jump height of the player
9       private float horizontalInput;
10      private float forwardInput;
11      private Rigidbody playerRb;
12
13      // Start is called before the first frame update
14      void Start()
15      {
16          playerRb = GetComponent<Rigidbody>(); // allows access to the player's rigidbody component
17      }
18
```

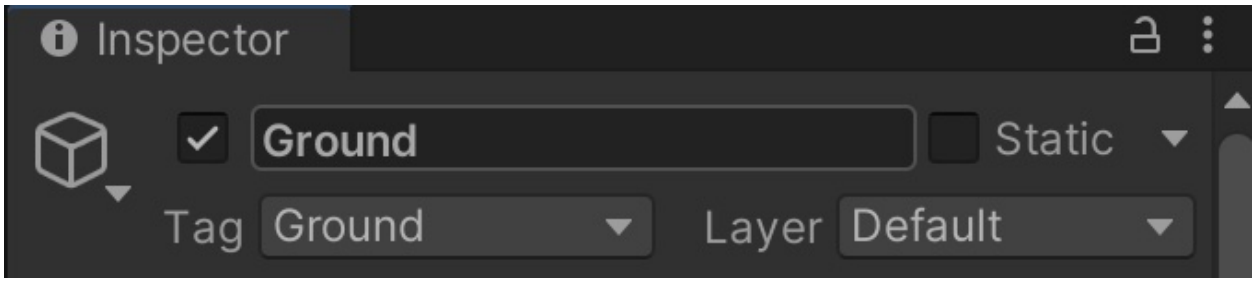
I introduced two new variables, “jumpForce” and “playerRb”. Jumpforce allows me to change the height the player jumps to and playerRb is set to the Rigidbody component of the player(line16).

```
19      // Update is called once per frame
20      void Update()
21      {
22          //Get player input for horizontal and forward movement
23          horizontalInput = Input.GetAxis("Horizontal");
24          forwardInput = Input.GetAxis("Vertical");
25
26          // Move the player forwards or horizontally based on input
27          transform.Translate(Vector3.forward * Time.deltaTime * speed * forwardInput);
28          transform.Translate(Vector3.right * Time.deltaTime * speed * horizontalInput);
29
30          //Lets the player jump
31          if(Input.GetKeyDown(KeyCode.Space))
32          {
33              playerRb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
34          }
35      }
36  }
```

I then added an If statement in the update method to continuously check whether the player presses the spacebar. When the player presses the spacebar, an upwards force is applied. The strength of the force is reliant on the value of “jumpForce”. “ForceMode.Impulse” allows the force to be applied immediately after pressing the spacebar.

Test	Input	Justification	Expected Outcome	Actual Outcome
Does the player jump?	Press Space bar on keyboard	This allows the player jump	The player will jump	The player jumps

However, I have encountered an issue. The player is able to jump multiple times despite still being in the air. This is not intended as the player must touch the ground before jumping each time.



To solve this problem I decided to check if the player is making contact with the ground before he is able to jump. Therefore I added a tag to the plane called “Ground”.

```
9       public bool isOnGround = true;
```

```
31      //Lets the player jump
32      if(Input.GetKeyDown(KeyCode.Space) && isOnGround)
33      {
34          playerRb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
35          isOnGround = false;
36      }
```

I have then added a new boolean variable called “isOnGround” which is set to true. I have also altered the if statement to include the ground check. When the player jumps, “isOnGround” is set to false which stops the player from jumping again in the air. I now need to make sure “isOnGround” is set back to true when the player touches the ground.

```
39      private void OnCollisionEnter(Collision collision)
40      {
41          if (collision.gameObject.CompareTag("Ground"))
42          {
43              isOnGround = true;
44          }
45      }
```

I have created a new method called “OnCollisionEnter”. This allows me to set the variable “isOnGround” to true whenever the player collides with an object with the tag “Ground”. This has now fixed the problem allowing the player to jump normally.

Test	Input	Justification	Expected Outcome	Actual Outcome
Does the player only jump again after touching the ground?	Press Space bar on keyboard	This means the player can jump more realistically	The player will jump only after touching the ground	The player only jumps after touching the ground

First Person View:

```
7       // sensitivites for horizontal and vertical mouse movement
8       public float SensX;
9       public float SensY;
10      //orientation of the player model that follows the camera
11      public Transform orientation;
12      //X and Y rotation of the camera
13      float xRotation;
14      float yRotation;
```

I first created 2 floats for the X and Y sensitivities.I then created a transform which holds the orientation of the player and two more floats that hold the X and Y rotation of the camera.

```
16      // Start is called before the first frame update
17      private void Start()
18      {
19          Cursor.lockState = CursorLockMode.Locked;//keeps the cursor at the center of the screen
20          Cursor.visible = false;//makes the cursor invisible
21      }
```

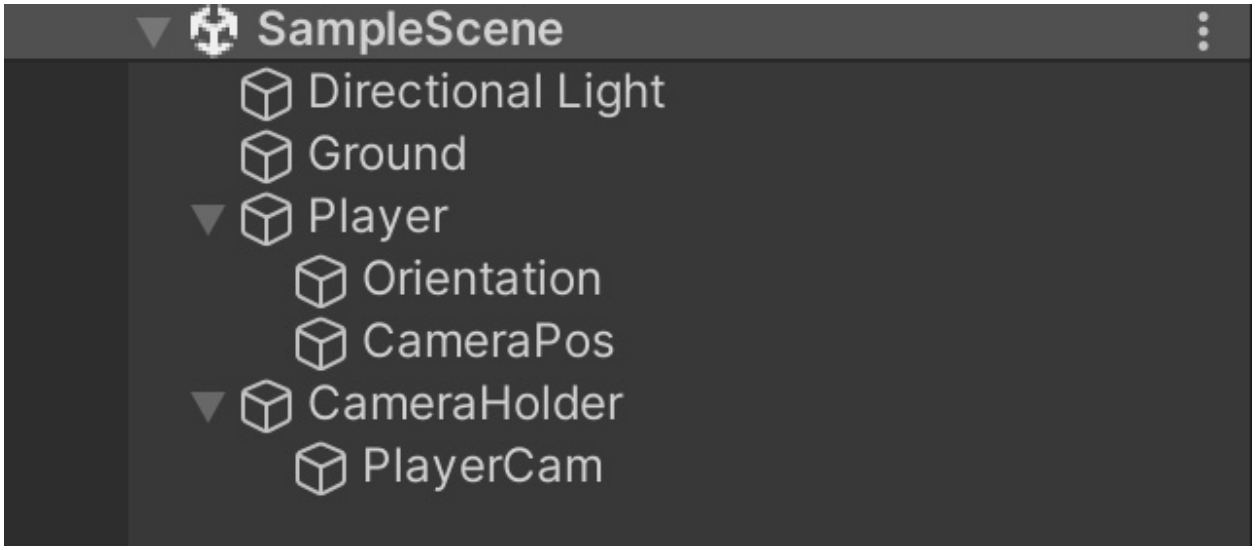
In the Start method, I made the cursor locked to the center of the screen and also made it invisible. This prevents the cursor from accidentally leaving the game window. Making the cursor invisible is also necessary as I am going to add a crosshair in the middle of the screen in the future.

```
23      // Update is called once per frame
24      private void Update()
25      {
26          // to get mouse input
27          float mouseY =  Input.GetAxisRaw("Mouse Y") * Time.deltaTime * SensY;
28          float mouseX =  Input.GetAxisRaw("Mouse X") * Time.deltaTime * SensX;
29
30          //update rotation based on mouse input
31          yRotation += mouseX;
32          xRotation += mouseY;
```

“mouseY” and “mouseX” store any new input either vertically or horizontally from the mouse. They are multiplied by SensY or SensX as this allows the speed of the camera rotation to be adjusted. These new inputs are used to update the current X and Y rotation of the camera.

```
34          // to rotate cam and orientation
35          transform.rotation = Quaternion.Euler(xRotation, yRotation, 0);
36          orientation.rotation = Quaternion.Euler(0, yRotation, 0);
```

The first line allows for the X and Y axis rotation of the camera both vertically and horizontally. The second line allows the orientation of the player to match with that of the horizontal camera rotation, which creates a more natural looking movement as the player looks around.



Finally, to implement the first person camera within the game, I attached the “PlayerCam” script to the main camera. I then put the main camera into a game object called “CameraHolder”. This is to prevent the camera from bugging out or splitting away from the player object as this tends to happen with a camera on a rigidbody object. I also created 2 more game objects within the player object, Orientation and CameraPos. Orientation is an empty game object simply used to keep track of the direction the player is facing. CamerPos is also an empty game object and is used to hold the position of the camera.

```
5   public class MoveCam : MonoBehaviour
6   {
7       public Transform cameraPosition;
8
9       // Update is called once per frame
10      private void Update()
11      {
12          transform.position = cameraPosition.position;
13      }
14  }
```

I also created a script for the camera holder so that it always updates to the position of the CameraPos game object within the player object. This allows the camera to consistently move with the player.

I now need to be able to link the movement of the player to the first person camera as currently the player always moves in the same direction regardless of where he is facing. To do this I will

adapt the PlayerMovement script to incorporate the change in direction when the orientation of the player changes.

```
27      //get forward and right directions from cam orientation
28      Vector3 forwardDirection = Camera.main.transform.forward;
29      Vector3 rightDirection = Camera.main.transform.right;
30
```

```
31      forwardDirection.y = 0f;
32      rightDirection.y = 0f;
33
34      //Normalize to ensure consistent speed in all directions
35      forwardDirection.Normalize();
36      rightDirection.Normalize();
```

The Y component of forwardDirection and rightDirection is set to 0 to ensure that the movement is only horizontal. The vectors are also normalised so movement speed is consistent regardless of the player's orientation, this means that the vector's length is always 1 even when the player is changing directions.

```
38      // Calculate the movement direction based on camera orientation
39      Vector3 moveDirection = forwardDirection * forwardInput + rightDirection * horizontalInput;
40      moveDirection.Normalize();
41
```

```
42      // Move the player
43      transform.Translate(moveDirection * speed * Time.deltaTime);
```

Test	Input	Justification	Expected Outcome	Actual Outcome
The player can look to the left	Move mouse to the left	Allows the player to look around	The player camera will rotate to the left	The player camera rotates to the left
The player can look to the right	Move mouse to the right	Allows the player to look around	The player camera will rotate to the right	The player camera rotates to the right
The player can look upwards	Move mouse up	Allows the player to look around	The player camera will rotate upwards	The player camera rotates downwards
The player	Move mouse	Allows the	The player	The player

can look downwards	down	player to look around	camera will rotate downwards	camera rotates upwards

From the results of the test, it seems that the vertical camera rotation has been inverted. Therefore, I went back to the PlayerCam and looked at all the code relating to the vertical mouse input (mouseY variable) and the vertical rotation of the camera(xRotation variable).

```
26      // to get mouse input
27      float mouseY =  Input.GetAxisRaw("Mouse Y") * Time.deltaTime * SensY;
28      float mouseX =  Input.GetAxisRaw("Mouse X") * Time.deltaTime * SensX;
```

The line declaring and assigning the mouseY variable seems to have no issues as it just takes the vertical mouse input, multiplies it by the sensitivity and makes it frame-rate independent.

```
30      //update rotation based on mouse input
31      yRotation += mouseX;
32      xRotation += mouseY;
```

The code for the horizontal rotation of the camera seems to be correct as the when the mouse moves to the right, mouseX will be positive and hence the horizontal rotation will increase. However, when the mouse moves upwards, mouseY is negative. This causes the vertical rotation to decrease, causing the camera to rotate downwards. Therefore, replacing the “+” with a ”-” in line 32 should solve the issue.

```
30      //update rotation based on mouse input
31      yRotation += mouseX;
32      xRotation -= mouseY;
33
```

Test	Input	Justification	Expected Outcome	Actual Outcome
The player can look upwards	Move mouse up	Allows the player to look around	The player camera will rotate upwards	The player camera rotates upwards
The player can look downwards	Move mouse down	Allows the player to look around	The player camera will rotate downwards	The player camera rotates downwards

The camera system seems to be functioning as expected, however there is a slight issue. The rotation of the camera is infinite. Although this isn’t an issue when the camera is rotated

horizontally, the camera can be rotated vertically upwards/downwards until the camera is flipped upside down and inverted.

```
38          //clamps vertical rotation
39          xRotation = Mathf.Clamp(xRotation, -90f, 90f);
40
```

A simple solution to this is clamping the vertical camera rotation(xRotation) between -90 and 90 degrees. This is done by using the “Mathf.Clamp()” function from Unity’s provided Mathematics functions class.

Player Health:

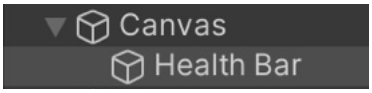
```
8       private float maxHealth;
9       private float currentHealth;
```

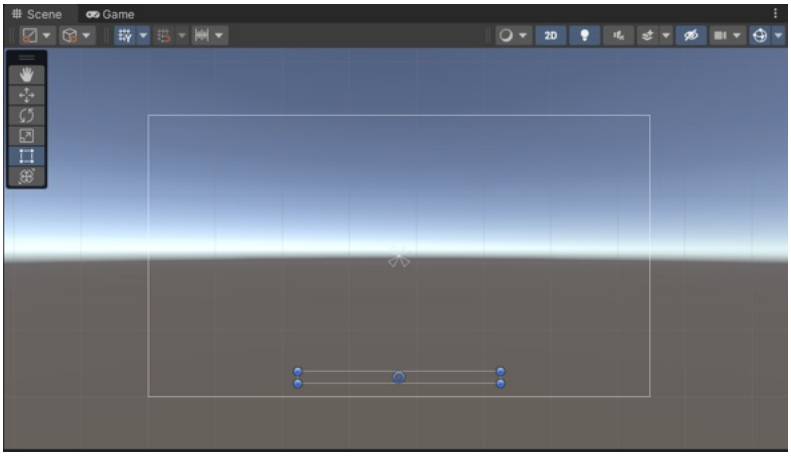
I first created two float variables, “maxHealth”, which holds the maximum health the player can have and “currentHealth”, which holds the health the player currently has in game.

```
11      // Start is called before the first frame update
12      void Start()
13      {
14          // sets player's health to full
15          currentHealth = maxHealth;
16      }
17
```

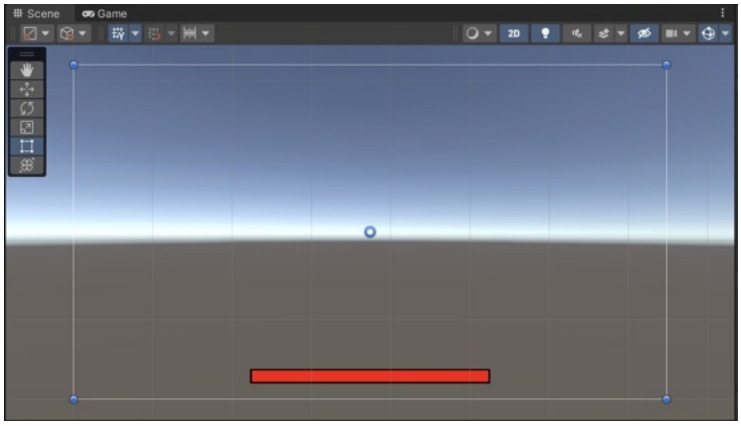
Within the “Start” function, I have set the current health of the player to the maximum health as this is the health the player will start with at the beginning of the game.

I am also going to create a simple health bar for the time being. The client may later decide if the health bar is suitable enough or would like something more sophisticated.



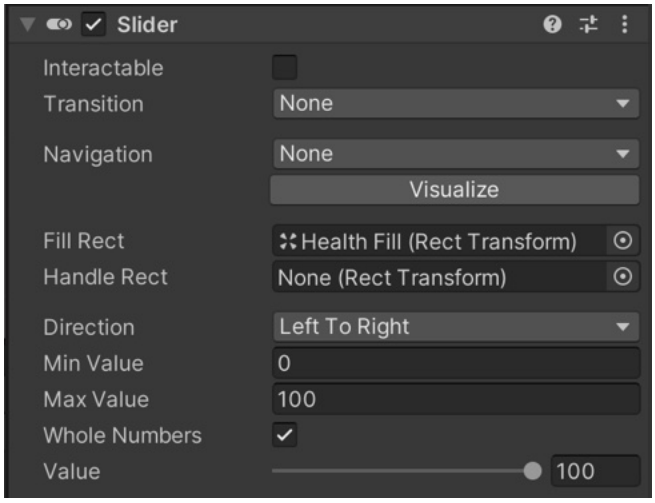


To do this, I first add a canvas game object and create an empty child object called health bar. I have then moved this child object within the canvas to the bottom of the screen and changed its shape to represent a simple health bar.



I then created two image child objects to the Health Bar object and coloured one of them red and the other black. I used an anchor preset to match the images to the shape of the health bar. I also made the red image slightly smaller so it seems like there is a black outline around the health.

The red image will be the fill for the health bar and will be dynamic as it will change size to represent the health. The black image will just be a static background for the health so that the player can always clearly see their health bar.



I also added a slider component to the “Health Bar” game object. I then referenced the health fill and also set the max value to 100 as this will be the max health. I also made the slider use whole numbers only.

I also created a new script for the health bar game object.

```
4   using UnityEngine.UI;
5
6   public class HealthBar : MonoBehaviour
7   {
8       public Slider healthSlider;
9
```

I first referenced the unity UI namespace “UnityEngine.UI” as this is necessary when developing the user interface. I also created a new variable called “healthSlider” which references the Slider component.

```
10      // updates the value of the slider
11      public void SetSlider(float amount)
12      {
13          healthSlider.value = amount;
14      }
15
16      //sets the slider to the max value
17      public void SetSliderMax(float amount)
18      {
19          healthSlider.maxValue = amount;
20          SetSlider(amount);
21      }
```

The first function “SetSlider” updates the value of the slider to the variable ”amount”. The second function sets the slider to its max value. This is done by setting “amount” to the max value of the slider and passing it as a parameter to the “SetSlider” function.

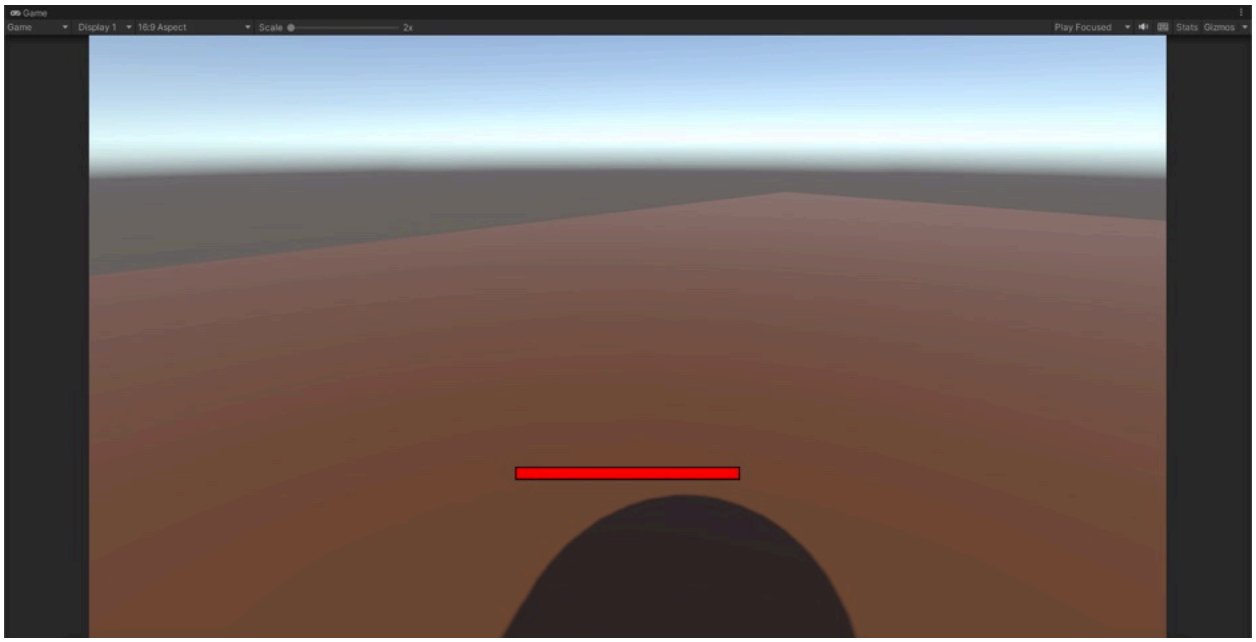
```
11      public HealthBar healthBar;
12
13      // Start is called before the first frame update
14      void Start()
15      {
16          // sets player's health to full
17          currentHealth = maxHealth;
18          //sets slider to full
19          healthBar.SetSliderMax(maxHealth);
20      }
```

I then go back into the “PlayerHealth” script and reference the “HealthBar” script. This allows me to set the slider of the health bar to “maxHealth” by using the “SetSliderMax” function. This gives the player a functioning health bar. I will add the logic for the player taking damage after implementing the enemy attack system.

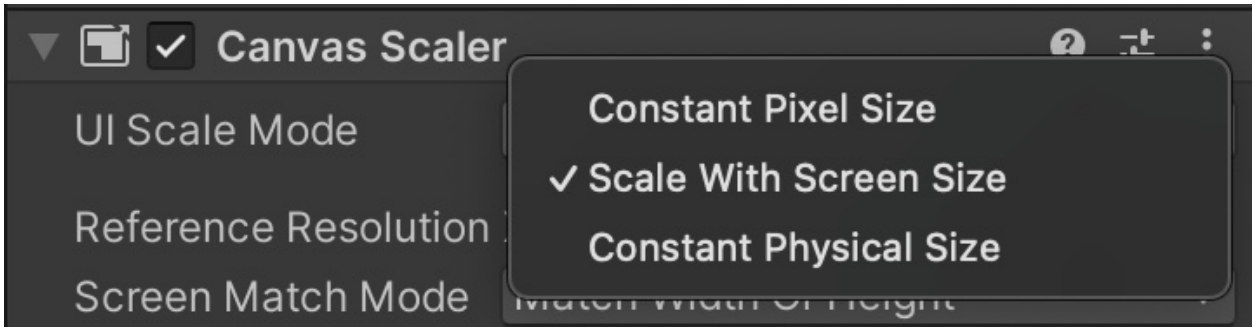
Test	Input	Justification	Expected Outcome	Actual Outcome
The health bar is	N/A	Allows the player to visually keep	The health bar will display on the	The health bar is displayed on

displayed on the screen		track of their health	screen	the screen
The health of the player matches the slider of the health bar	N/A	Improves the player's immersion as they can see the health go down when they are hit	The health bar will match the health of the player	The health bar matches the health of the player

I also encountered a slight issue where the health bar would move position based on the size of the screen. I realised this when I put the game into full screen mode and saw that the health bar was nearly in the middle of the screen.



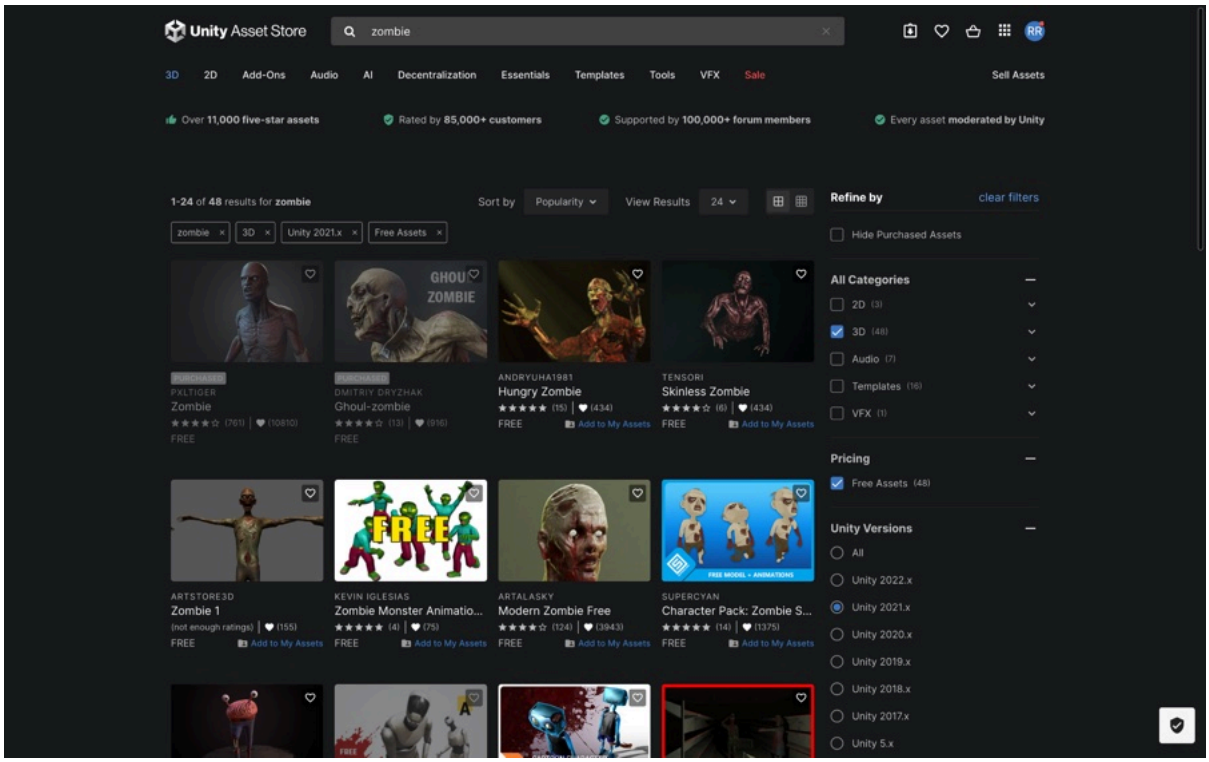
I thought that the canvas would automatically scale itself to match the size of the screen however this is a separate option which needs to be enabled. As of now my canvas is set to have a constant pixel size.



Setting this to "Scale With Screen Size" instantly fixed the issue.

Enemy

I will now start creating the enemy for my game. I first created a 3D capsule object in the hierarchy and named it “Enemy”. I now need to find a suitable 3D model for the enemy. From the analysis and design, the proposed plan is to have a zombie-like enemy. To do this I figured that the best and most efficient option would be to find a zombie 3D model from the Unity Asset store. I will also need to find animation for the zombies.



I searched for a zombie model on the Unity asset store and then set some necessary filters. The model must be 3D, it must be free and should also be compatible with my version of Unity, 2021.3.14f1. I then contacted my client, Samuel Mandoza to see what asset he would desire the most. I gave him the choice of the first 2 assets as these seem to be the most reliable and had the best reviews.
I downloaded and imported the asset into the project and added it into my game as a game object called “Enemy”.

Enemy States:

The enemy for my game will have 3 main states; patrolling, chasing and attacking. In the patrolling state, the enemy would just wander around the map. In the chasing state, the enemy would speed up and start running while following the player once they are within a certain range of the player. In the attacking state the enemy will attack once they are within range of the player.

I will also need to be using the UnityEngine.AI namespace to have access to the NavMesh system so that the enemy is able to manoeuvre through the map.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.AI;
5
6   public class EnemyAI : MonoBehaviour
7   {
8       public NavMeshAgent enemy;
9       public Transform player;
10      public LayerMask whatIsGround, whatIsPlayer;
```

Here I introduced a NavMeshAgent called “agent” . This is necessary for being able to navigate on a NavMesh. I also introduce a Transform called “Player”. I introduced 2 layer masks as well.

```
12      //For Patrolling
13      public Vector3 walkPoint;
14      bool walkPointSet;
15      public float walkPointRange;
16
17
18      //For Attacking
19      public float timeBetweenAttacks;
20      bool alreadyAttacked;
21
22      //states
23      public float sightRange, attackRange;
24      public bool playerInSightRange, playerInAttackRange;
```

The vector “walkPoint” is used to hold the point the enemy would walk to when patrolling. The boolean ”walkPointSet” indicates whether a walk point has been set or not. The float “walkPointRange”
The float “timeBetweenAttacks” holds the time interval between attacks from the enemy. The boolean “alreadyAttacked” indicates whether the enemy has attacked already or not.

The float “sightRange” is the range within which the enemy detects the player. The float “attackRange” is the range within which the enemy can attack the player. The booleans “playerInSightRange” and “playerInAttackRange” indicates if the player is within sight or attack range.

```
26      private void Awake()
27      {
28          player = GameObject.Find("Player").transform;
29          enemy = GetComponent<NavMeshAgent>();
30      }
```

I created a method called Awake() which finds the game object with the name “Player” and assigns its transform to the player variable. The variable “agent” is set to the NavMeshAgent component of the enemy.

```
45      private void Patrolling()
46      {
47
48      }
49
50      private void ChasePlayer()
51      {
52
53      }
54
55      private void AttackPlayer()
56      {
57
58      }
```

I have also created 3 more methods, one for each of the enemy states. These methods will hold the logic for each action

.

```
32      //Update is called once per frame
33      private void Update()
34      {
35          //Check for sight and attack range
36          playerInSightRange = Physics.CheckSphere(transform.position, sightRange, whatIsPlayer);
37          playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, whatIsPlayer);
38
39          //Conditions for each state
40          if(!playerInSightRange && !playerInAttackRange) Patrolling();
41          if(playerInSightRange && !playerInAttackRange) ChasePlayer();
42          if(playerInSightRange && playerInAttackRange) AttackPlayer();
43      }
```

Both “playerInSightRange” and “playerInAttackRange” use a sphere cast through the Physics.CheckSphere() function to check whether the player is within range of the enemy sight or attack. If the player is in range of sight or attacks the respect the boolean will be set to true. The CheckSphere() function works similar to a raycast however it uses a sphere to check for any collisions with an object. However, the function only detects collisions on objects with the “whatIsPlayer” layer mask.

The next three if statements determine the behaviour of the enemy based on the sight and attack ranges. The player must not be in sight and not in the enemy’s attack range for the enemy to be in the patrolling state. If the player is in sight of the enemy but not close enough to be attacked, the enemy will start chasing the player to get closer. If the player is in sight of the enemy and in range of an attack, the enemy will start attacking the player.

```
45      private void Patrolling()
46      {
47          if (!walkPointSet) SearchWalkPoint();
48
49      }
```

Within the patrolling method I checked whether the walk point is set, if it is not set, then a walk point is found through the “SearchWalkPoint” function. A walk point is a designated spot in the game where the enemy should move to.

```
60  private void SearchWalkPoint()
61  {
62      //finds random number within range
63      float randomZ = Random.Range(-walkPointRange, walkPointRange);
64      float randomX = Random.Range(-walkPointRange, walkPointRange);
65
66      //calculates new walkpoint
67      walkPoint = new Vector3(transform.position.x + randomX, transform.position.y, transform.position.z + randomZ);
68  }
```

ALSO MAY BE A PROBLEM WHEN SPAWNING ON THE NEW MAP DUE TO INCONSISTENCIES IN HEIGHT

In this function, a random walk point is found by finding a random number within the walk point range for the X component and the Z component. The walk point is then set to a new vector where the random x and z values are added to the current x and z coordinates of the enemy. The y coordinate remains unchanged since the map is a flat surface, this allows the enemy to move smoothly while staying at the same level.

```
45      private void Patrolling()
46      {
47          if (!walkPointSet) SearchWalkPoint();
48
49          if (walkPointSet)
50              enemy.SetDestination(walkPoint);
51
52          //calculates distance to walkpoint
53          Vector3 distanceToWalkPoint = transform.position - walkPoint;
54
55          //Walk point reached
56          if(distanceToWalkPoint.magnitude < 1f)
57              walkPointSet = false;
58
59      }
```

I then added the code for what would happen if the walk point is already set. The vector distance to the walk point is calculated by subtracting the walk point from the current position of the enemy. Once the enemy is within 1 magnitude of the walk point, the “walkPointSet” is set back to false, causing the “SearchWalkPoint” function to be called to find the next walk point.

```
71      private void ChasePlayer()
72      {   //makes enemy chase player by its position
73          enemy.SetDestination(player.position);
74          //make sure that the enemy looks at the player
75          tansform.LookAt(player);
```

The method for chasing the player is much simpler. The “SetDestination” is a method from the NavMeshAgent system and it sets the destination for the enemy to the position of the player, causing the enemy to follow the player. The “LookAt” function allows the enemy to constantly look at the player. The NavMesh system takes care of the work required for the pathfinding of the enemy, while also taking into account any obstacles. This will be useful when the map for my game is more developed.

```
78      private void AttackPlayer()
79      {
80          //make sure enemy doesn't move
81          enemy.SetDestination(transform.position);
82          //make sure that the enemy looks at the player
83          transform.LookAt(player);
84      }
```

For the attacking method, the enemy’s destination is set to its current position, this stops the enemy from moving while attacking the player. The “LookAt” function makes the enemy face the player, making the attacks much more realistic.

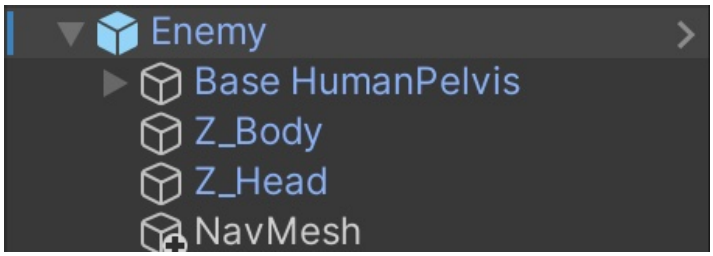
```
76      private void AttackPlayer()
77      {
78          //make sure enemy doesn't move
79          enemy.SetDestination(transform.position);
80
81          transform.LookAt(player);
82
83          if(!alreadyAttacked)
84          {
85              alreadyAttacked = true;
86              //causes enemy to attack after specified time
87              Invoke(nameof(ResetAttack), timeBetweenAttacks);
88          }
89      }
90
91      //resets the enemy attack
92      private void ResetAttack()
93      {
94          alreadyAttacked = false
95      }
```

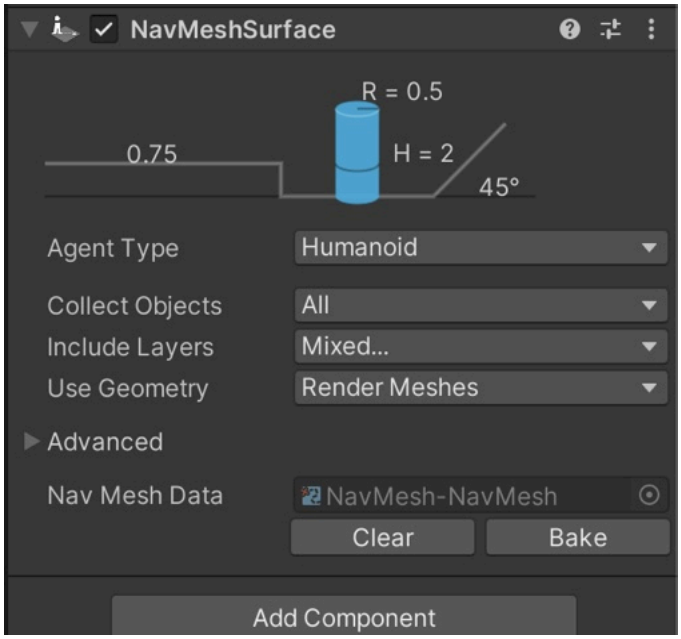
If the enemy hasn't attacked yet, "alreadyAttacked" is set to true once the enemy attacks. The "Invoke" function calls the "ResetAttack" function and schedules the next attack using the "timeBetweenAtttacks" variable. This introduces an alterable cooldown between attacks for the enemy. The "ResetAttack" function resets the attack by changing the "alreadyAttacked" boolean back to false.

I will program the logic for the enemy attack while doing the animations for the enemy as this would make more sense.

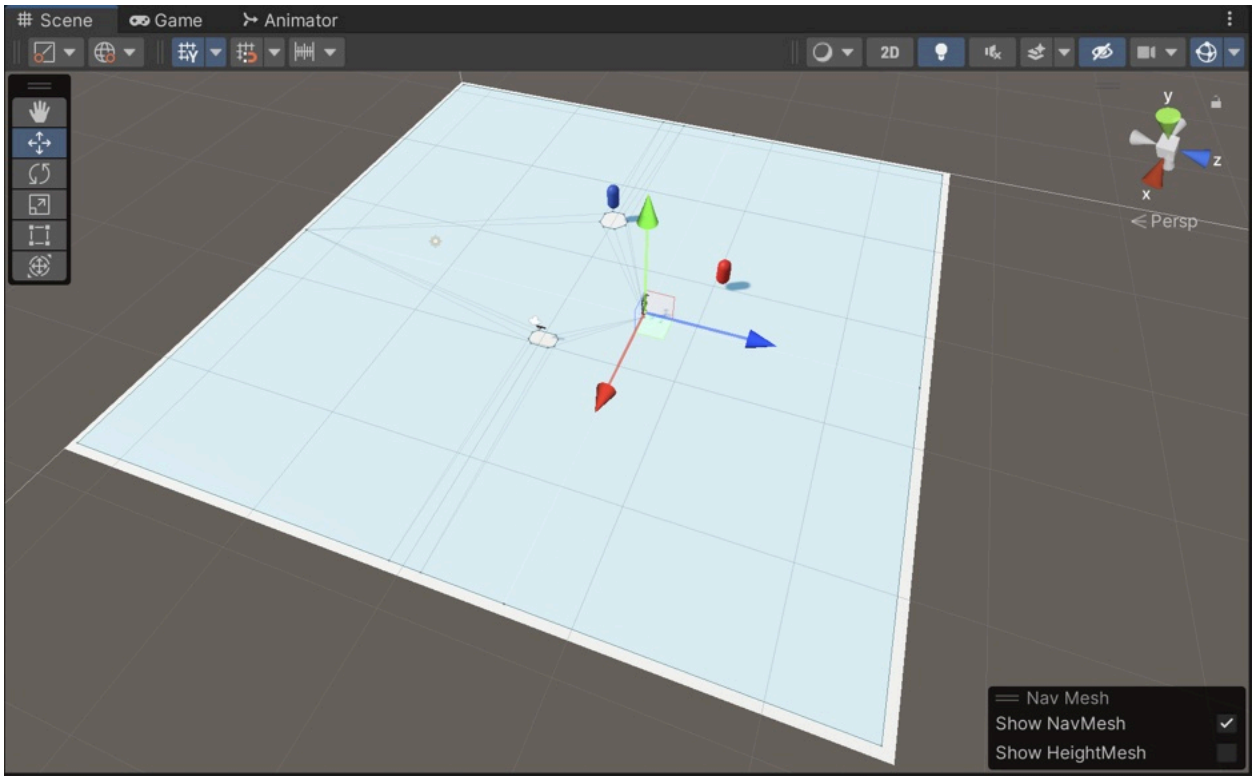
Now I need to set up the NavMesh system for the enemy so that the zombie can traverse the map properly, I can then add the respective animation states. Firstly, I added the Nav Mesh Agent component to the "Zombie" game object

I then created a new empty game object within the Zombie called “NavMesh”, which will only hold the “NavMeshSurface” component.



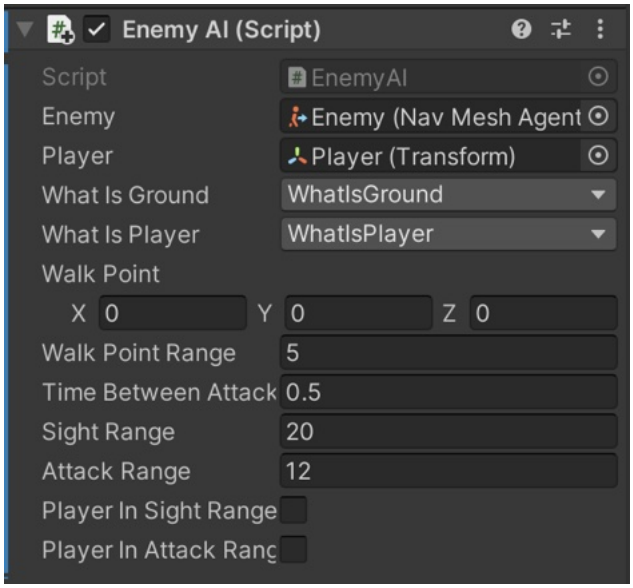


The “NavMeshSurface” component allows me to define the areas that should be included in the NavMesh”. I can also choose what layers should be included in the NavMesh, in my case, I will need to add a layer to the ground, called “WhatIsGround”.



This is the resulting NavMesh after pressing “Bake”, since I have added the layer “WhatIsGround” to the ground and given the layer as a parameter within the “NavMeshSurface" component.

I then added the EnemyAI script to the Enemy game object.

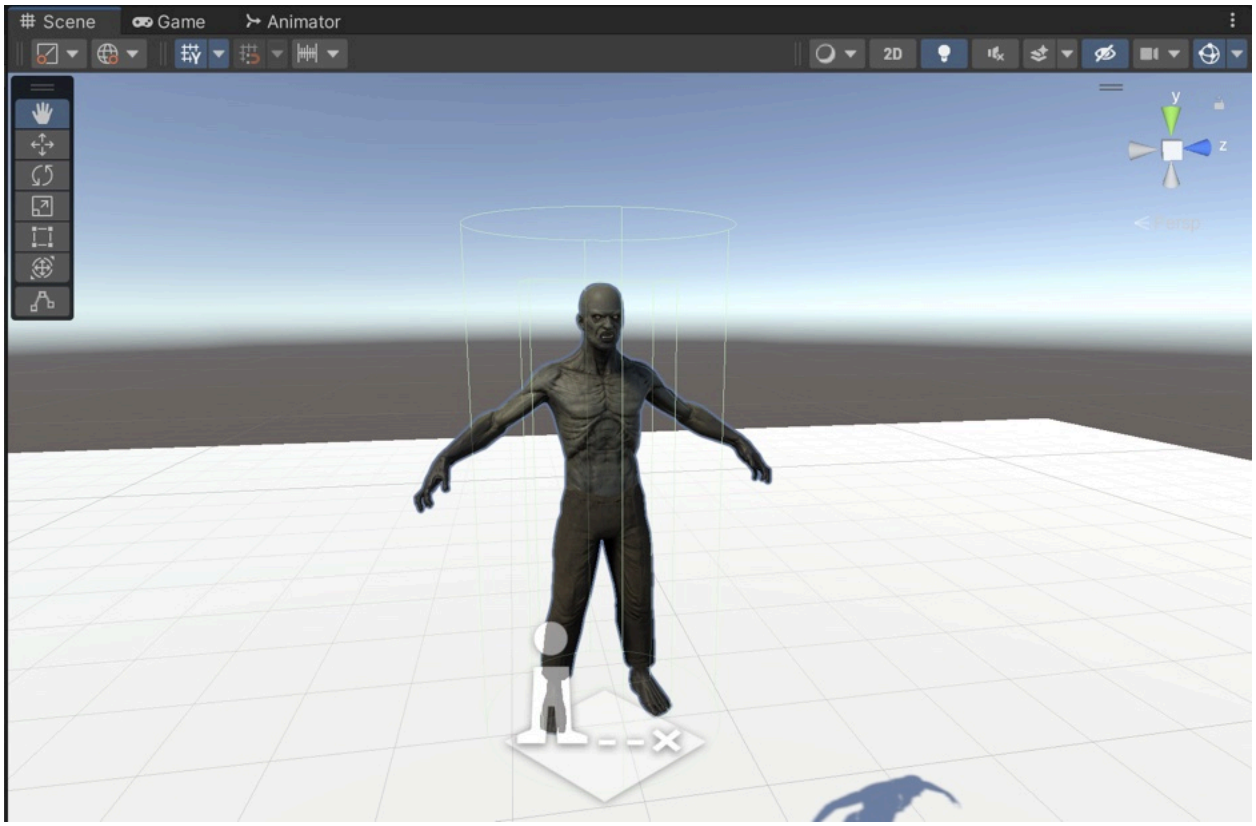


I provided the Enemy object which has the “NavMeshAgent” component and the Player object as this is who the enemy is targeting. I then set the ground layer “WhatIsGround” and also added a new layer to the Player object called “WhatIsPlayer”. I also set some values for the different ranges and the attack rate. Now I will need to test the functionality of the NavMesh system for my enemy.

Test	Input	Justification	Expected Outcome	Actual Outcome
The enemy moves around when player is completely out of range(sight and attack)	N/A	Allows the enemy to function properly	The enemy will move around randomly	The enemy falls through the ground
The enemy chases when player is in sight range but out of attack range	N/A	Allows the enemy to function properly	The enemy will chase the player until the are out of range	The enemy falls through the ground
The enemy attacks when player is completely in range(sight and attack)	N/A	Allows the enemy to function properly	The enemy will attack the player until they are out of attack range	The enemy falls through the ground

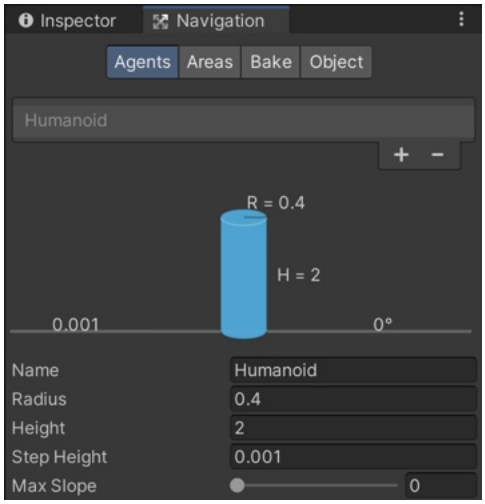
The enemy faces the direction of the player	N/A	This makes the enemy more realistic	The enemy will look at the player	The enemy looks at the player

This was completely unexpected, however I quickly realised that the enemy model was missing a collider of some sort. Therefore I decided to add a simple box collider for the enemy.

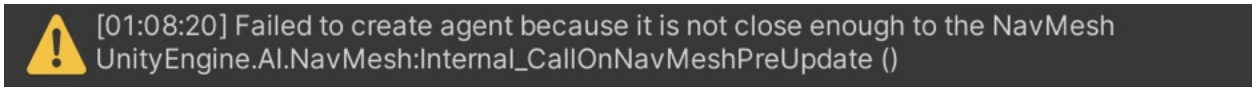


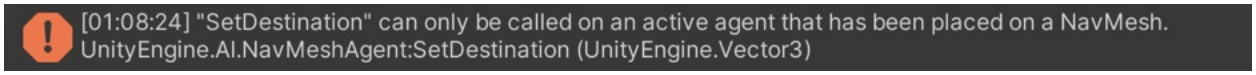
The inner square is the box collider for the enemy and the outer cylinder is the NavMeshAgent for the enemy. After adding the box collider I was certain that the issue would be fixed since it would collide with the collider of the plane, preventing it from falling through. However this issue still persisted.

My second thought was that it had something to do with the baked NavMesh as this is what controls the area where the zombie is able to walk. Since my ground is completely flat and contains no steps or slopes, I decided to put these values to 0 when baking my NavMesh.

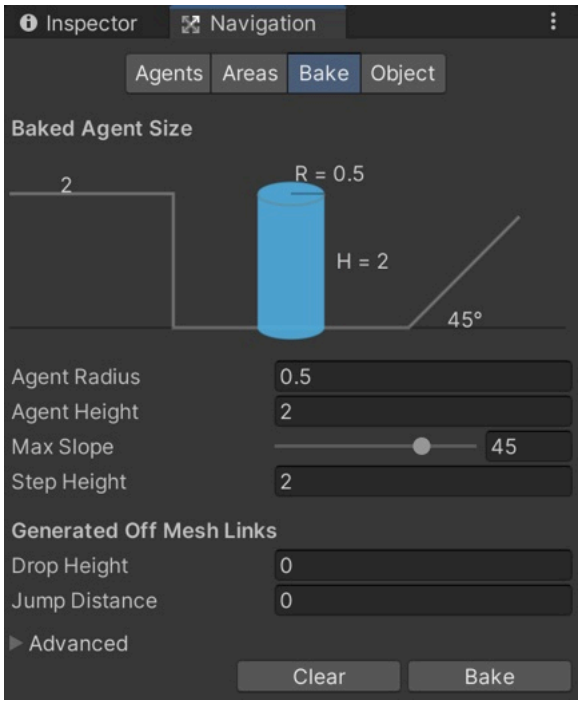


As you can see both the max slope has been set to 0 and the step height minimum was 0.001. I then baked a new NavMesh and tried running the game again, however I was reported issues within the console multiple times.





These errors both suggest there is a problem with the NavMesh. Hence, I continued exploring the NavMesh system in Unity and found this tab called “Bake”.

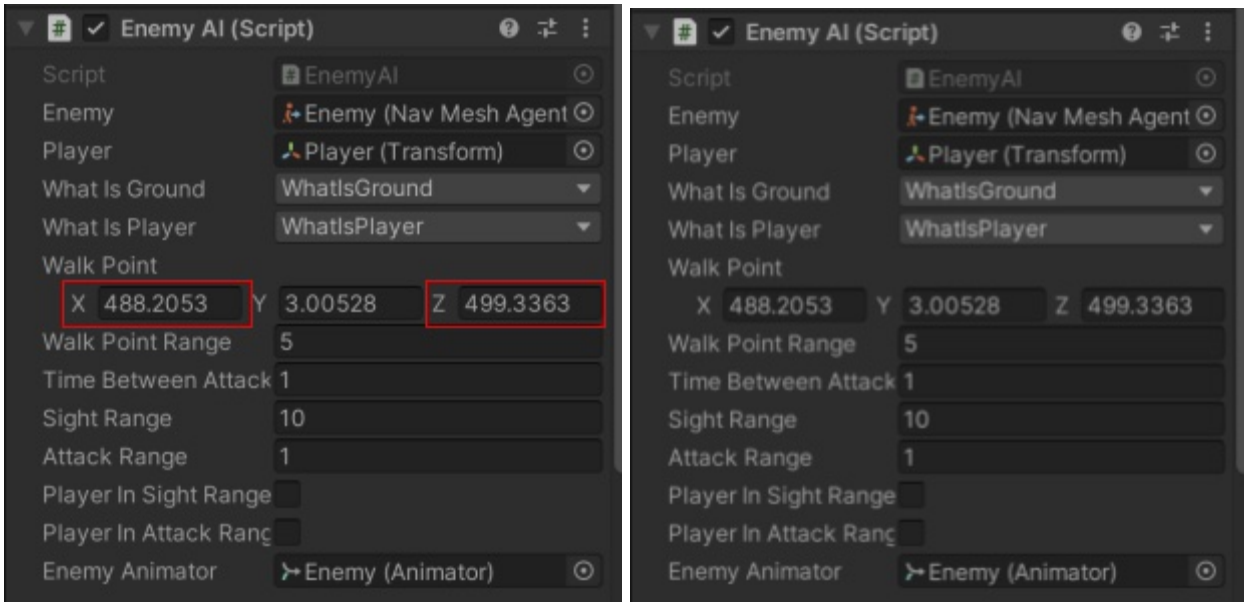


For some reason the step height and max slope was not both set to 0 even though I changed it within the “Agents” tab. Therefore it was still baking a NavMesh with the incorrect step height and slope. So after setting these to 0 again, I deleted the previous NavMesh and baked a new one with the right settings and tested again.

Test	Input	Justification	Expected Outcome	Actual Outcome
The enemy moves around when player is completely out of range(sight and attack)	N/A	Allows the enemy to function properly	The enemy will move around randomly	The enemy stands still
The enemy chases when player is in sight range but out of attack range	N/A	Allows the enemy to function properly	The enemy will chase the player until the are out of range	The enemy follows the player
The enemy attacks when player is completely in range(sight and attack)	N/A	Allows the enemy to function properly	The enemy will attack the player until they are out of attack range	The enemy does not attack

For some reason the zombie did not move at all when it was in its patrolling state, yet it still continued to look at the player. The zombie did start moving towards the player when the player was in the zombie’s sight range and …in the attack range.

The zombie not moving at all was very unusual. Furthermore, within the Enemy AI script component for the zombie object, I recognised that a random number was constantly being generated for the X and Z coordinates of the walkpoint.



This means that a new walkpoint for the zombie to go to was being calculated but the zombie was not actually moving to this point. Therefore I decided to look through the script for the zombie.

```
65    private void SearchWalkPoint()
66    {
67        //finds random number within range
68        float randomZ = Random.Range(-walkPointRange, walkPointRange);
69        float randomX = Random.Range(-walkPointRange, walkPointRange);
70
71        //calculates new walkpoint
72        walkPoint = new Vector3(transform.position.x + randomX, transform.position.y, transform.position.z + randomZ);
73
74        if (Physics.Raycast(walkPoint, -transform.up, 2f, whatIsGround))
75            walkPointSet = true;
76    }
```

I realised I missed a key statement within the “SearchWalkPoint()” function. This statement first checks if the new walk point is on the actual map, this is crucial as if the new point is outside the map, the zombie will just fall off the map. If this statement is true, “walkPointSet” is set to true which triggers the zombie to start moving.

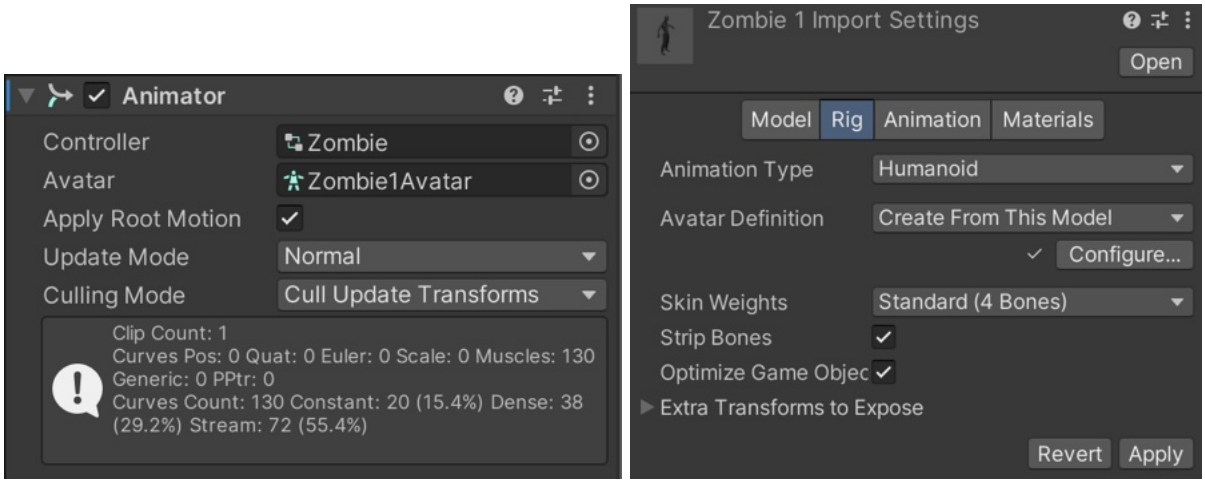
Test	Input	Justification	Expected Outcome	Actual Outcome
The enemy moves around when player is completely out of range(sight and attack)	N/A	Allows the enemy to function properly	The enemy will move around randomly	The enemy moves around randomly

The enemy attacks when player is completely in range(sight and attack)	N/A	Allows the enemy to function properly	The enemy will attack the player until they are out of attack range	The enemy does not attack

This is simply because I have not implemented the logic for the enemy attacking yet.

Animations:

Now I need to implement the animations for the zombie as well as the logic for the zombie melee attack. To do this I will need to use Unity's built-in animator component and the animator controller. First I added the animator component to the enemy and referenced an animator controller that I created called "Zombie". I also referenced the avatar of the zombie which I created through the FBX of the zombie.



Now within the animator controller, I firstly just placed down the walk animation for the zombie. I also ensured that the animation was looped as the zombie is continuously walking.

At this point the zombie is continuously walking regardless of the state it is in. I also adjusted the speed of the animation as well as the angular speed of the Nav Mesh Agent so that the animation is played smoothly and is realistic.

Now I will begin scripting the animation for the enemy. I first referenced the animator component of the enemy.

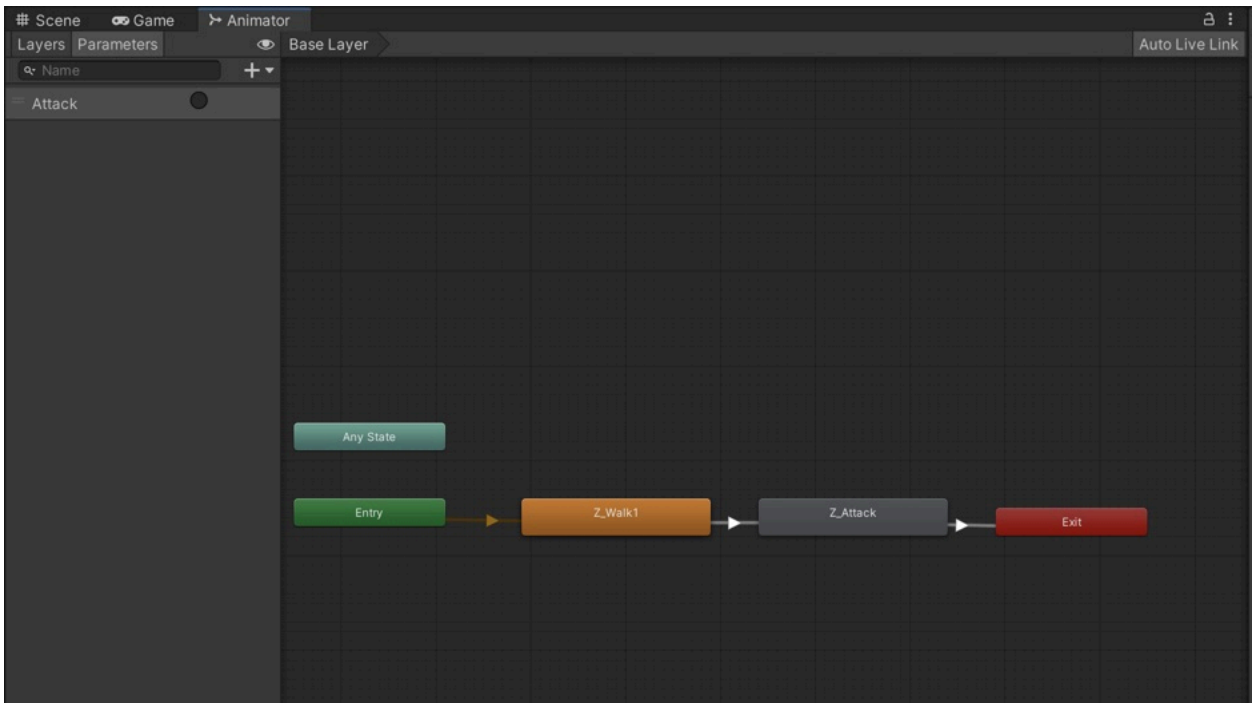
```
26      public Animator enemyAnimator;
```

```
32      enemyAnimator = GetComponent<Animator>();
```

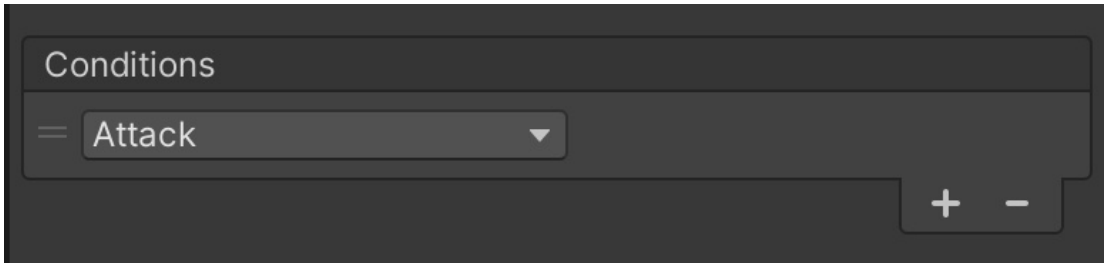
The animator component of the enemy is set to the variable “enemyAnimator” so that I can access it.

```
86      //triggers the attack animation
87      animator.SetTrigger("Attack");
```

This line is used to trigger the “Attack” parameter within the animator controller.



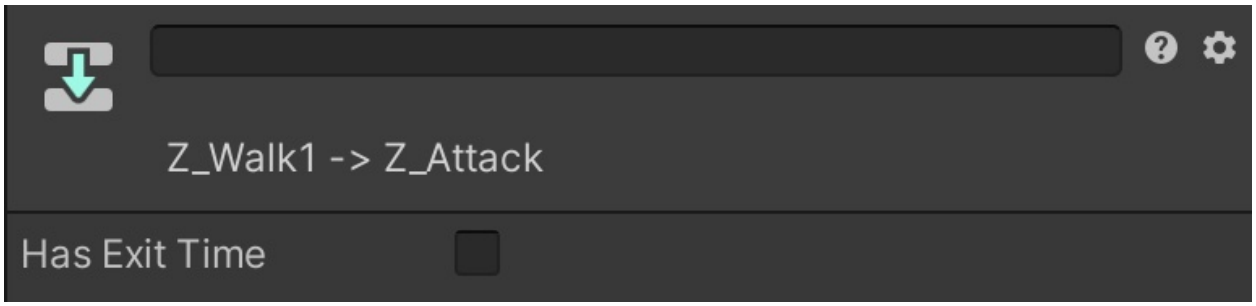
I have now added a new parameter called “Attack” and set it as a trigger. I also added the attack animation for the zombie into the animator controller. I have also made 2 extra transitions, one from the walk animation to the attack animation and one from the attack animation to the exit.



I also set a condition in the transition between the walk animation to the attack animation. This means that the attack animation is only carried out once triggered in the “AttackPlayer” function.

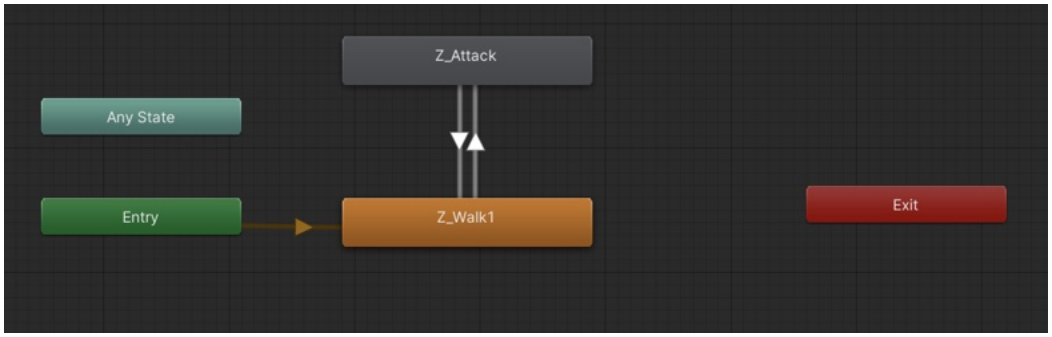
Test	Input	Justification	Expected Outcome	Actual Outcome
The enemy attacks the player when in the attack range	Move into attack range	Allows the enemy to function properly	The enemy will attack the player once they are in the attack range	The enemy attacks the player in the attack range

I figured that although the transitions between the different animations work, there are some problems that make the enemy unrealistic. For starters, the enemy has a delay between transitioning from the walking animation to the attacking animation.

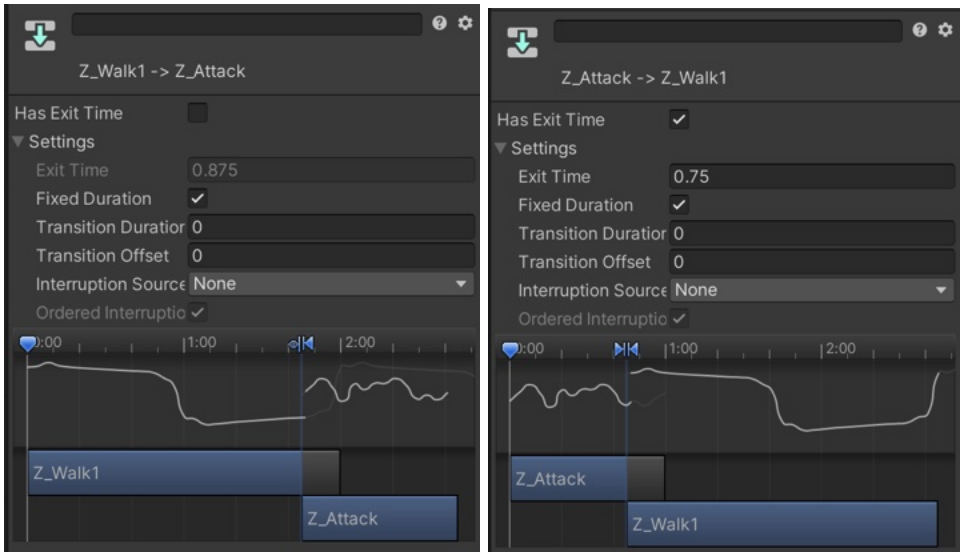


To fix this, I decided to disable the “Has Exit Time” option within the transition from the walk to the attack animation. This quickly solved the issue as the walk animation did not have to finish before transitioning to the attack animation.

I also saw that the attack animations were getting cut off before completely looping. I first decided to change the layout of the animator controller so that the attack animation transitions back into the walk animation instead of exiting.



Although this did make the transition between animations slightly smoother, the attack animation was still being cut off. I decided to experiment with the transition duration between the animations.



I realised that setting this to 0 meant that the animation instantly switches without any blending between the animations (this can be seen in the diagram at the bottom of both pictures). Therefore, it was a little unrealistic but this was much better as the attack animation is now fully playing.

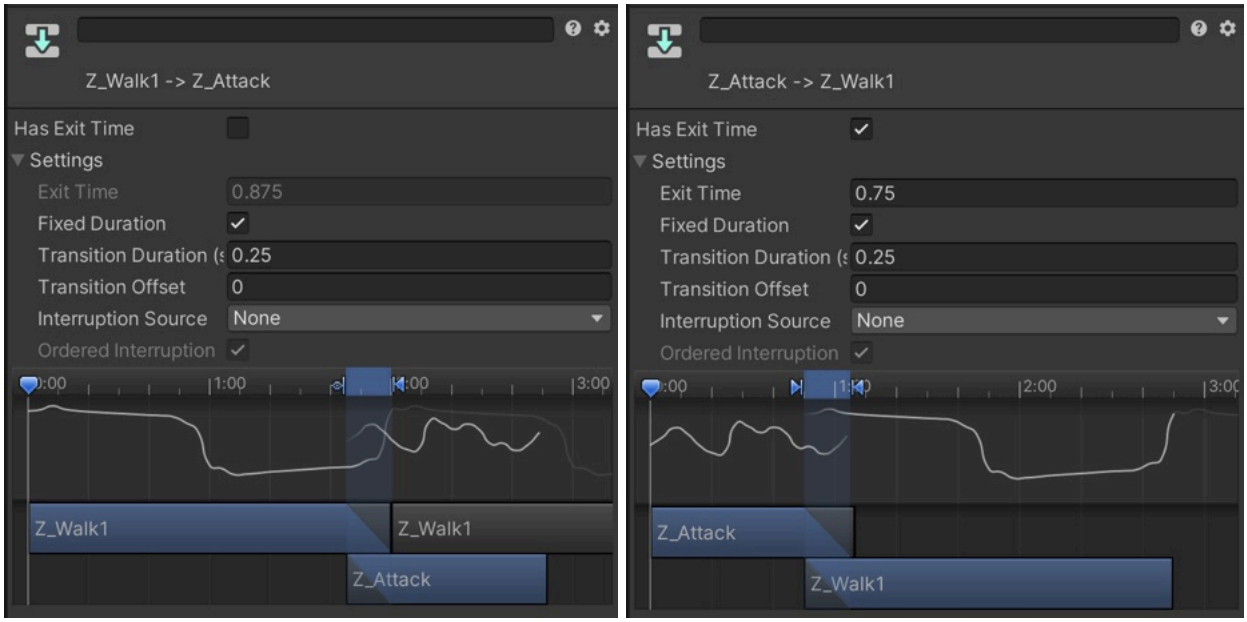
Another issue I encountered was that the enemy would always attack twice, even if the player was not in the attack range for the second attack. My first thought was to decrease the attack range of the enemy from 5 to 1, so that the player has to be closer to get attacked by the enemy. However the issue still persisted.

I then realised that the zombie may have been attacking twice but the second attack was being delayed due to the first attack. Therefore I decided that the enemy should switch back to the walk animation after an attack to prevent a second attack from initiating.

```
85          if (!enemyAnimator.GetCurrentAnimatorStateInfo(0).IsName("Z_Attack"))
86          {
87              //triggers the attack animation
88              enemyAnimator.SetTrigger("Attack");
89              //make sure enemy doesn't move
90              enemy.SetDestination(transform.position);
91          }
```

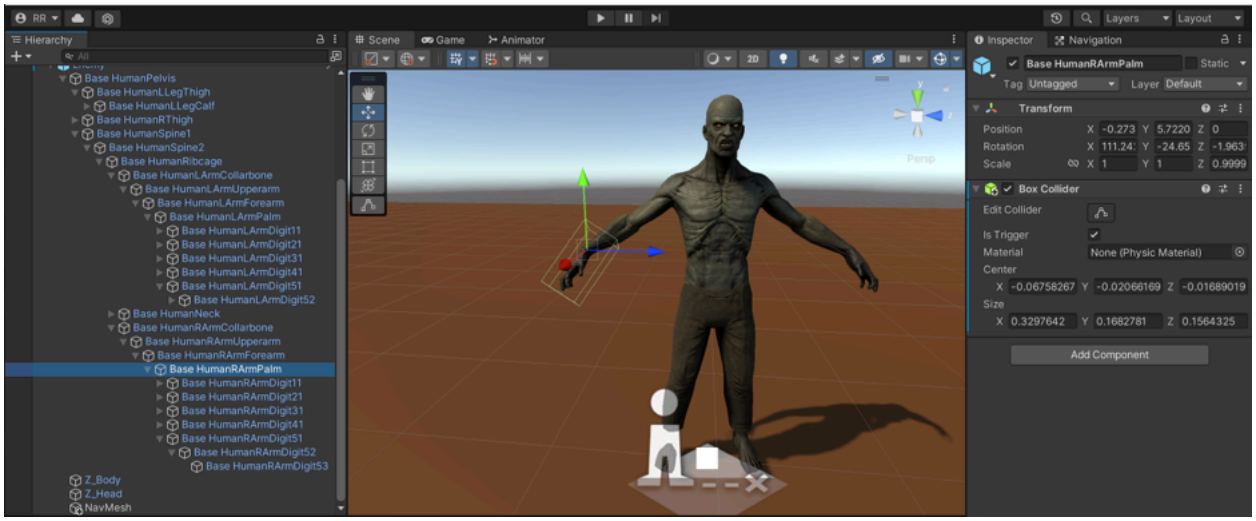
This if statement means that the current animation should not be the attack animation within the zombie controller for the attack to be triggered again. Therefore the enemy must transition back to the walking animation to trigger the attack animation again. This finally solved the issue.

This condition also meant that the attack animation must be fully played before being played again. This means that I can add a transition duration between animations without it causing any issues of cutting off the attack animation.



I have now introduced a small transition duration between the attack and walk animations. This makes the transitions between animations much more smoother and realistic.

I am now going to create the ability for the enemy to actually damage the player. To do this I first needed to make a box collider around the zombie's hand. This is done by going into the prefab of the zombie and looking through the child objects until I find the hands of the zombie.



I found the right hand of the zombie and added a box collider as seen above. I decided to use the right hand even though the zombie swings both hands when it attacks since the right hand attacks first. I then adjusted the box collider to a suitable size and made it a trigger.

```
13      public BoxCollider boxCollider;
14
```

```
34      boxCollider = GetComponentInChildren<BoxCollider>();
```

I first need to reference the box collider for the enemy hand. I did this by creating a box collider variable and then giving it the value of the box collider component. I used “GetComponentInChildren” instead of “GetComponent”as each body part of the zombie is seen as its own game object. This is also possible since there are no other child objects with a box collider.

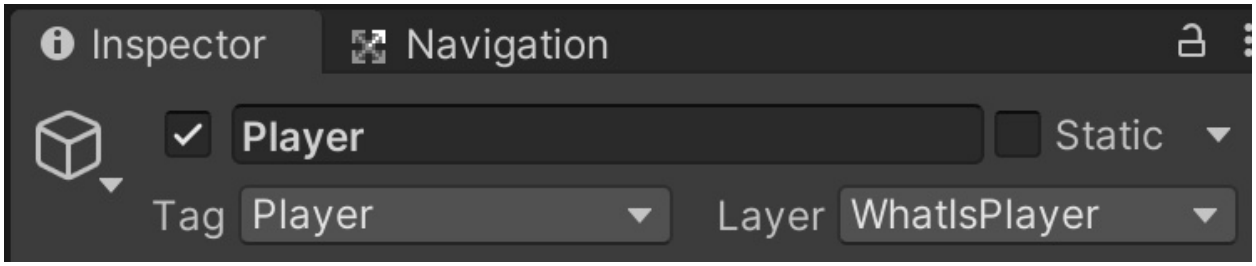
```
110     //turns the box collider on
111     private void EnableAttack()
112     {
113         boxCollider.enabled = true;
114     }
115
116     //turns the box collider off
117     private void DisableAttack()
118     {
119         boxCollider.enabled = false;
120     }
```

I then created two new functions, “EnableAttack” and “DisableAttack”‘. These functions are used to turn on and off the box collider on the enemy hand. This is necessary as it prevents the player getting damaged if they have walked into the enemy and hit the collider. The collider will only be active during the attack animation.

Now I need to handle the collision between the enemy and the player.

```
104     private void OnCollisionEnter(Collision collision)
105     {
106         if (collision.gameObject.CompareTag("Player"))
107         {
108             print("HIT!");
109         }
110     }
```

I first checked if the collision between the enemy and the player was functioning correctly by checking if “HIT!” was printed when the enemy attacked.



I first created a new tag called “Player” and set it to the player. This is necessary as it is needed to for the

Test	Input	Justification	Expected Outcome	Actual Outcome
The enemy successfully collides with the player when attacking	N/A	Allows the enemy to attack the player	The enemy will attack when the player is in the attack range and “HIT!” is printed in the console	The enemy attacks the player and “HIT!” is not printed

I encountered an issue where the “HIT!’ message was not playing when the enemy’ collider in their hand collided with the player. However I realised that “HIT!” was being printed when I walked into the enemy without even touching the enemy’s hand. Furthermore, there were no error messages in the console, therefore it was a logic error.

After some research, I learned that collisions between colliders use a different function if one of the colliders was a trigger. I needed to use the “OnTriggerEnter” function instead of “OnCollisionEnter”. I also need to use “Collider” as a parameter instead of “Collision”.

```
104     private void OnTriggerEnter(Collider collision)
105     {
106         if (collision.gameObject.CompareTag("Player"))
107         {
108             print("HIT!");
109         }
110     }
```

Test	Input	Justification	Expected Outcome	Actual Outcome

The enemy successfully collides with the player when attacking	N/A	Allows the enemy to attack the player	The enemy will attack when the player is in the attack range and “HIT!” is printed in the console	The enemy attacks the player and “HIT!” is printed

Now to actually damage the player I will need to go back to the “PlayerHealth” script and adjust the health and slider by the correct amounts.

```
22      public void TakeDamage(float amount)
23      {
24          currentHealth -= amount;
25          healthBar.SetSlider(currentHealth);
26      }
```

Within the script, I created a new function called “TakeDamage" which subtracts an amount from the current health of the player and also updates the health bar slider to match this value. I now need to make sure the player only loses health when hit by the zombie’s attack.

```
21      //For Attacking
22      public float timeBetweenAttacks;
23      public float damage;
24      bool alreadyAttacked;
```

Within the “EnemyAI” script I created a new variable called “damage” which will hold the damage the enemy inflicts when the player is hit by the enemy’s attack.

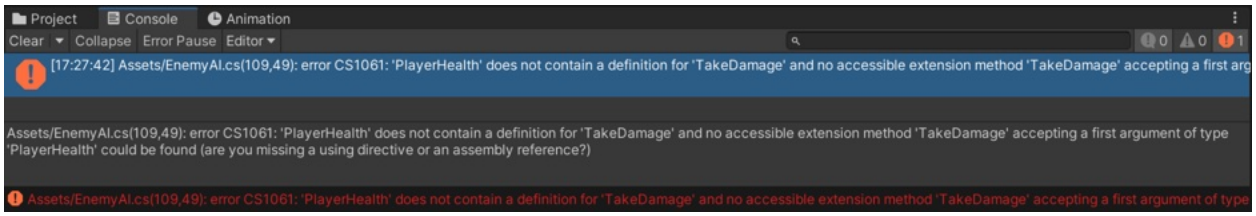
I will now replace the temporary “Hit!” which was being printed to the console and make it so that the player actually takes damage.

```
105     private void OnTriggerEnter(Collider collision)
106     {
107         if (collision.gameObject.CompareTag("Player"))
108         {
109             player.GetComponent<PlayerHealth>().TakeDamage(damage)
110         }
111     }
```

Within the same “OnTriggerEnter” function, I replace the temporary “print(“HIT!”)” with a new line. This line checks the components of the player object, which contains the “PlayerHealth” script and uses the “TakeDamage” function from this script. The variable “damage”, which I have currently set to 5, is taken as a parameter. Therefore, everytime the player gets hit by the enemy, the player loses 5 health points.

Test	Input	Justification	Expected Outcome	Actual Outcome
The enemy successfully damages the player when attacking	N/A	Allows the enemy to hurt the player	The enemy will attack when the player is in the attack range and the player loses health	The game does not run
The player’s health bar goes down when they lose health	N/A	Allows the player to visually keep track of their health	The health bar will go down after getting hit by the enemy	The game does not run

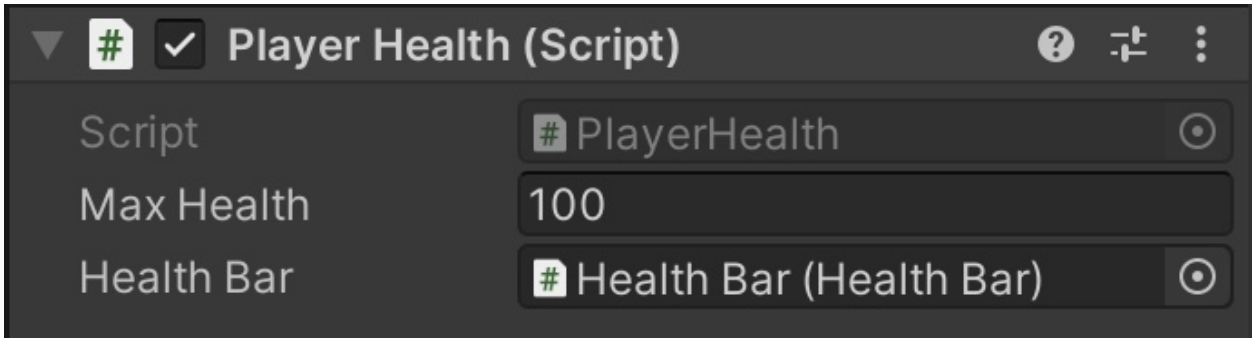
I encountered an unexpected issue:



For some reason Unity could not see the “TakeDamage” function I was referencing from the “PlayerHealth” script‘. This was unusual as I went back into the “PlayerHealth” script to ensure that the function had no syntax/logic errors. The function seems to be perfectly fine.

I then checked the new line I added into the “EnemyAI” script as this where the error in the console points to. However again, there seems to be no errors with the statement.

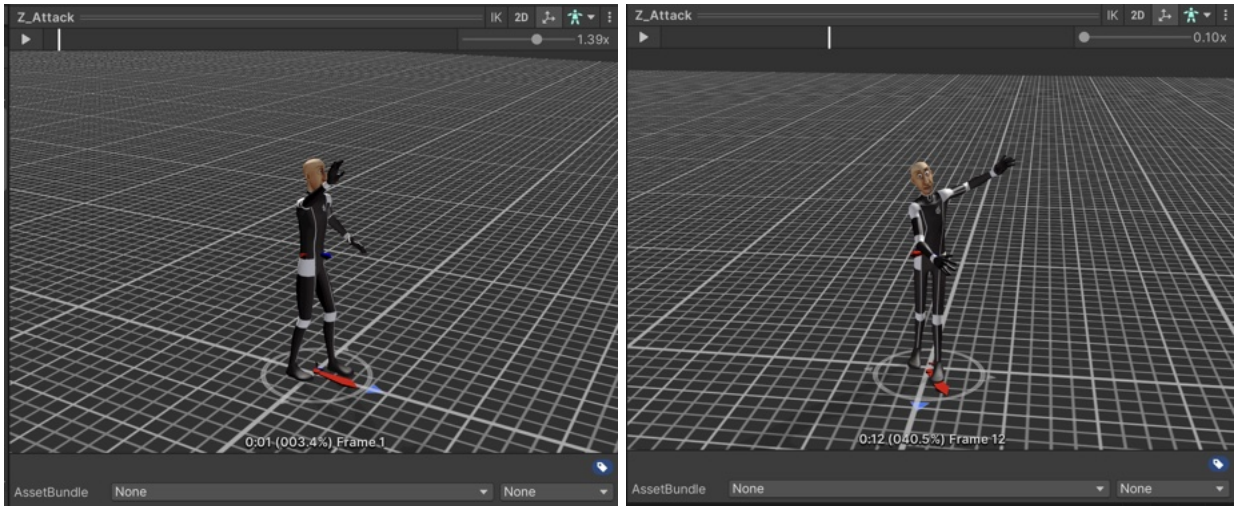
My next thought was that I referenced the player component incorrectly. I thought that instead of “.GetComponent<PlayerHealth>”, it may be “.GetComponent<Player Health>”. This was because although the script was called “PlayerHealth”, the name of the actual component for the script in the inspector was “Player Health”.



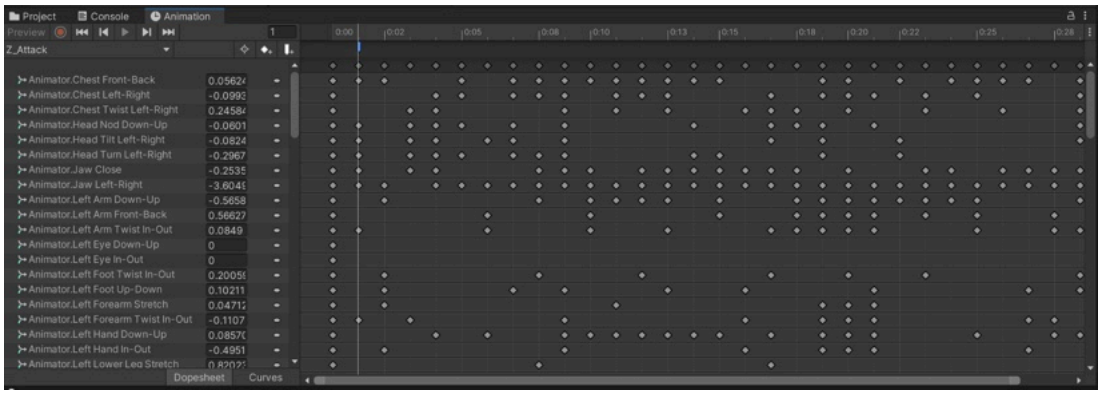
However this led to no avail as the issue still persisted. My final option was to just rewrite the "PlayerHealth" and the new line in the "EnemyAi" script, as well as any related components. This somehow magically fixed the error, even though I did the exact same thing and wrote the exact same script again. Hence I believe this must just be a bug that occurred within Unity itself.

Test	Input	Justification	Expected Outcome	Actual Outcome
The enemy successfully damages the player when attacking	N/A	Allows the enemy to hurt the player	The enemy will attack when the player is in the attack range and the player loses health	The player loses health when hit by the enemy's attack
The player's health bar goes down when they lose health	N/A	Allows the player to visually keep track of their health	The health bar will go down after getting hit by the enemy	The health bar goes down when the player is hit by the enemy

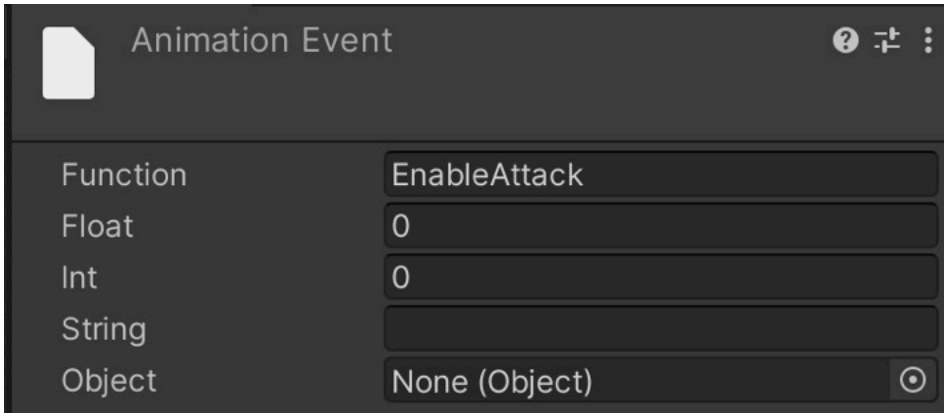
I now need to make sure the game knows when to disable and enable the enemy's attack collider. To do this I will need to go into the attack animation used for the zombie and add an event at the right time in the animation. I first need to enable the collider as soon as the enemy begins to swing their right arm and disable it just before the enemy swings their left arm.



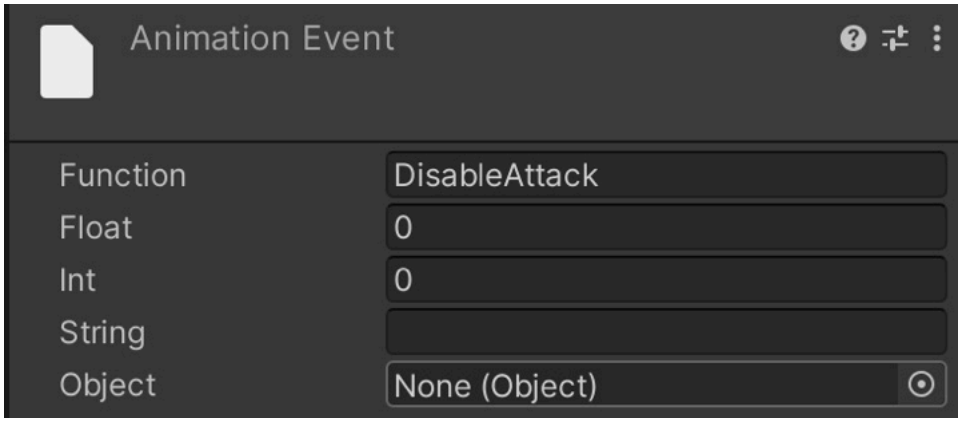
By slowly looking through the animation, it seems that the enemy begins swinging their right arm on frame 1 and starts swinging their left arm on frame 12. Therefore I will need to enable the box collider at frame 1 and disable it at frame 12.



To do this, I locate the desired part of the animation on the above picture and add an event.



Within the event I reference the exact name of the function I would want to get called at this time. In this case, the collider is enabled.



The same is done for when the collider needs to be turned off, this is done at the 12th frame.

Enemy Health:

I will create a simple script that gives the enemy health and also the logic for losing health when hit. Although this will only be fully implemented after I've finished developing the shooting system for the game. This script will be very similar to the script I had created for the player health, except there won't be any code related to the health bar.

```
7       private float maxHealth;
8       private float currentHealth;
```

I first created a new float variable called "maxHealth" which will hold the maximum health of the enemy. I then created another variable called "currentHealth" which will hold the current health of the enemy.

```
10      // Start is called before the first frame update
11      void Start()
12      {
13          // sets player's health to full
14          currentHealth = maxHealth;
15      }
16
```

Within the start function, I set the current health of the enemy to its max health. This is because the enemy will always spawn in with full health.

```
17      public void TakeDamage (float amount)
18      {
19          currentHealth -= amount;
```

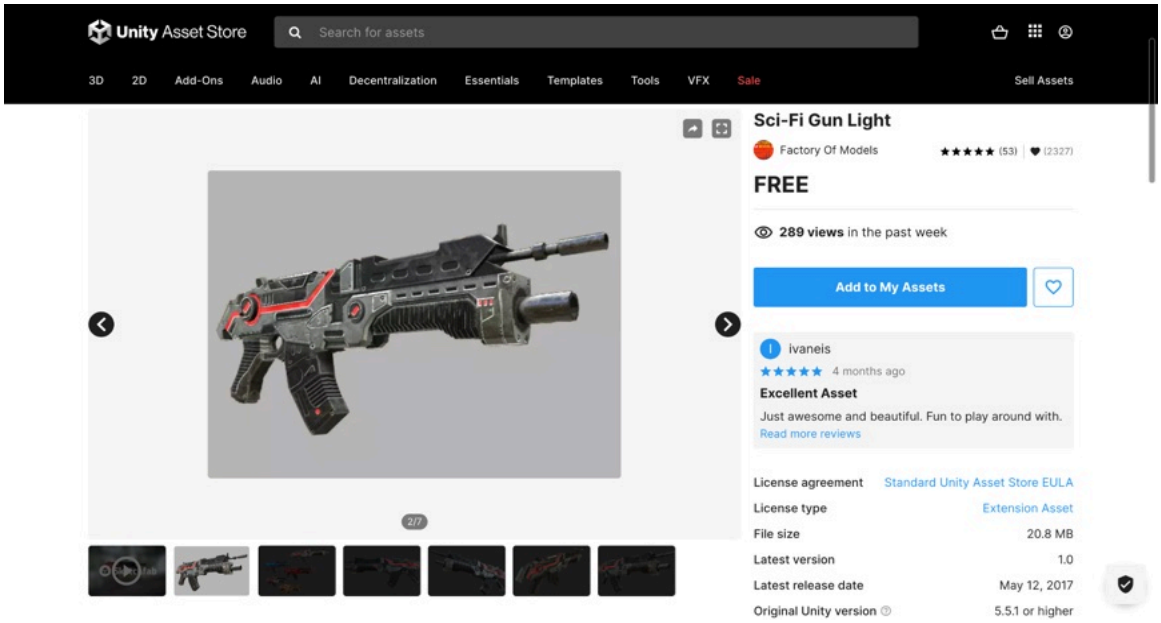
I then added a function called “TakeDamage” which takes a float “amount” and subtracts this from the enemy’s current health.

```
17      public void TakeDamage (float amount)
18      {
19          currentHealth -= amount;
20
21          if (currentHealth <= 0)
22          {
23              Destroy(gameObject);
24          }
25      }
```

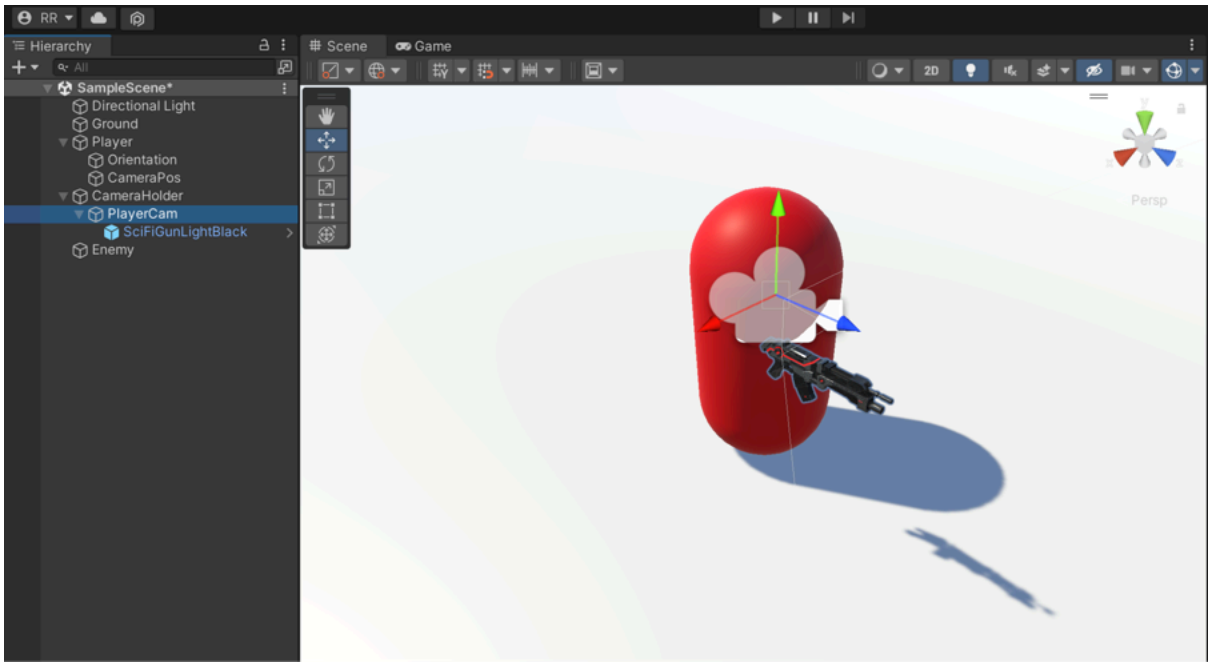
Within the “TakeDamage” function I also use an if statement to check whether the enemy’s health is less than or equal to 0. If this happens I will destroy the enemy game object, which will cause the enemy to disappear.

Gun:

I will now begin developing the shooting aspect of the game. This will also require some third party assets from the Unity Asset store. This will most likely just include the 3D model of the gun and possibly some visual effects as well. I again decided to give the choice for the gun model to my client, Samuel Mandoza. He ended up going for a sci-fi theme and chose this:



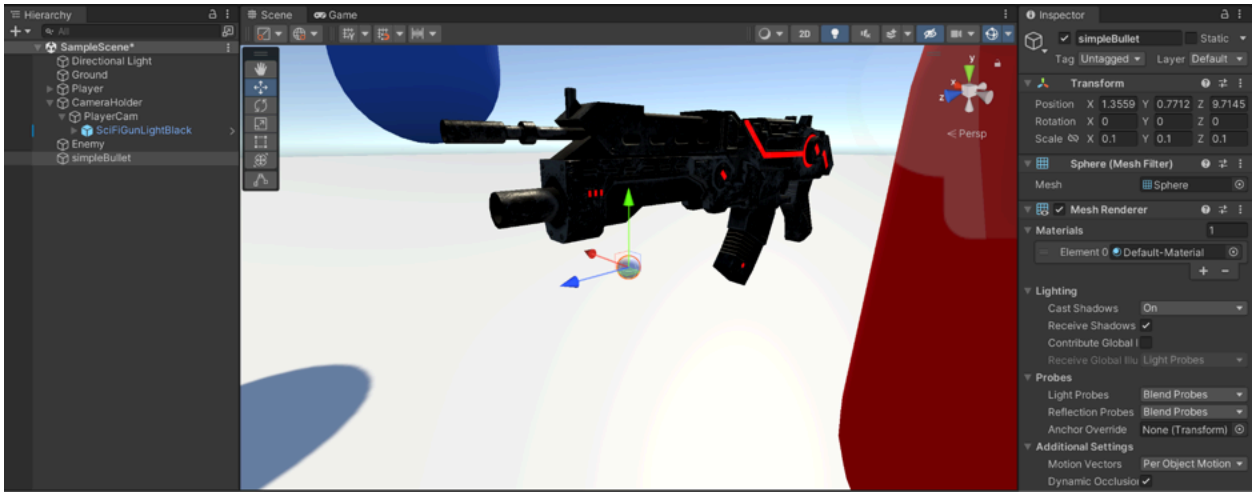
After downloading the asset, I imported it into my project through the Package manager. To set up the gun, I need the gun to move with the camera. Therefore, I put the gun prefab inside the “PlayerCam” and adjusted its position so that it looks natural.



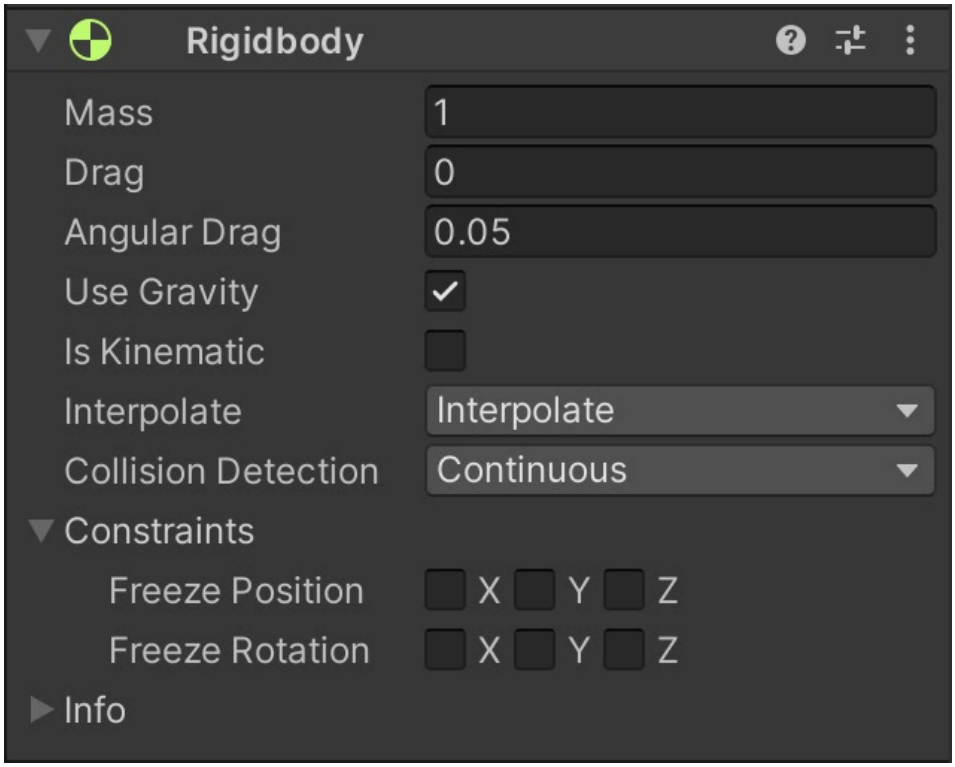
This is how the gun is set up within the camera. I have placed the gun at a suitable height and position so that it looks good in game. This is the current game view:

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class GunProjectile : MonoBehaviour
6   {   //bullet object
7       public gameObject bullet;
8
9       public int magazineSize,;
10      private int bulletsLeft;
11
12      private bool readyToShoot
```

I firstly initialised a public game object called ”bullet”, this will be the actual bullets within the game. I then created the “magazineSize” and “bulletsLeft” integers. “magazineSize” is the maximum number of bullets in the magazine of the gun, “bulletsLeft” holds the the number of bullets currently in the gun. The boolean “readyToShoot” will allow the player to shoot or not shoot depending on the situation.



For the bullet game object, I decided to just use a small sphere 3D object called “basicBullet”’. I made a temporary dimension of 0.1 for the X, Y and Z scaling. I then needed to add a rigidbody component to the bullet.



For the rigidbody of the bullet, I set the bullet to interpolate so that it moves smoothly. I also made the collision detection continuous as it is necessary for the bullet to detect when it is and isn’t hitting an enemy.

```
12      private void Awake;
13      {
14          // make sure magazine is full
15          bulletsLeft = magazineSize;
16          readyToShoot = true;
17
18      }
```

Within the Awake method, I have set the “bulletsLeft” to the magazine size of the gun and “readyToShoot” is also set to true. This is done before the game starts so the player should load into the game with a full magazine and also be able to shoot.

```
11      public bool allowButtonHold;
12
13      private bool readyToShoot, shooting
```

I have introduced 2 new boolean variables called “allowButtonHold” and “shooting”.

```
23      private void Update()
24      {
25          myInput();
26      }
27
28      private void myInput()
29      {
30          //check if allowed to hold down button to shoot
31          if(allowButtonHold) shooting = Input.GetKey(KeyCode.Mouse0);
32          else shooting = Input.GetKeyDown(KeyCode.Mouse0);
33
34      }
```

I have created a new function called “myInput” which is called within the update function. Creating a separate function to be called within the Update method keeps the code modular which promotes readability and can help when debugging code.

Within the myInput function, the if statement checks whether the mouse button is being held down, this will mean that “shooting” will be set to true. If the player is not allowed to hold down the mouse button, the same thing is done however ”GetKey()” changes to “GetKeyDown()” as the player will only be able to tap to shoot a bullet at a time.

```
5   public class GunProjectile : MonoBehaviour
6   {   //bullet object
7       public gameObject bullet;
8
9       public int magazineSize,;
10      private int bulletsLeft, bulletsShot;
11      public bool allowButtonHold;
12
13      private bool readyToShoot, shooting, reloading;
```

I introduced 2 new variables. The boolean “reloading” and the integer “bulletsShot”.

```
28      private void myInput()
29      {
30          //check if allowed to hold down button to shoot
31          if (allowButtonHold) shooting = Input.GetKey(KeyCode.Mouse0);
32          else shooting = Input.GetKeyDown(KeyCode.Mouse0);
33
34          //Shooting
35          if (readyToShoot && shooting && !reloading && bulletsLeft > 0)
36          {
37              //Set bullets shot to 0
38              bulletsShot = 0;
39
40              Shoot();
41          }
```

Before the player can shoot they must meet the conditions of the if statement. This means that “readyToShoot” must be true, the player must either be clicking or holding down the left mouse click, they should not be reloading at the same time and they must have more than 0 bullets in the magazine of the gun. The “bulletsShot" variable is used to keep track of the bullets shot, this is especially useful when the gun shoots multiple bullets at a time. Finally, after the player meets all the requirements, the shoot function is called.

```
46      private void Shoot()
47      {
48          readyToShoot = false;
49
50          //bullets in mag decrease by 1
51          bulletsLeft--;
52          //bullets shot increase by 1
53          bulletsShot++;
54      }
```

Within the “Shoot” function “readyToShoot” is set to false as the player is already shooting. The bullets left in the magazine are then decremented by 1 and the bullets shot are incremented by 1. I will now have to develop the logic for actually shooting the gun.

```
15      //reference camera
16      public camera PlayerCam
```

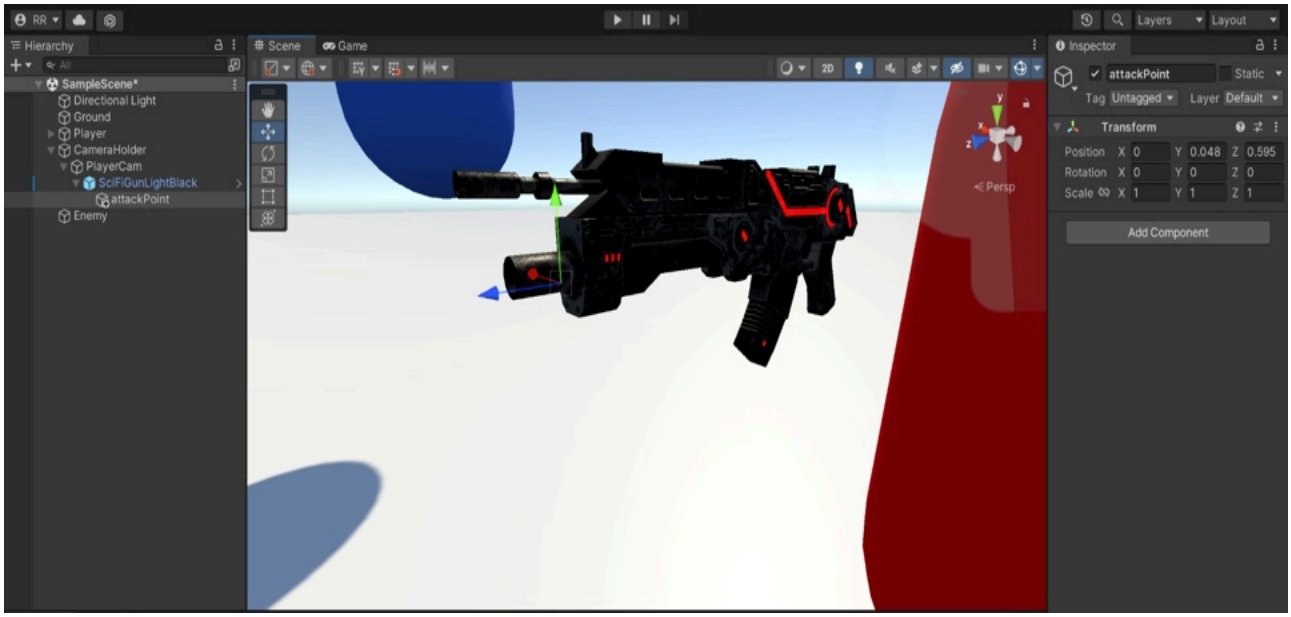
I first reference the player camera game object, “PlayerCam”.

```
50      //Find the exact hit position of bullet using a raycast
51      Ray ray = PlayerCam.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
52      Raycast hit;
```

In the first line a ray is casted from the center of the camera ”PlayerCam”. The number 0.5 for the x and y coordinate represent the middle of the camera. This allows the bullet to be sent out of the crosshair which will also be in the middle of the screen. If the ray hits any object, it is stored in the Raycast “hit”.

```
54          //Check if ray hits something
55          Vector3 targetPoint;
56          if (Physics.Raycast(ray, out hit))
57              targetPoint = hit.point;
58          else
59              targetPoint = ray.GetPoint(75);
```

The “Physics.Raycast” function checks if the ray has hit something. If so, “targetPoint” is set to the point in 3D space where the ray has hit. Otherwise, if the ray did not hit anything, “targetPoint” is set to a point 75 units in the direction of the ray. This means that “targetPoint” will still have a value even if it did not hit an object, this prevents any unexpected situations or possible crashes.




```
17      public Transform attackPoint
```

I then made another reference to the attackPoint game object. This is just an empty game object which is placed at the end of the gun as this is where the bullets will appear from.

```
65      //calculates direction from attackPoint to targetPoint
66      Vector3 direction = targetPoint - attackPoint.position;
```

Subtracting “attackPoint” from “targetPoint” results in a vector pointing from “attackPoint” to “targetPoint”. This vector is then stored in the variable “direction”.

```
68      //instantiate bullet
69      GameObject currentBullet = Instantiate(bullet, attackPoint.position, Quaternion.identity);
70      //Rotate bullet to shoot direction
71      currentBullet.transform.forward = direction.normalized;
72
```

This line of code instantiates a new bullet game object at the position “attackPoint”. The “Quaternion.identity” parameter means that the bullet is instantiated with no rotation. The spawned bullet is instantly stored in the game object “currentBullet", this allows me to keep track of the bullet.

The next line of code adds rotation to the bullet to ensure it is travelling in the correct direction along the desired trajectory which is held by the “direction” vector. This vector is normalised so that the magnitude is set to 1. This allows the vector to be used purely for direction and that the magnitude will have no effect on the speed of the bullet.

```
9       public float shootForce;
10
```

I created a new float called “shootForce”. This allows the speed of the bullet to be set.

```
73      //add forces to the bullet
74      currentBullet.GetComponent<Rigidbody>().AddForce(direction.normalized * shootForce, ForceMode.Impulse);
75
```

This line is used to add forces to the bullet. The rigidbody component of the bullet is retrieved and a force is applied to it. The direction of the bullet is multiplied by the shoot force, the vector “direction” is also normalised so that the magnitude relies solely on the “shootForce” variable. “ForceMode.Impulse” allows the force to be applied immediately which is necessary for something like a gun.

```
11      public float timeBetweenShooting;
```

```
26      public bool allowInvoke = true;
```

I introduced the “timeBetweenShooting” variable which will hold the time before the next bullet in the gun can be shot. I also created a new boolean variable called “allowInvoke” and set it to true.

```
84          //Invokes ResetShot function
85          if (allowInvoke)
86          {
87              Invoke("ResetShot", timeBetweenShooting);
88              allowInvoke = false;
89          }
90      }
91
92      private void ResetShot()
93      {
94
95      }
```

I created an empty function called “ResetShot” which is going to hold the logic of being able to shoot the next bullet. This function is called at the end of the shooting function through the “Invoke” function. The invoke function calls “ResetShot” after a specified amount of time which is held in the “timeBetweenShooting” variable. Then “allowInvoke” is also set to false.

```
92      private void ResetShot()
93      {
94          //allows shooting and invoke again
95          readyToShoot = true;
96          allowInvoke = true;
97      }
```

Within the “ResetShot” function, “readyToShoot” and “allowInvoke” are both set back to true. This allows the next bullet to be shot.

Test	Input	Justification	Expected Outcome	Actual Outcome
The player can shoot once	Left mouse click	Allows the player to shot the enemy	The player will shoot one bullet	The player shoots one bullet
The player can shoot continuously	Hold Left mouse click	Allows the player to shoot the enemy	The player will keep shooting until they run out of bullets	The player keeps shooting until they have no more bullets
The bullets travel in a straight line	Left mouse click	Allows the player to aim properly	The bullet will travel in a straight line	The bullet

Reloading:

```
12      public float timeBetweenShooting, reloadTime;
```

I added a new float called “reloadTime” which holds the time taken for the player to completely reload their gun.

```
99      private void Reload()
100     {
101         reloading = true;
102         Invoke("ReloadFinished", reloadTime);
103     }
104
105     private void ReloadFinished()
106     {
107
108     }
```

The relation between the “Reload” and “ReloadFinished” function is similar to that of the “Shoot” and “ResetShot” function. When the player reloads, “reloading” is set to true and the “ReloadFinished” function is invoked alongside the time taken to reload held in the “reloadTime” variable.

```
105     private void ReloadFinished()
106     {
107         bulletsLeft = magazineSize;
108         reloading = false;
109     }
```

Within the “ReloadFinished” function, the “bulletsLeft” variable is to the “magazineSize”. This means that the bullets in the gun magazine are set back to its maximum capacity. “reloading” is then set to false.

```
41      //Reloading
42      if (Input.GetKeyDown(KeyCode.R) && bulletsLeft < magazineSize && !reloading) Reload();
43
```

I then added this if statement in the “MyInput” function. The “Reload” function is only called if the player presses the “R” key, is not already reloading and the magazine isn’t full.

Test	Input	Justification	Expected Outcome	Actual Outcome
The player can reload their gun	Press “R” key	Allows the player to play kill more	The gun will reload and have all the bullets in	The gun reloads with all the bullets in the

		enemies	the magazine	magazine
The player cannot shoot while reloading	Left mouse click when reloading	This makes the game more realistic and fair	The gun will not shoot while reloading	The gun does not shoot while reloading

Bullet Collisions

Currently the bullet bounces off the enemy and stays on the ground of the map as an instantiated object. I need to check whether the bullet collides with the enemy or not, and then make the bullet disappear. The enemy's health must also deplete when hit with the bullet.

To do this I created a new script called "BulletCollisions" and attached it to the bullet prefab in my game.

```
5   public class BulletCollisions : MonoBehaviour
6   {
7       public float bulletDamage = 20f;
8
```

I first created a new float variable named "bulletDamage" and set its value to 20, this will be used as a parameter when calling the "TakeDamage" function.

```
9   private void OnCollisionEnter(Collision collision)
10  {   //checks if bullet collides with enemy
11      if (collision.gameObject.CompareTag("Enemy"))
12      {
13          //gets the enemy health component
14          EnemyHealth enemyHealth = collision.gameObject.GetComponent<EnemyHealth>();
15      }
16
```

Within the "OnCollisionEnter" function, I first checked if the bullet had collided with the enemy. If it did, I get the "EnemyHealth" component attached to the enemy game object and set it to "enemyHealth".

```
16          if (enemyHealth != null)
17          {
18              enemyHealth.TakeDamage(bulletDamage);
19          }
20          //destroys the bullet
21          Destroy(gameObject)
```

This if statement will help with the robustness of the code. Since the main “Enemy” game object contains the “EnemyHealth” component, every instantiated clone of the “Enemy” game object will also contain the “EnemyHealth” component. Therefore “enemyHealth” will always be set to the “EnemyHealth” component.

However, in some cases, perhaps an error during instantiation, the game object may not spawn in with the “EnemyHealth” component, this means that the “enemyHealth” variable will be set to null. The if statement makes sure that the enemy game object hit with the bullet contains the “EnemyHealth” component before carrying out the “TakeDamage” function.

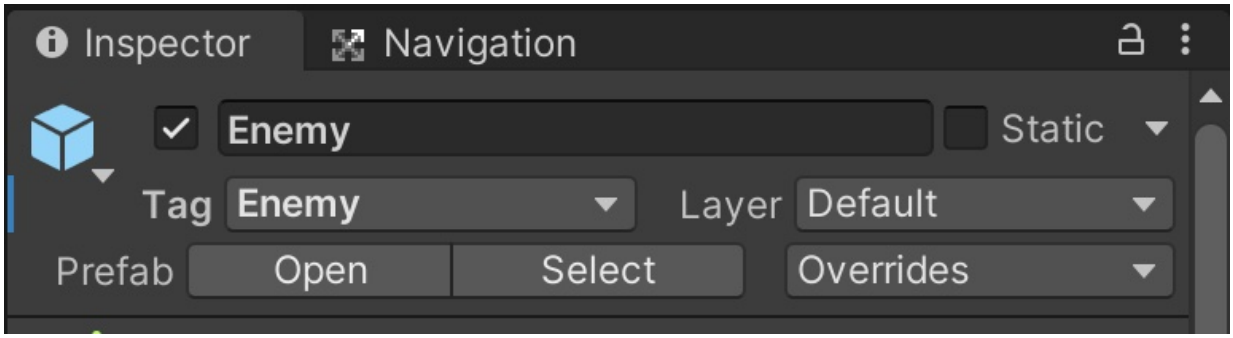
The bullet is then destroyed after damaging the enemy.

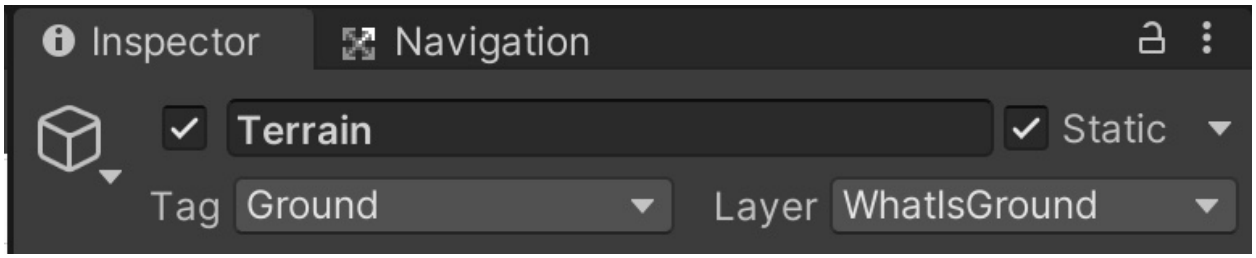
```
24          //checks if bullet collides with ground
25          else if(collision.gameObject.CompareTag("Ground"))
26          {
27              Destroy(gameObject);
28          }
29      }
```

If the bullet didn’t collide with the enemy , another check is done to see if it had collided with the ground. If it did, the bullet would also be destroyed.

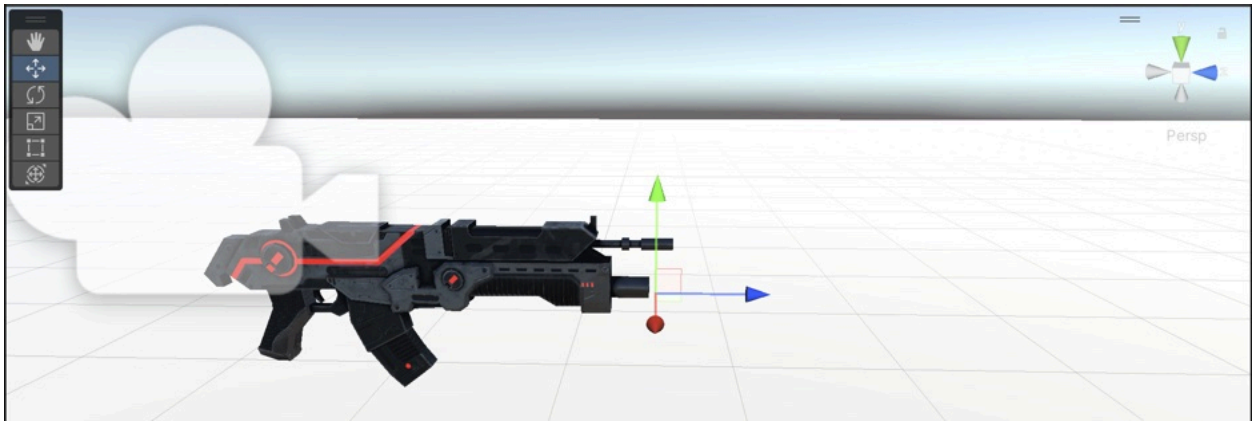
Test	Input	Justification	Expected Outcome	Actual Outcome
The bullet is destroyed when it collides with the ground	N/A	Improves resource management and optimisation	The bullet will be destroyed when it collides with the ground	The bullet is destroyed as soon as it is instantiated
The bullet is destroyed when it collides with the enemy	N/A	Improves resource management and optimisation	The bullet will be destroyed when it collides with the enemy	The bullet is destroyed as soon as it is instantiated

This was surprising as the script for the bullet collision was quite straightforward. Somehow my bullet was colliding with objects that contained either the “ground” or “enemy” tag as soon as the game started. At first I checked the tags set for all the game objects within my game and made sure only the enemy and terrain had their respective tags. This could prevent any possible unknown collisions causing the bullet to get destroyed.





However this didn't solve the issue, I then considered the way the bullet was spawned in. I had a game object called "attackPoint" in the barrel of the gun. There was a possibility that when the bullet spawned, it may have collided and fell to the ground before I could shoot.



Therefore I moved the attack point to right in front of the barrel instead of inside it, however this also had no effect as the bullet kept disappearing.

I then decided to track the position of the bullet game object when the game starts. I realised that since the bullet has gravity, it would drop and since it was right above the terrain it would just get destroyed. I figured, turning off gravity for the bullet would quickly solve the issue, I also felt that the bullet was better without gravity as otherwise the player would need to take into account the bullet drop making the game much more complicated.

However this caused another issue causing the bullets to fly off the map if they were shot too high upwards which prevented the bullets from being destroyed. To prevent this I thought I could

destroy the bullet if they did not collide with anything for a relatively long period of time, maybe around 15 seconds.

When an enemy is hit with a bullet, he gets pushed out the way, therefore I changed the rigidbody of the enemy from static to dynamic

Test	Input	Justification	Expected Outcome	Actual Outcome
The bullet is destroyed when it collides with the ground	N/A	Improves resource management and optimisation	The bullet will be destroyed when it collides with the ground	The bullet is destroyed as soon as it is instantiated
The bullet is destroyed when it collides with the enemy	N/A	Improves resource management and optimisation	The bullet will be destroyed when it collides with the enemy	The bullet is destroyed as soon as it is instantiated

Finally the whole shooting mechanism for the game should be functioning properly.

Developer-Client Review:

These are the requirements from the success criteria that were looked at:

No.	Success Criteria	Justification	☑	Reference
4.	Display the health bar of the players and enemies	This allows the player to keep track of how much health he has until he dies. The enemies health bar tells the player how much more damage is required to kill it.	☐	COD Zombies
6.	Player can move in all 4 directions as well as jump upwards	This allows the player to move away from enemies and to traverse the map. This feature is absolutely necessary for the game.	☐	COD Zombies
8.	The player can shoot their gun at the enemy	This allows the player to defend themselves from the enemies and is a key element for the player's survival	☐	COD Zombies

11.	The player starts with 100 health	This causes the player to be wary of getting attacked by an enemy as once they lose all their health, they will lose a life.	□	COD Zombies
15.	Attacking animation for the enemy	This will make the game more realistic and immersive which leads to a better gaming experience.	□	Interview
17.	Chasing animation for the enemy	This will make the game more realistic and immersive which leads to a better gaming experience.	□	Interview
20.	The enemy can attack and kill the player	This is necessary for a survival game as the player must avoid taking too much damage from the enemy as they would die otherwise	□	The Forest

Me: I have finished the development of the first prototype for this game. The core aspects of the game have been worked on and developed, the game is now in an almost playable state. The player movement has been tested and is completely functional. The enemy system has also been tested, with enemies being able to spawn, attack, run and die. The ability for the player to shoot has been developed. The shooting and reloading aspect of the gun has also been tested and seems to be fully functional as well.

Samuel Mandoza: Amazing! I love the progres you have made on the game so quickly. After looking at the prototype and testing it out, it looks like the game is heading in a good direction. However I have some suggestions and possible improvements for the game. The game currently feels quite empty, I think the next step is to begin creating a suitable map for the player, possibly a space themed map similar to what I mentioned in the interview previously. I also have a few smaller features to add to some of the game's developments. These include:

- The player can jump continuously by holding the space bar
- The player can sprint by holding the "Shift" key
- The gun automatically reloads when the player tries shooting with no bullets in the magazine.
- Fixing an issue where the zombie tilts backwards when player is too close

New requirements for development after client review.

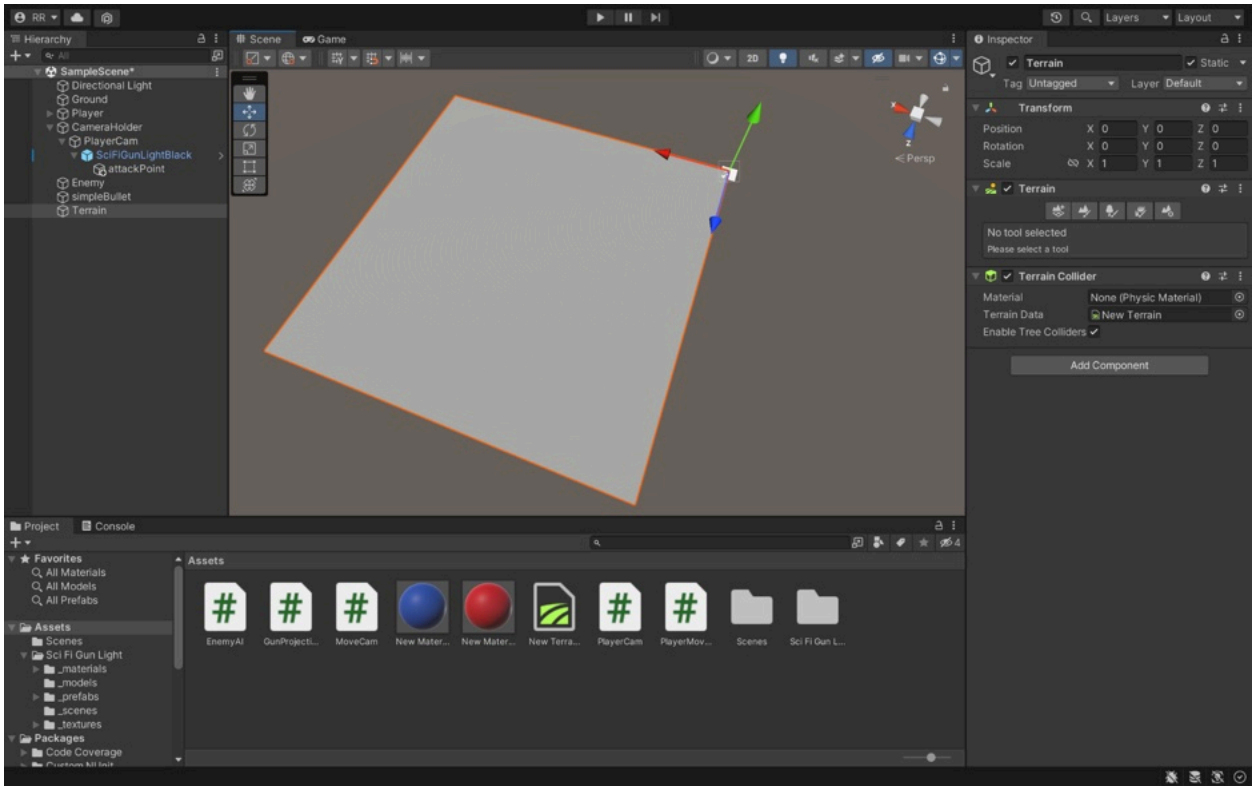
Requirement	Justification		Reference
The player can jump continuously	This makes jumping easier as the player does not need to repetitively		Client Review

	press the spacebar		
The player can sprint to move faster	This allows the player to move faster in dangerous situations to avoid dying		Client Review
The gun automatically reloads if the player tries shooting with no bullets in the magazine	This is just a quality of life improvement so that the player doesn't have to press the "R" key every time.		Client Review
The zombie stays upright when close to the player	This allows the enemy to function normally		Client Review

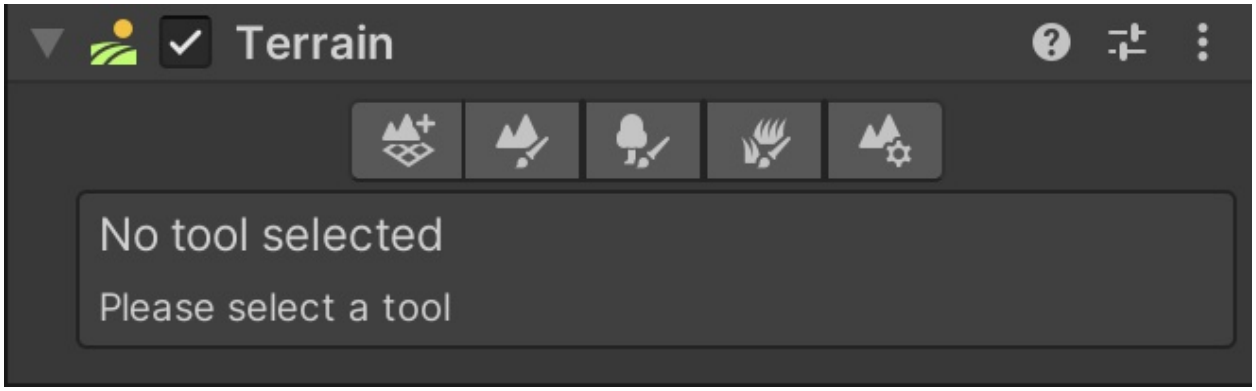
Map for the game:

I will start by creating a map since this was the next step for the game according to my client, Samuel. Since he had wanted a space theme to the map, I was thinking of making something similar to the moon. Possibly a rocky surface for the ground and stars in the background and the sky.

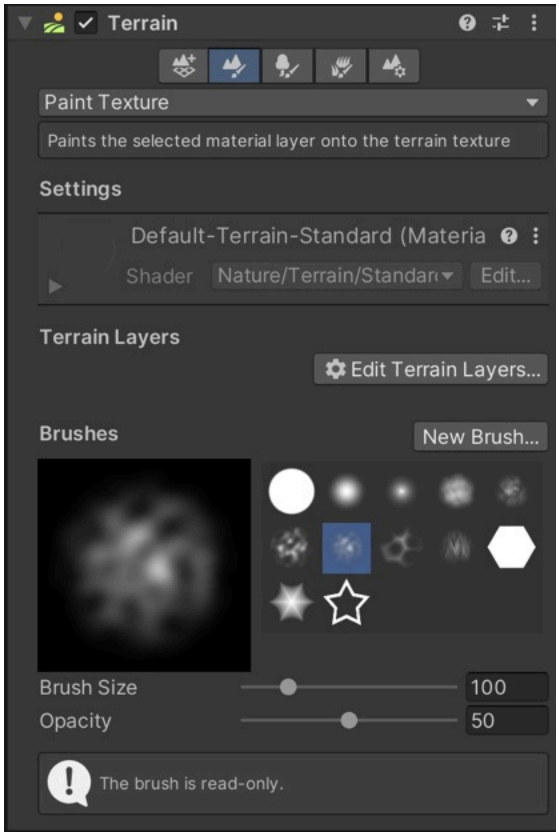
I first started by adding a terrain object to the project as this will be the most suitable way to create the map. The terrain will be relatively easy to design and edit due to Unity's inbuilt terrain editor. It will also be the most efficient way to create a map in Unity.

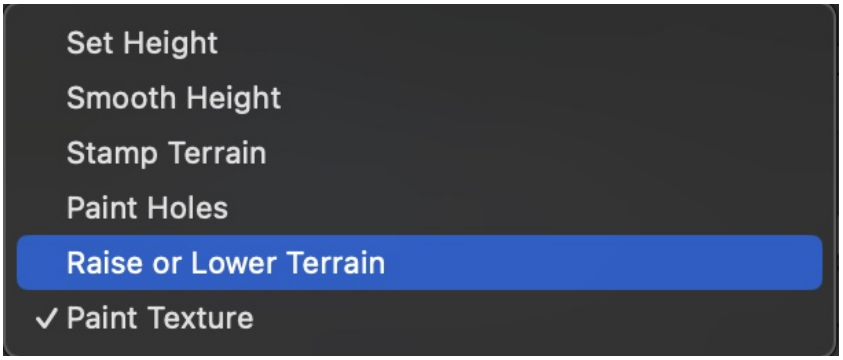


As you can see the terrain is quite big however this can be adjusted later on if necessary. To edit the terrain I will need to use the sculpting tool in the inspector for the terrain.

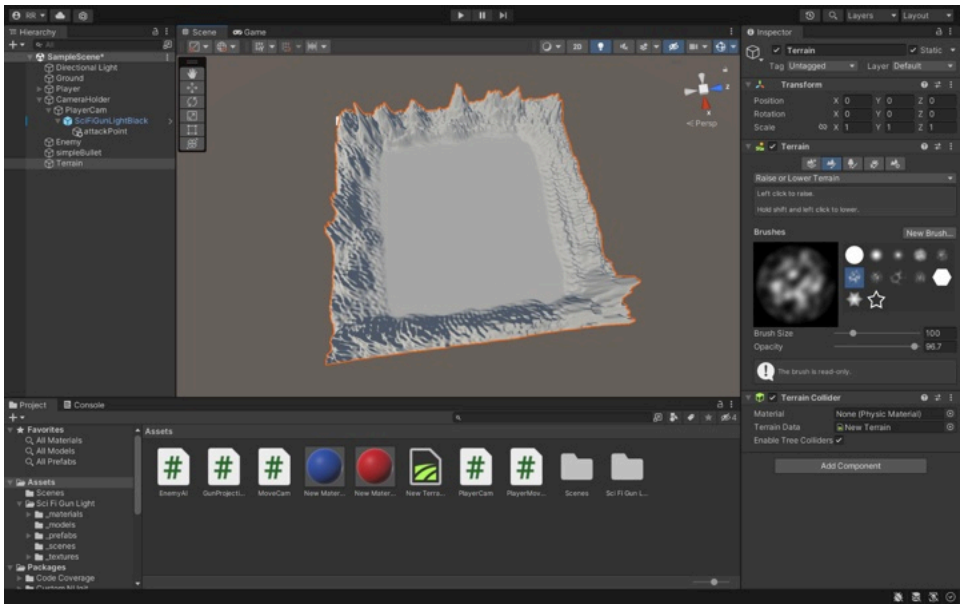


There are multiple tools available for designing the terrain. The first tool is used to add neighbouring terrains, the second tool is used to paint the terrain, the third tool is used to paint trees for the terrain, the fourth tool is used to paint any details and the final tool contains the setting for the terrain. The tool I will be using the most is probably the one for painting the terrain as this is the main tool for changing the look of the terrain.



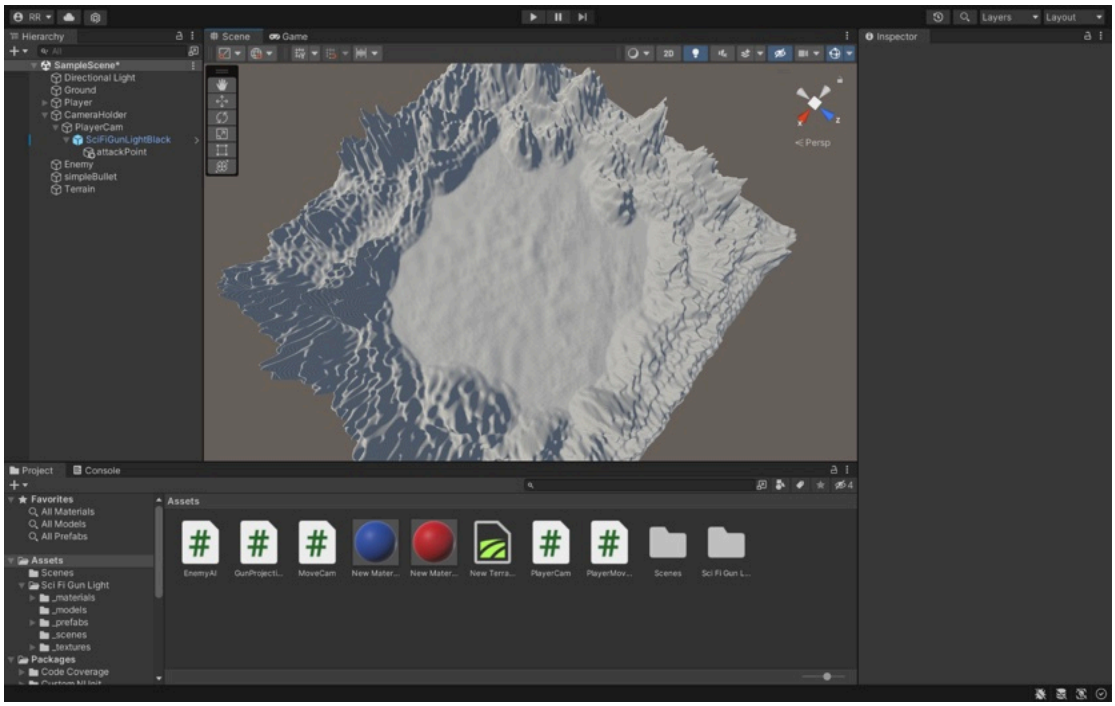


When selected we can see there are many options for creating the terrain. My first initial thought is to create a rocky/bumpy terrain with large steep rocks surrounding the edge of the map. This creates a closed space in the terrain for the player to move around and play.

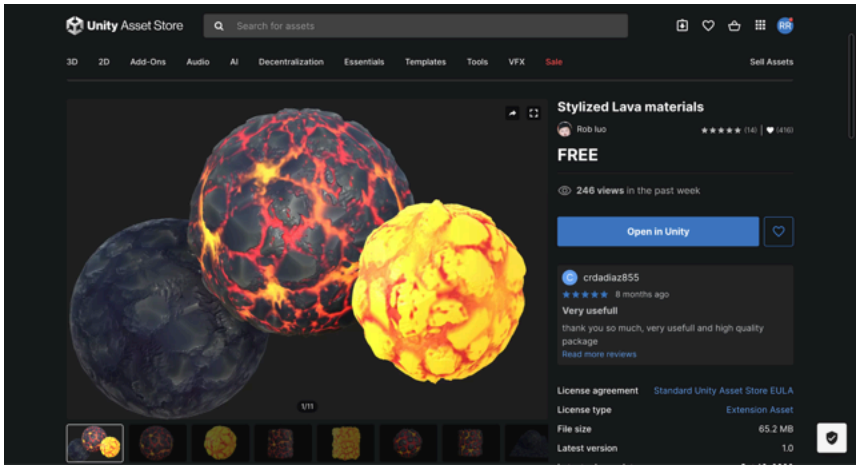


To begin creating the terrain I used a brush with quite a large brush size and opacity. The brush size determines how large the area of terrain the brush is affecting, the opacity determines the

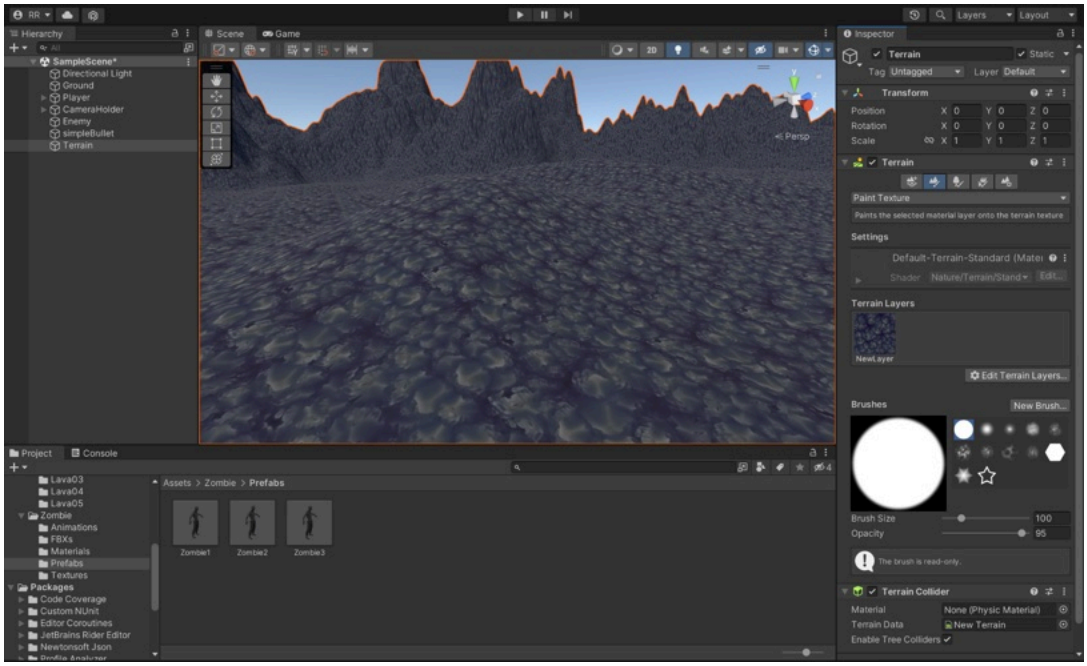
strength of the brush. I specifically used a spiky brush, this allowed me to get the large pointy mountains that surround the map. However the map looks quite artificial as there are no irregularities. Furthermore I think that the middle area of the map shouldn't be completely flat, especially if we're going for the look of a planet in space.



Therefore I made the edges between the mountains and the middle ground jagged so it looks more natural. I also made the ground uneven and bumpy to make the map more interesting and realistic. Now I need to add some textures to the terrain, this will require some more third party assets from the Unity asset store.



I decided to use this asset since it has a good texture for the ground. Furthermore, I can use the lava textures to add some more detail in the map, for example I can make the point mountains seem like volcanos.



To do this I downloaded the asset then imported into my project. I then went back into the Paint Terrain tool and selected the paint texture option. I then created a new layer with the chosen material which immediately applied to the whole terrain.

However I encountered a major issue with the change in terrain I had made. Therefore I carried out a test to see what had been affected.

Test	Input	Justification	Expected Outcome	Actual Outcome
The player can move in all directions freely	W,A,S, D keys	Allows the player to move around the map	The player will move freely on the ground in all directions	The player glitches through the ground when walking into a slight hill
The player can jump	Space bar	Allows the player to jump	The player will be able to jump up and back down	The player does not jump

Both the directional movement and the jumping ability of the player had been affected. I quickly realised the issue with the player movement was in the way I decided to program it. I used the “Transform” class to translate the position of the player on the X and Z axis, however this did not take into account the y coordinates of the player. I found that a solution to this problem was to remake the code for the movement to use the “Rigidbody” class instead of the “Transform” class. This will allow me to add a force on the player depending on the direction which will allow

them to travel up hills. However, due to time constraints I have, I decided to just make the ground flat.

The flat ground also means that the NavMesh system for the enemy will remain practically the same and I would have to spend less time changing it to suit the new map. Furthermore, finding walk points for the enemy would be much more complicated as I would have to take into account the y position of the new calculated point on the map.

Another issue I encountered was that the player wasn't able to jump. I was initially very confused as when I pressed the spacebar, the player didn't even attempt to jump, whereas the player could still move slightly on the bumpy ground. However, after recognising that the player still couldn't jump after making the ground flat, I realised that I simply forgot to add the "ground" layer to the terrain. This was necessary as the player can only jump when they are making contact with the ground.

Test	Input	Justification	Expected Outcome	Actual Outcome
The player can move in all directions freely	W,A,S, D keys	Allows the player to move around the map	The player will move freely on the ground in all directions	The player moves freely on the ground
The player can jump	Space bar	Allows the player to jump	The player will be able to jump up and back down	The player jumps

Client review features:

I will now need to try and implement the new features which my client had requested for. First starting with the features related to the movement system for the game.

Continuously Jumping:

To allow the player to continuously jump by holding down the space bar, I figured a simple solution would be to change how the input for the space bar is taken.

```
45          //Lets the player jump
46          if(Input.GetKey(KeyCode.Space) && isOnGround)
47          {
48              playerRb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
49              isOnGround = false;
50          }
```

Instead of using the “GetKeyDown()” function, I will use the “GetKey()” function within the update method. This will continuously check whether the space bar is held down or not, allowing the player to continuously jump.

Test	Input	Justification	Expected Outcome	Actual Outcome
The player can jump	Space bar	Allows the player to jump	The player will be able to jump up and back down	The player jumps
The player can continuously jump	Hold space bar	Prevents the player for having to continuously press the spacebar	The player will jump continuously on the ground	The player jumps continuously on the ground

Player Sprinting:

To add the feature of player sprinting to the game should also be quite simple. I just need to increase the player’s speed when they press the shift key.

```
7     public float speed = 5.0f; // Determines the speed of the player
8     public float sprintSpeed = 10.0f; //Determines the speed of the player when sprinting
```

I first introduced a new variable called "sprintSpeed” and set it to a faster speed than the regular player’s speed

```
42
43        float currentSpeed = speed;
44
45        if (Input.GetKey(KeyCode.LeftShift))
46        {
47            currentSpeed = sprintSpeed;
48        }
```

I then created a new variable called “currentSpeed” which will hold the speed of the player depending on whether they are sprinting or not. “currentSpeed” is initially set to the regular speed but is changed to the sprint speed if the player holds down the shift key.

```
50        // Move the player
51        transform.Translate(moveDirection * currentSpeed * Time.deltaTime);
52
```

I then altered the code that moves the player to use the “currentSpeed” instead of just “speed”.

Test	Input	Justification	Expected Outcome	Actual Outcome
The player can sprint	Hold shift key + “W, A, S, D” keys	Allows the player to move faster in critical situations	The player will move faster in every direction	The player moves faster in every direction
The player can jump while sprinting	Hold shift key + space bar + “W, A, S, D” keys	Allows the player move in the same way as when they are not sprinting	The player will be able to jump while moving faster in every direction	The player jumps while moving faster in every direction

Reloading Feature:

To be able to automatically reload the gun when the player shoots with no bullets, I will just have to call the “Reload” function when the player attempts to shoot.

```
53        //Automatically reload when shooting with no ammo
54        if (readyToShoot && shooting && !reloading && bulletsLeft = 0) Reload();
```

This simple if statement allows the “Reload” function to be called if all the criteria is met. The player must not already be shooting, they should be clicking / holding down the left mouse click, they should not be already reloading and should have no bullets left in the magazine.
However I am faced with an unexpected error:

```
[20:08:42] Assets/GunProjectile.cs(54,13): error CS0019: Operator '&&' cannot be applied to operands of type 'bool' and 'int'
```

This was very confusing as it makes complete sense for the “&&” operator to be used in this situation. Furthermore, I have multiple other lines of code with the same layout:

```
51        //Reloading
52        if (Input.GetKeyDown(KeyCode.R) && bulletsLeft < magazineSize && !reloading) Reload();
```

```
56        //Shooting
57        if (readyToShoot && shooting && !reloading && bulletsLeft > 0)
```

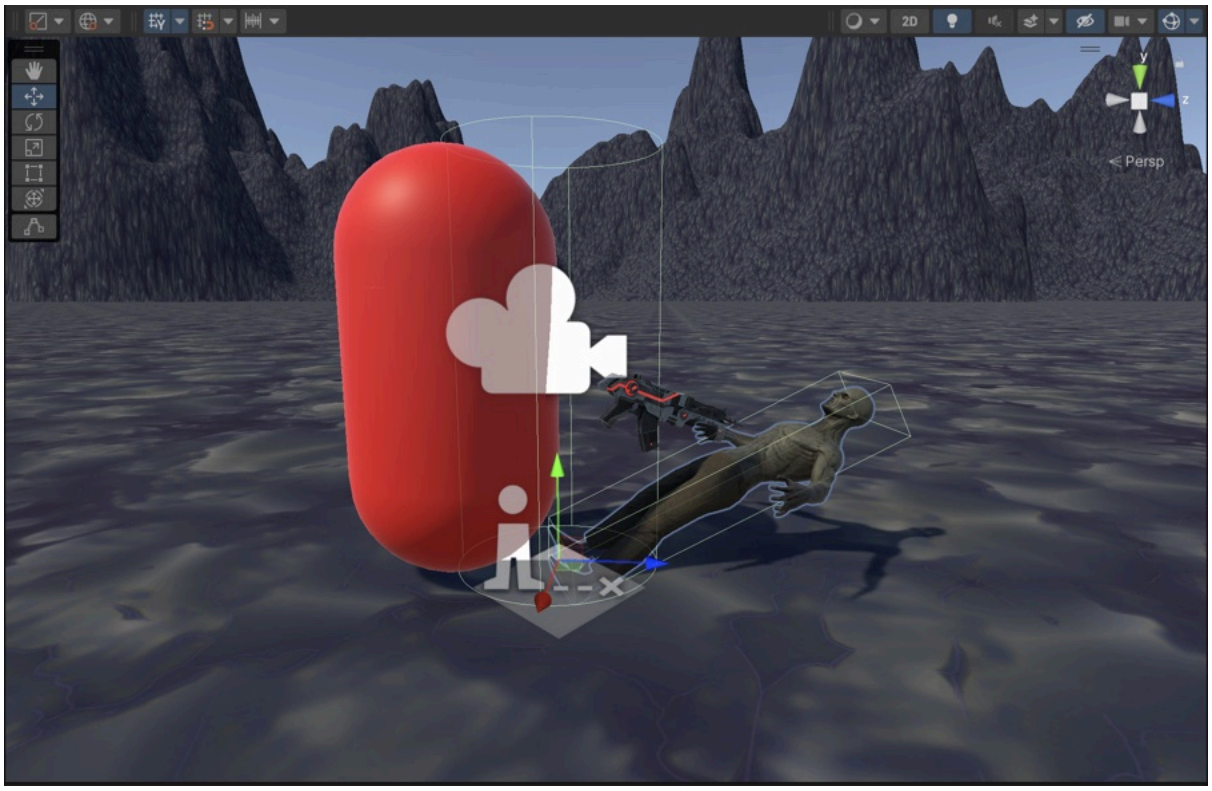
I then realised that the issue wasn’t the “&&” operator but the fact I was using a boolean and integer, however this was also the same in previous cases which worked as expected. I then realised that I used a “=” instead of a “==” when checking if the bullets left are equal to 0.

```
53        //Automatically reload when shooting with no ammo
54        if (readyToShoot && shooting && !reloading && bulletsLeft == 0) Reload();
```

Test	Input	Justification	Expected Outcome	Actual Outcome
The player can sprint	Hold shift key + “W, A, S, D” keys	Allows the player to move faster in critical situations	The player will move faster in every direction	The player moves faster in every direction

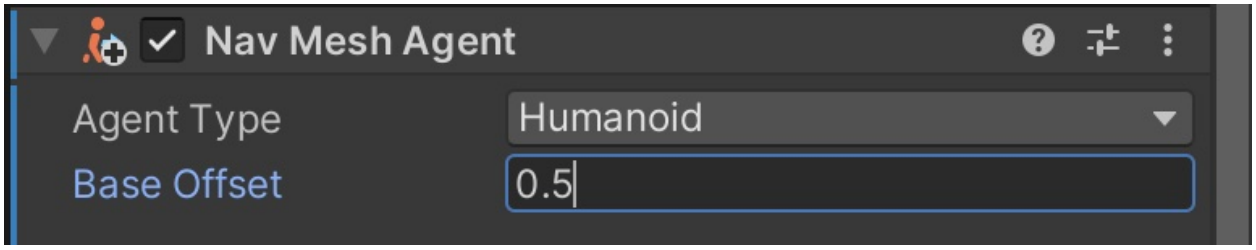
Error Fixing:

This issue had occurred when the player walks right next to the enemy, the enemy would tilt backwards to the ground as seen in the picture below:

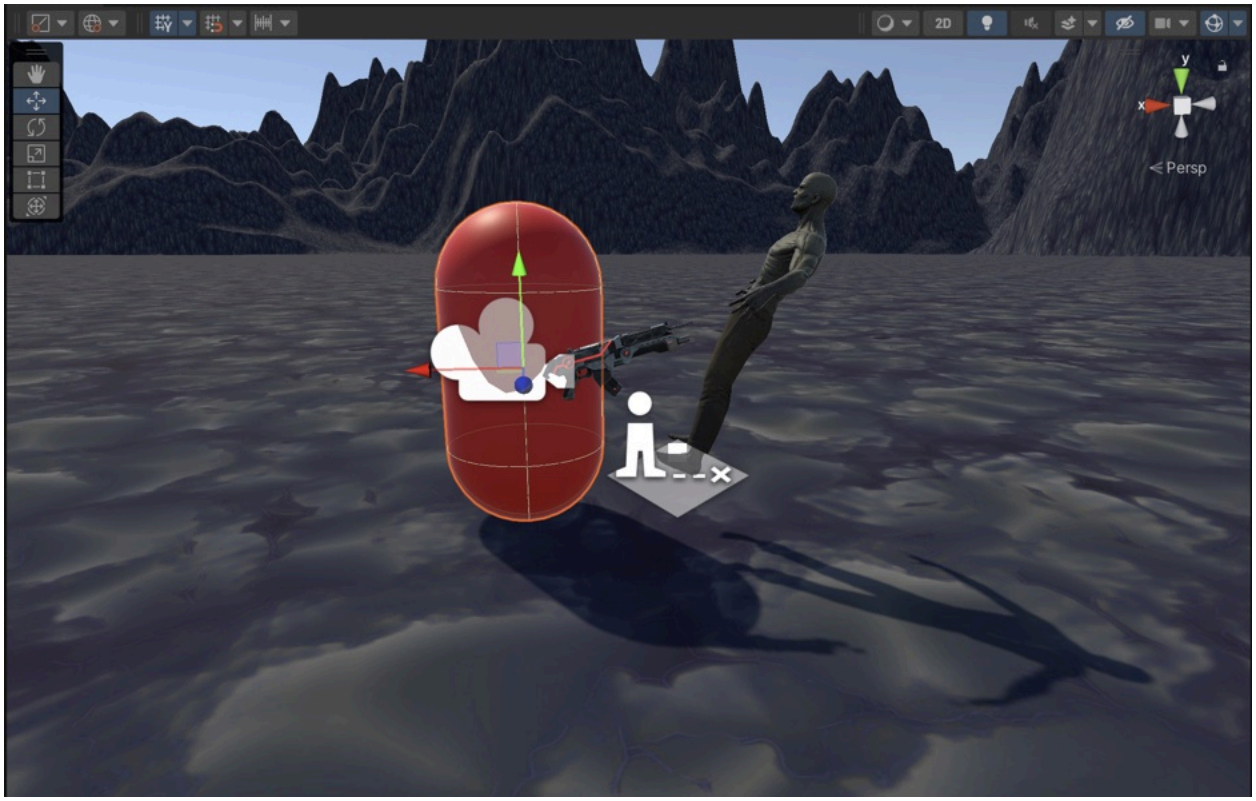


This definitely should not be happening, the enemy is meant to be upright regardless of the distance from the player. My first thought was to simply constraint the X rotation within the Rigidbody component of the enemy as this should technically prevent the enemy from rotating backwards.

However this had no effect, which made me think that this must be an issue with the NavMesh system, possibly forcing the enemy to tilt backwards. I first started by tampering with the base offset of the “Nav Mesh Agent” as this property controls the height of the enemy's center point.

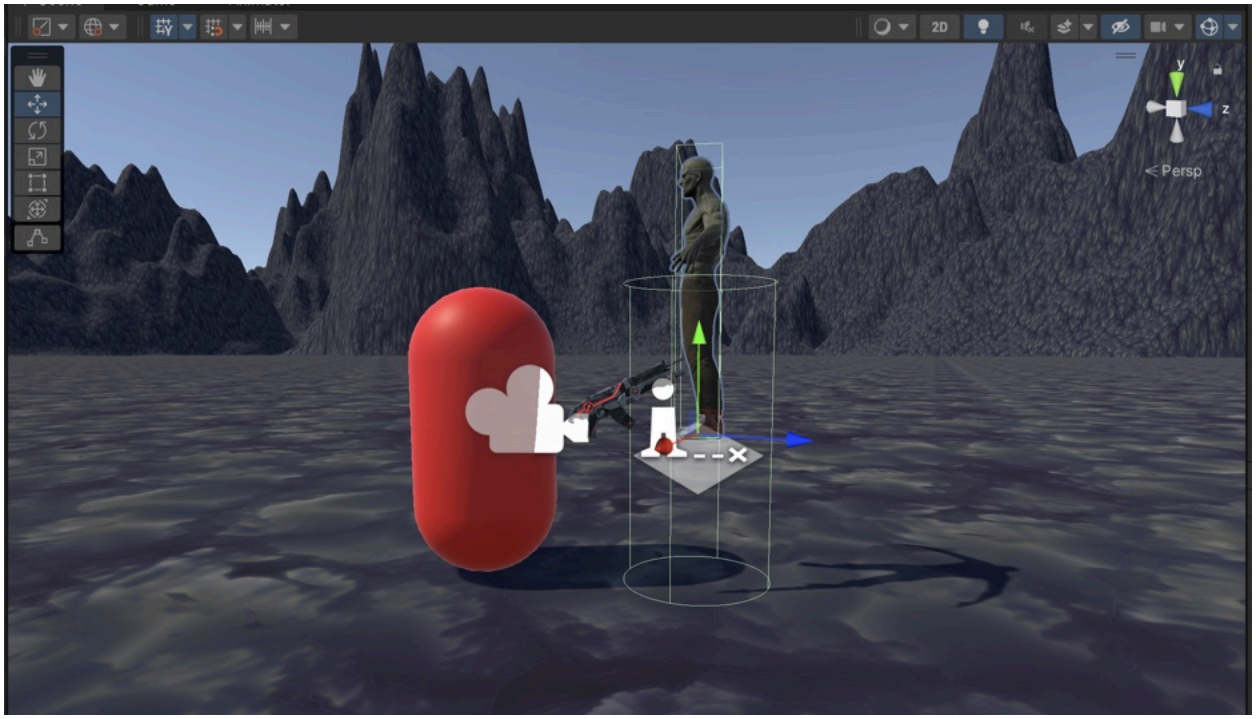


I first started by increasing the base offset to 0.5 from 0. This had no effect but only adjusted the height of the zombie.



I kept on experimenting with the base offset until I found that when the base offset was set to 1, the enemy had stopped tilting backwards when I was close to it. However this wasn’t a suitable solution as the enemy was still fairly high above the ground. The enemy also instantly tilted

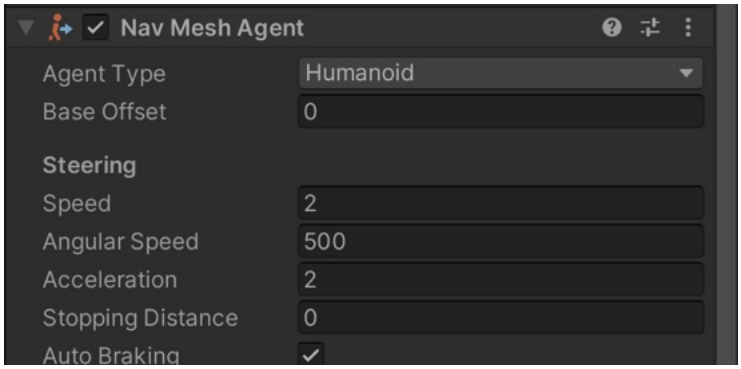
backwards again as soon as I jumped.



After some more experimentation, I realised that the enemy only tilted back when I was within the attack range. I realised this by setting the attack range further away and seeing that the enemy still tilted back even though I wasn't close. So this meant it must've been an issue with the script for the enemy, therefore I revisited the attack function.

```
88        transform.LookAt(player);
```

This particular line stood out to me, as the zombie could possibly be tilting to face the player. I decided to see how the enemy would perform without this line, and it seemed to function perfectly fine. The zombie also continued to face the direction of the player while chasing and attacking. I believe this was because the zombie was scripted to follow the player so it would face the player either way. The only issue was that the zombie was turning very slowly. A simple fix to this was to just increase the angular speed of the enemy in the Nav Mesh Agent component.



An angular speed of 500 seemed to be suitable as the enemy took about 1-2 seconds to complete a whole 180 degree turn.

I also realised that the line of code that I removed would prevent the enemy from looking directly at the player on slopes. This means that any slopes on the map would mean that the enemy wouldn't be able to function realistically.

Test	Input	Justification	Expected Outcome	Actual Outcome
The enemy is upright when the player moves into the attack range	"W, A, S, D" keys towards the enemy	Allows the player to move freely and realistically	The enemy will stay upright	The enemy stays upright
The enemy always faces the player when they are within range	N/A	Makes the enemy seem more realistic	The enemy will keep looking at the player in both attack and chase stats	The enemy looks at the player while chasing but then stops when attacking the player.

Even though the zombie seemed to perform fine without the "LookAt" function, after a few attacks the zombie begins to randomly turn. This means that the "LookAt" function was important for the enemy to work properly. However it will cause issues so I am not sure of what to do. I first thought that the only issue was that the enemy was looking upwards. I could possibly only make the enemy look at the player horizontally, preventing the enemy from tilting backwards.

```
84      private void AttackPlayer()
85      {
86
87          if (!enemyAnimator.GetCurrentAnimatorStateInfo(0).IsName("Z_Attack"))
88          {
89              //triggers the attack animation
90              enemyAnimator.SetTrigger("Attack");
91              //make sure enemy doesn't move
92              enemy.SetDestination(transform.position);
93              //make sure that the enemy looks at the player
94              transform.LookAt(player);
95          }
```

But then I realised that the "LookAt" function was previously in the "ChasePlayer" function as well, this wasn't necessary as I realised the enemy looks at the player while chasing anyways, due to "SetDestination". I then tried adding the "LookAt" function to the "AttackPlayer" function to see if this would work properly with the Nav Mesh.

Test	Input	Justification	Expected Outcome	Actual Outcome
The enemy always faces the player when they are within range	N/A	Makes the enemy seem more realistic	The enemy will keep looking at the player in both attack and chase stats	The enemy looks at the player while chasing and attacking

Developer-Client Review 2:

These are the requirements from the success criteria that were looked at:

No.	Success Criteria	Justification	☑	Reference
5.	A terrain for the game	This allows the player and enemies to move around properly. It also adds a lot to the game compared to an empty plane being used for the ground	□	COD zombies
4.	Display the health bar of the players and enemies	This allows the player to keep track of how much health he has until he dies. The enemies health bar tells the player how much more damage is required to kill it.	□	COD Zombies
7.	Player can sprint to move faster	This allows the player to move faster in dangerous situations to avoid dying	□	The Forest

After completing all the previously updated requirements, I decided to have another meeting with my client to provide an update on the status of the project.

Me: I have completed the base design for the map. I have also addressed the majority of the suggestions and improvements which you had provided. The only exception was the muzzle flash on the gun as I believed this wasn't a necessity and would save me a lot of time if I were to avoid developing it. I am hoping to develop some of the remaining core features of the game by the next prototype so that the development project is closing in on becoming a full proper game.

Samuel Mandoza: That is great to hear! I am glad you listened to the majority of my suggestions and they seem to be working great! I agree with your plan of trying to finish the game by either this or the next iteration. There are only a few more parts of the game which need to be added, such as the UI, a game save feature and possibly multiplayer/ co-op. Some more specific things I suggest are:

- The spawning of the enemies in waves and groups
- The player death system which include lives and respawning as well as the game over screen
- A score and round counter
- An ammo counter at the bottom of the screen
- A crosshair in the middle of the screen

Requirements	Justification		Reference
The spawning of the enemies in waves and groups	This makes the game more interesting.		Client Review
The player death system which include lives and respawning as well as the game over screen	Allows the player to be afraid of the zombie enemies as they can attack and kill the player		Client Review
A score and round counter	Allows the player to visually keep track of their progress in the game		Client Review
An ammo counter at the bottom of the screen	Allows the player to visually keep track of their bullets		Client Review
A crosshair in the middle of the screen	Allows the plate to aim where the bullets travel to		Client Review

I decided to wait on designing and developing the UI as I feel this would be suitable towards the end of the development for the game. I think I'll begin working on the enemy spawn system as I just finished developing parts of the enemy and would like to just continue on.

Enemy spawn system:

I also decided to make amends to my original plan for how I would spawn the enemies. Instead of spawning the zombies in small groups around the map, I settled to just spawn the zombies randomly across the map.

This seemed to be a much quicker and simpler solution to spawning enemies on the map. Furthermore, the way I have implemented the navigation system for the zombies will cause

them to move around randomly while in the patrol state, this would mean spawning enemies in groups may be redundant as they will just end up spreading out across the map either way.

```
7       public GameObject Enemy;
8       //max no. of enemies
9       public int enemyCountLimit;
```

I first referenced the “Enemy” game object which I will be spawning. I also created a variable called “enemyCountLimit” which holds the maximum number that will spawn on the map.

```
24      IEnumerator EnemySpawner()
25      {
26          //current no. of enemies
27          int enemyCount = 0;
```

I created an enumerator called “EnemySpawner”, this is because an enumerator will allow the code to be run over multiple frames. And also another variable called “enemyCount” which holds the current number of zombies that have been spawned on the map.

```
28
29          while (enemyCount < enemyCountLimit)
30          {
31              //generates random x and z value
32              float xPosition = Random.Range(xPositionMin,xPositionMax);
33              float zPosition = Random.Range(zPositionMin,zPositionMax);
```

Within the enumerator “EnemySpawner”, I have a while loop in which a random Z and X position is continuously generated. This while loop only happens when the number of enemies on the map is less than the maximum number of enemies which will spawn.

```
34
35              //spawns in enemy
36              Instantiate (Enemy, new Vector3(xPosition, 3, zPosition), Quaternion.identity);
37
```

The “instantiate” function is used to spawn in a new instance of the “Enemy” game object with the vector coordinate of the random X and Z position that was previously generated. The Y position is always set to the value 3 as this is the level of the ground. “Quaternion.Identity” is used so that the instantiated object is spawned with no rotation.

```
38          //time betwween each enemy spawn
39          yield return new WaitForSeconds(0.1f);
40          enemyCount += 1;
41      }
```

Line 39 is a special construct which is usually used to introduce a delay in the execution of the coroutine. This means that the zombies don't spawn all at once, they have a 0.1 second time gap between each of them. The enemy count is also increased after each new zombie is instantiated.

```
19      void Start()
20      {
21          StartCoroutine(EnemySpawner());
22      }
```

Within the start function, the coroutine “EnemySpawner” is started by the “StartCoroutine” function.

Test	Input	Justification	Expected Outcome	Actual Outcome
The enemies spawn on the map	N/A	Allows the game to function as intended	The enemies will spawn randomly across the map	The enemies spawn randomly across the map

I now need to be able to form waves of enemies, this will be done when the player kills all the enemies on the map. To do this I will need to repeat the coroutine with an increased “enemyCountLimit.”

```
 9      //max no. of enemies
10      public int initialEnemyLimit;
11      public int currentEnemyLimit;
12      //enemeies added after each wave
13      public int additionalEnemies;
```

I first added 2 new variables, “initialEnemyLimit” and “additionalEnemies” and changed the name of the “enemyCountLimit” variable to “currentEnemyLimit” as this name will make more sense.

```
23      // Start is called before the first frame update
24      void Start()
25      {
26          currentEnemyLimit = initialEnemyLimit;
27          StartCoroutine(EnemySpawner());
28      }
```

Within the start function I added a new line that set the ”initialEnemyLimit” to the “enemyCountLimit”. This is because the “currentEnemyLimit” will change throughout the algorithm as the coroutine gets called iteratively.

```
30      IEnumerator EnemySpawner()
31      {
32          while (true)
33          {
34              //current no. of enemies
35              int enemyCount = 0;
36
37              while (enemyCount < currentEnemyLimit)
38              {
39                  //generates random x and z value
40                  float xPosition = Random.Range(xPositionMin,xPositionMax);
41                  float zPosition = Random.Range(zPositionMin,zPositionMax);
42
43                  //spawns in enemy
44                  Instantiate (Enemy, new Vector3(xPosition, 3, zPosition), Quaternion.identity);
45
46                  //time betwween each enemy spawn
47                  yield return new WaitForSeconds(0.05f);
48                  enemyCount += 1;
49              }
```

I also placed the previous contents of the coroutine within a while loop that repeats infinitely. This means when the coroutine is called, it will keep spawning waves of enemies after each complete loop of the “while(true)” condition. However I need to prevent the waves of enemies being continuously spawned one after the other as this would spawn too many enemies and possibly cause the game to crash.

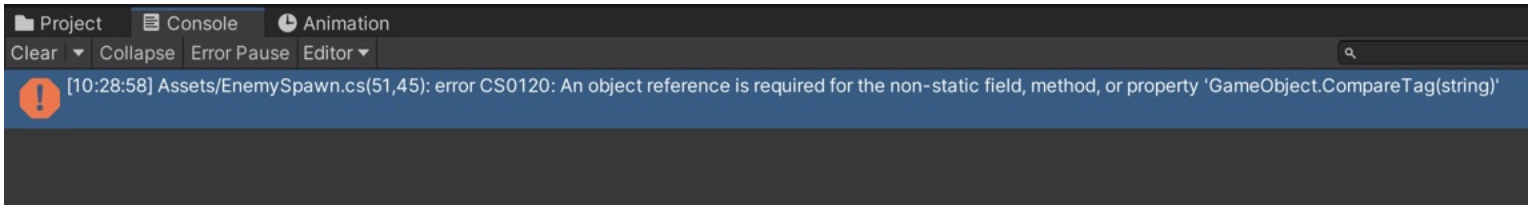
```
//wait for all enemies to be killed before restarting
yield return new WaitUntil(()=> GameObject.CompareTag("Enemy").Length == 0);
```

This line within the while loop allows me to control when the wave of enemies are spawned. Similar to when the enemies are slightly delayed from spawning all at once, this line delays the execution of the coroutine from spawning a wave of enemies until the condition is met. This condition means that the coroutine will only continue once all the enemies have been killed.

```
53          //increments the enemy limit
54          currentEnemyLimit += additionalEnemies
```

Finally the “currentEnemyLimit” is incremented before the next spawning of enemies.

Test	Input	Justification	Expected Outcome	Actual Outcome
The enemies spawn on the map	N/A	Allows the game to function as intended	The enemies will spawn randomly across the map	The game does not run
The enemies spawn in controlled waves	N/A	Allows the player to keep killing zombies without getting too overwhelmed	The enemies spawn again after previous wave is killed	The game does not run



I am faced with this error on line 51. This is the line that delays the execution of the coroutine if the specified condition is met. I realised that the “.CompareTag” function cannot be used since I am not specifically checking a single game object if it has the “Enemy” tag, hence the error requesting for an object reference. I am rather looking through the whole game scene to see if there are any game objects with the tag “Enemy” left.

Therefore I have to use a different function called “FindGameObjectWithTag()” as this dynamically queries the whole Unity scene to find any game objects with the tag ”Enemy”.

Test	Input	Justification	Expected Outcome	Actual Outcome
The enemies spawn on the map	N/A	Allows the game to function as intended	The enemies will spawn randomly across the map	The game does not run
The enemies spawn in controlled waves	N/A	Allows the player to keep killing zombies without getting too overwhelmed	The enemies spawn again after previous wave is killed	The game does not run

```
Assets/EnemySpawn.cs(51,87): error CS1061: 'GameObject' does not contain a definition for 'Length' and no accessible extension method 'Length' accepting a first argument of type 'GameObject' could be found (are you missing a using directive or an assembly reference?)
```

For some reason the length check I did when finding the number of game objects with the "Enemy" tag is not working . Although the function "FindGameObjectWithTag()" returns a list of all the game objects, therefore ".Length()" should work perfectly fine.

After some external research, I realised there is a difference between "FindGameObjectWithTag()" and "FindGameObjectsWithTag()" . The first function (the one I used in my code) only finds a single game object from the scene and hence there is no list allowing ".Length()" to be used. The second function returns a list of all the game objects which means it should work properly.

Test	Input	Justification	Expected Outcome	Actual Outcome
The enemies spawn on the map	N/A	Allows the game to function as intended	The enemies will spawn randomly across the map	The enemies spawn randomly across the map
The enemies spawn in controlled waves	N/A	Allows the player to keep killing zombies without getting too overwhelmed	The enemies spawn again after previous wave is killed	No enemies are spawned.

I encountered a problematic issue when the player eliminates all the enemies on the map:

```
[01:27:46] MissingReferenceException: The object of type 'GameObject' has been destroyed but you are still trying to access it.
Your script should either check if it is null or you should not destroy the object.
```

It says that I am trying to access a game object that has already been destroyed. This error pops up as soon as I eliminate the last enemy on the map. Therefore I instantly thought that the issue must occur when I am trying to instantiate the enemy game object. I realised that this error makes sense as the enemy object I reference when instantiating is also walking around on the map. Therefore, killing this enemy prevents me from spawning any more.

I decided that the solution was to disable the enemy game object after instantiating the enemy clones and then reactivate it before the enemies are instantiated again for the next round.

```
37          while (enemyCount < currentEnemyLimit)
38          {
39              //generates random x and z value
40              float xPosition = Random.Range(xPositionMin,xPositionMax);
41              float zPosition = Random.Range(zPositionMin,zPositionMax);
42
43              //activates enemy game object
44              Enemy.SetActive(true);
45
46              //spawns in enemy
47              Instantiate (Enemy, new Vector3(xPosition, 3, zPosition), Quaternion.identity);
48
49              //time betwween each enemy spawn
50              yield return new WaitForSeconds(0.05f);
51              enemyCount += 1;
52              //deactivates enemy game object
53              Enemy.SetActive(false);
54
55          }
```

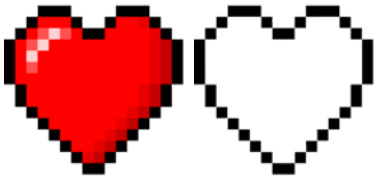
To do this, I used the simple ".SetActive()" method with the parameter of either true or false to set the "Enemy" game object to active or inactive. I first set the "Enemy" game object to true, allowing me to instantiate the new enemy clones. I then set it to false after all the enemies were spawned in.

Test	Input	Justification	Expected Outcome	Actual Outcome
The enemies spawn in controlled waves	N/A	Allows the player to keep killing zombies without getting too overwhelmed	The enemies will spawn again after previous wave is killed	The enemies spawn again after previous wave is killed

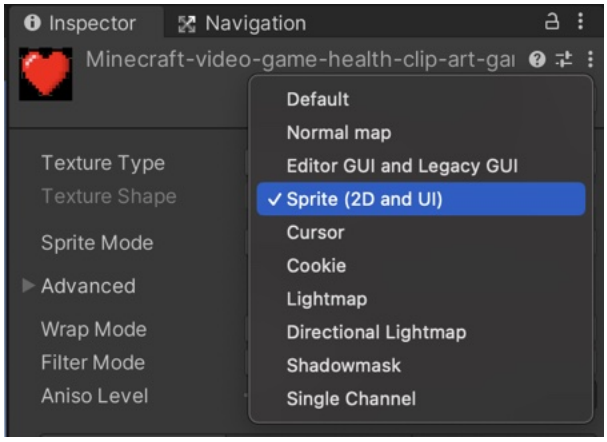
Player death system:

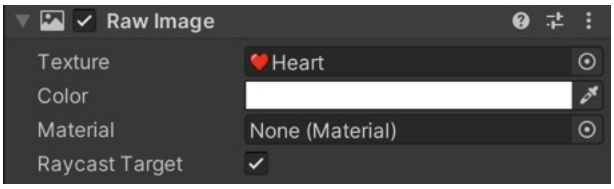
To develop the player death system, I will first have to introduce the concept of lives into the game. This means after the player loses all their health, they respawn while losing a life in the process. The player also needs a visual representation of how many lives they have left, this will be done using pixel hearts that disappear when a life is lost.

I will first begin the development by importing a picture of a heart that can be used as a sprite on the canvas for the game. I did this by going on the internet and searching for a simple pixel heart. I then took the same heart picture and edited it to remove the colour so I have an empty heart image as well.

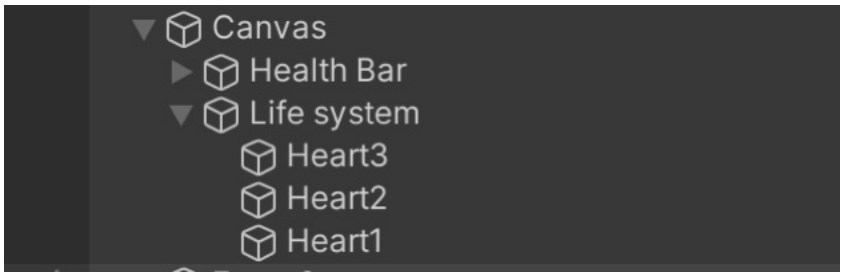


I imported the image as a new asset from the files in my computer. I first created a game object called “Life system” within the canvas and then created 3 new image child objects.

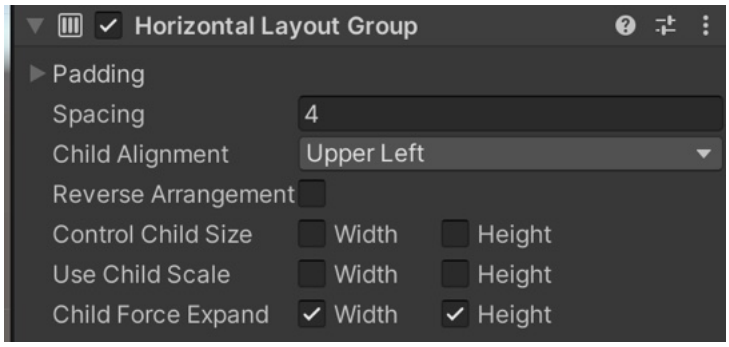




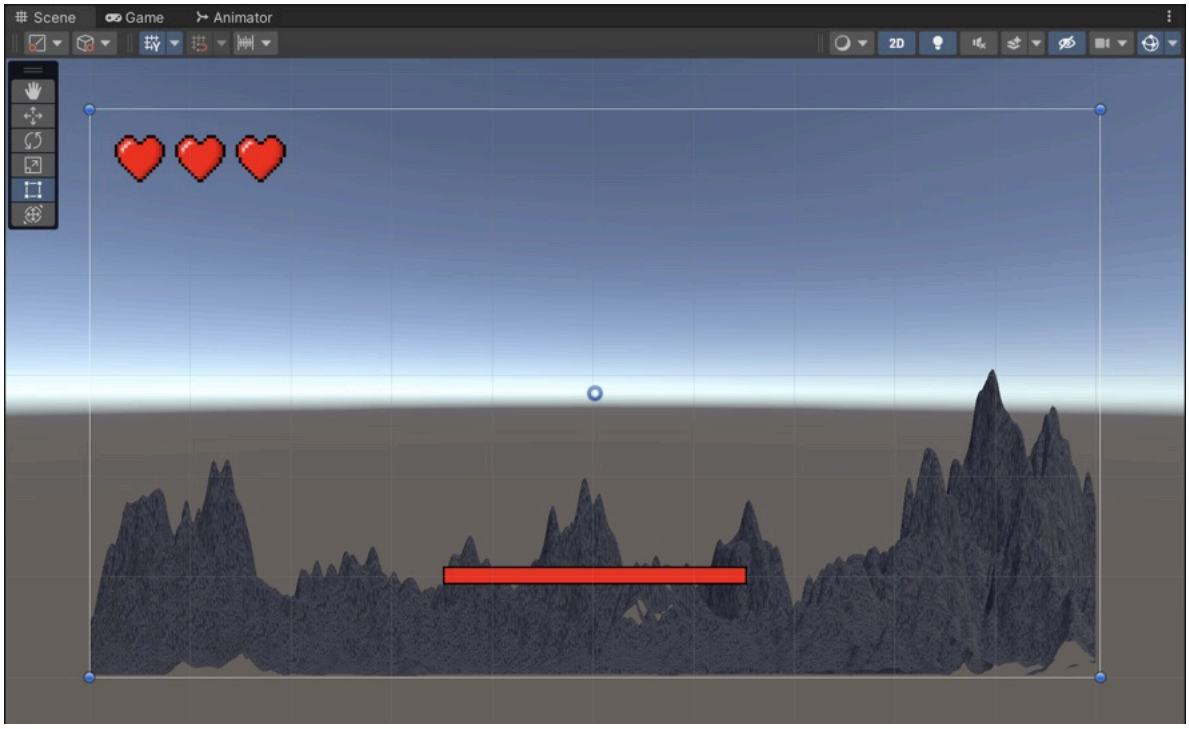
I then made sure that the texture for the image I used was set to a sprite for 2D and UI. I then referenced the image to the image child object for the canvas.



I then added a new component called “Horizontal Layout Group” to the “Life System” game object which allows me to easily control how the hearts will look on the canvas.



This allowed me to set even spacing between each heart as well as the position of the group of hearts on the screen



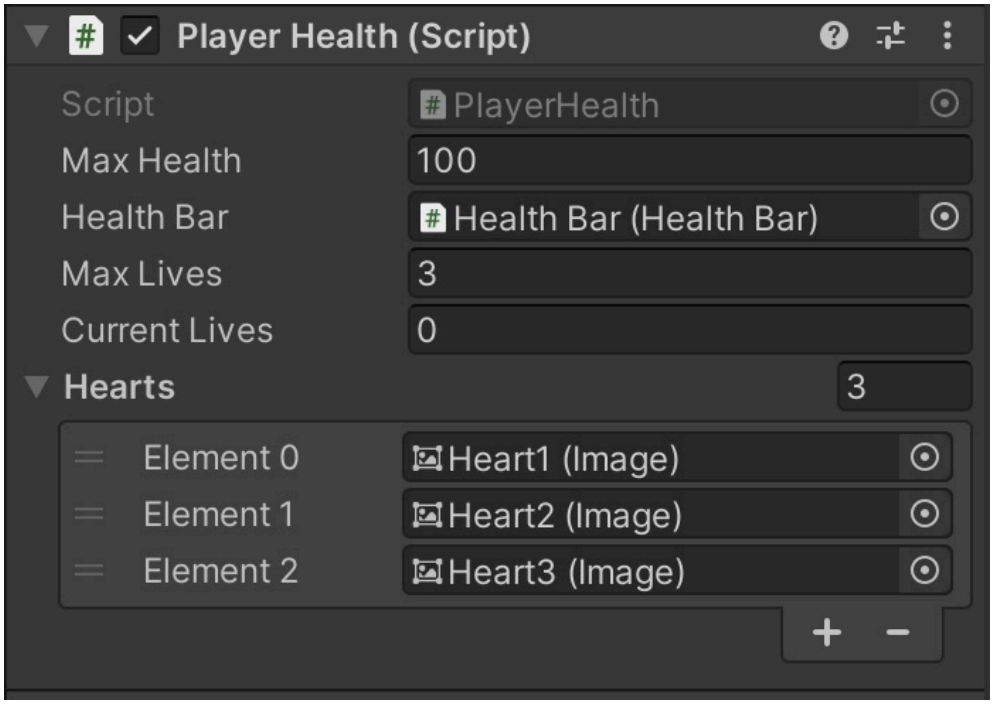
This is currently how the canvas for the game looks like. The hearts are ordered 1 to 3 from left to right. I now need to code the logic for the hearts disappearing as well as the life system for the player. I decided to code the logic in the “PlayerHealt” script as this keeps things ordered.

```
4    using UnityEngine.UI;
```

I first introduced the “UnityEngine.UI” namespace in the script as this is necessary when dealing with images on the canvas.

```
13    //Life variables
14    public int maxLives;
15    public int currentLives;
16    public Image[] hearts;
```

I then created 2 new variables called “maxLives”, which holds the maximum lives the player can have, and “currentLive”‘, which holds the number of lives the player currently has in the game. I then also created a new array called “hearts” that holds an array of images.



Within the “PlayerHealth” component for the player, I set the max lives to 3 and create 3 new heart elements. I then referenced the heart images for each element.

```
18      //reference to heart sprites
19      public Sprite emptyHeart;
20      public Sprite filledHeart;
```

I then also referenced both the empty and filled heart sprites within the script, this is necessary if I want to change the sprite being displayed.

```
18      // Start is called before the first frame update
19      void Start()
20      {
21          // sets player's health to full
22          currentHealth = maxHealth;
23          //sets slider to full
24          healthBar.SetSliderMax(maxHealth);
25          //sets player's lives to full
26          currentLives = maxLives;
27      }
```

Within the start method I also set the “currentLives” to the “maxLives” as the player always starts the game with full health.

```
34          //sets all heart sprites to filled
35          foreach (Image heart in hearts)
36          {
37              heart.sprite = filledHeart;
38          }
```

Also in the start method, I used a foreach loop which iterates through each heart image in the “hearts” array and sets the sprite to the “filledHeart” sprite. This is because the player will always start the game with 3 hearts.

```
45          if (currentHealth <= 0)
46          {
47              Die();
48          }
```

Within the “TakeDamage” function for the player, I introduced an if statement that checks when the player has no more health left. If true, the “Die” function is called which will hold the logic for what happens when the player dies.

```
52      void Die()
53      {   //removes a life
54          currentLives --;
55
56          if(currentLives <= 0)
57          {
58              Debug.Log("GameOver");
59          }
```

Within the “Die” function, I first remove a life from the “currentLives” variable. I then check if the player has any lives left, if they do, “GameOver” is printed to the console. Printing to the console is a temporary action I am taking. Generally a screen that says “Game Over” should pop up, which would clearly indicate to the player that the game is over. However, I decided to do this while developing the UI for the game as it would make the most sense.

```
61          else
62          {
63              //resets player health
64              currentHealth = maxHealth;
65              //update heart sprite after losing a life
66              hearts[currentLives].sprite = emptyHeart;
67
```

If the player does have lives left, The player's health is restored to full. The "hearts' array is also accessed with the index of the "currentLives". This works because arrays are zero-indexed, therefore when "currentLives" is 2, the second element of the array is the third heart, so this ends up working perfectly. After the respective heart element is taken from the array, it's sprite is changed to an "emptyHeart" sprite.

Test	Input	Justification	Expected Outcome	Actual Outcome
The player starts the game with 3 lives	N/A	This gives the player more than one chance	3 hearts will be displayed on the top left of the player's screen	3 hearts are displayed on the top left of the player's screen
The player loses a life when they die	N/A	The player is punished for dying and loses one of their lives	The rightmost heart sprite will change to an empty heart	The rightmost heart sprite changes to an empty heart
The player's health is fully restored after they die	N/A	Allows the game to function properly	The health bar will return to full health	The health bar does not return to full health

Somehow, the player's health is not fully restored. However, this was only for the health bar in the game, in reality the player health variable was restored but this was not shown in the player's health bar. This would be very confusing to the user. Therefore I went back into the "PlayerHealth" script, specifically the "Die" function and realised there was no code to update the slider back to full.

```
63              //resets player health
64              currentHealth = maxHealth;
65              healthBar.SetSliderMax(maxHealth)
```

I simply copied the statement in the "Start" method and added it to the "Die" function which should quickly solve the issue.

Test	Input	Justification	Expected Outcome	Actual Outcome
The player's health is fully restored after they die	N/A	Allows the game to function properly	The health bar will return to full health	The health bar returns to full health

However I have encountered another issue, when the player loses their last life, the heart sprite still remains filled. This should not happen as all the hearts must be empty when the player has lost all their lives. I believe my previous explanation on how the array being zero indexed ends up working perfectly was correct to begin with as when the player lost a life, they also lost the first heart, same with when they lost their second heart.

However on the third life this did not happen, my initial thought was to make the heart disappear right after the "currentLives" was decremented and before the game ends.

```
55      void Die()
56      {   //removes a life
57          currentLives --;
58          //update heart sprite after losing a life
59          hearts[currentLives].sprite = emptyHeart;
60
```

This seemed to have worked fine as when the player lost a life, they also lost the corresponding heart. This also worked for the final heart. However after the player had lost their final life, it seemed that the hearts array was still being indexed and updated even when the current lives was below 0.

To solve this problem I just ensured that "currentLives" has to be greater than or equal to 0. This prevents the index calculations from going out of range of the array.

```
55      void Die()
56      {
57          currentLives --;
58          // Updates the index to avoid out of range error
59          if (currentLives >= 0)
60              hearts[currentLives].sprite = emptyHeart;
61
```

This finally solved the issue as all the correct hearts are disappearing and there are no errors in the console.

Crosshair:

I first searched for a simple crosshair image online that I could use in my game. I then downloaded this image and imported it into my project. I set the texture type of the crosshair image to a sprite, similar to the hearts for the player lives.

Within the same canvas object I created a new image child object called “Crosshair” and referenced the image above as a sprite.

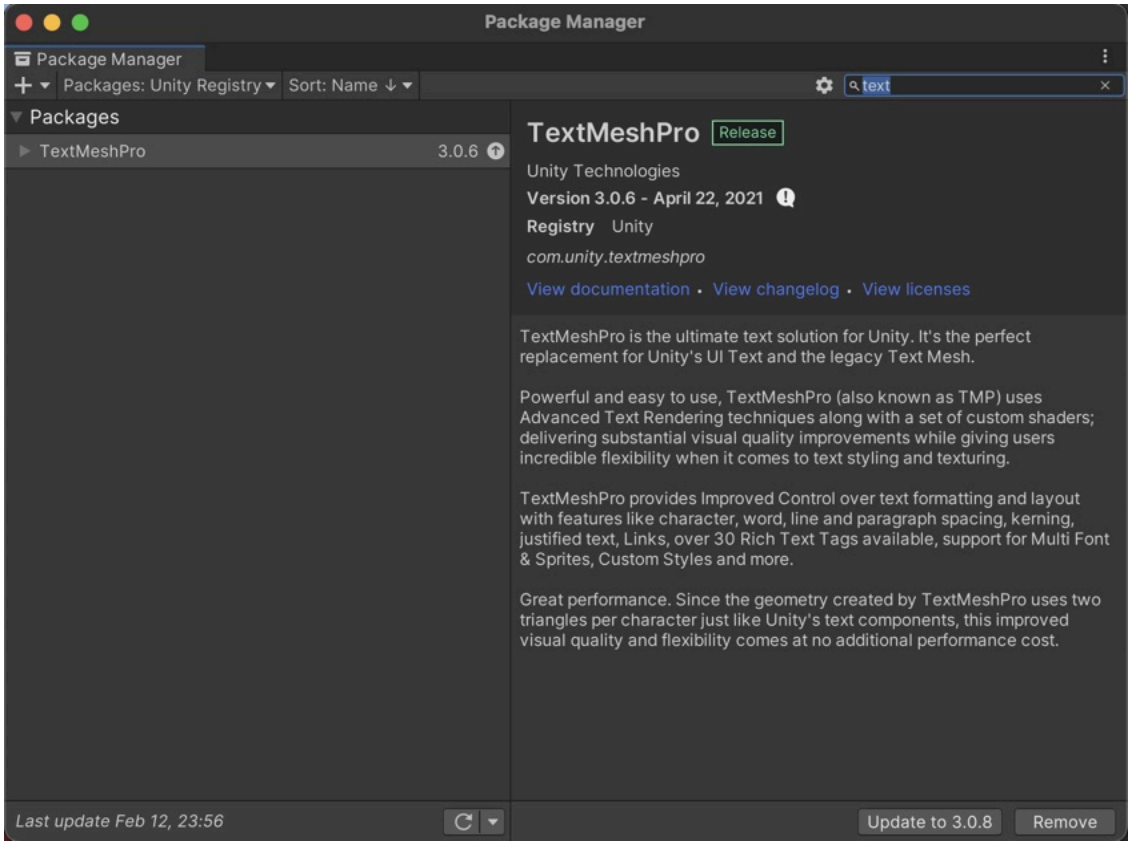
This is currently what the game looks like with the crosshair. I adjusted the dimensions of the crosshair to a suitable size so that it doesn’t completely block the enemies when the player is shooting at them.

Test	Input	Justification	Expected Outcome	Actual Outcome
The crosshair is displayed on the screen	N/A	Allows the player to aim at the enemy	The crosshair will be always displayed in the center of the screen	The crosshair is always displayed in the center
The bullets line up with the	N/a	Allows the player to use the	The bullets will shoot to the	The bullets shoot straight towards

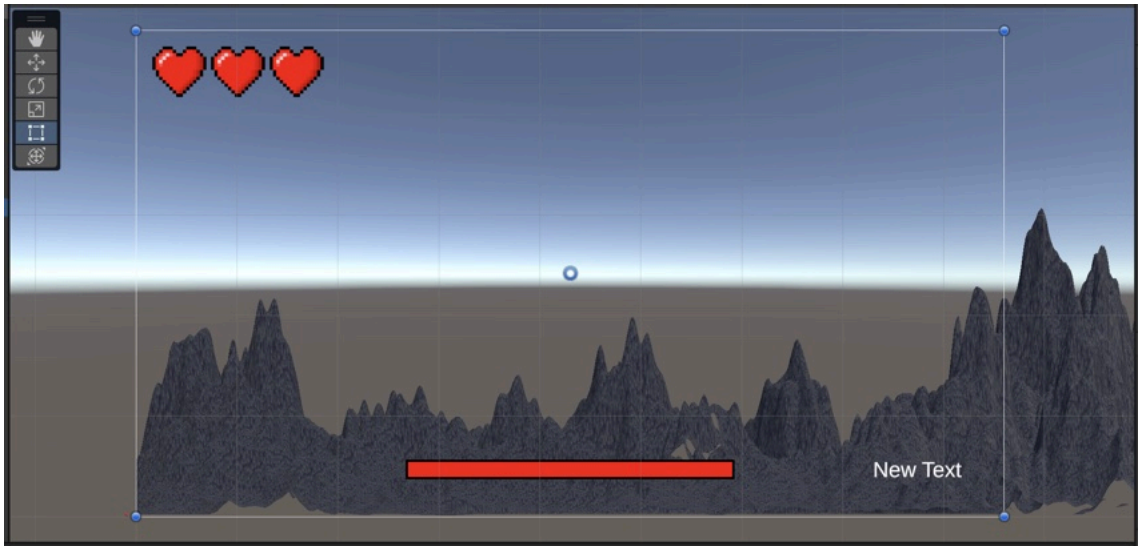
crosshair		crosshair normally	position of the crosshair	the crosshair.

Ammo Counter:

This won't be too difficult to implement as I will just continue developing the canvas that has already been created. Within this I will need to add a new text child object, but first I need to download and install the TextMeshPro package from the package manager.



After downloading this I will be able to add a text child object to the canvas. I then called this object ”ammunitionDisplay” and placed it at the bottom right of the canvas.



This is currently how the UI for the game looks, I will now need to script the text so that it shows the bullets that are currently in the magazine as well as the maximum capacity of the magazine. I decided to go back to the “GunProjectile” script to do this.

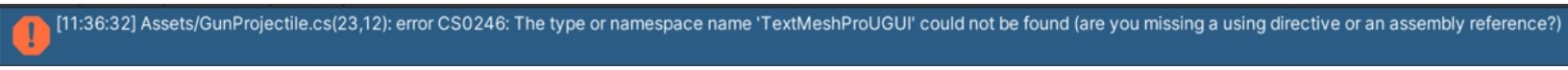
```
23      //Graphics
24      public TextMeshProUGUI ammunitionDisplay;
25
```

I first created a reference to the Text object which will allow me to manipulate it within the script.

```
36  private void Update()
37  {
38      myInput();
39
40      //set ammo display
41      if (ammunitionDisplay != null)
42          ammunitionDisplay.SetText(bulletsLeft + "/" + magazineSize);
43  }
```

Within the update function I added 2 new statements which allow the text to be displayed. The first if statement makes sure that “ammunitionDisplay‘ is not equal to null and does exist. This prevents any potential errors occurring if the text UI hasn’t been assigned properly. The next line sets the text for “ammunitionDisplay”, which is the no of bullets left in the magazine, “bulletsLeft” against the “magazineSize”

Test	Input	Justification	Expected Outcome	Actual Outcome
The ammo counter is displayed on the screen	N/A	Allows the player to keep track of their bullets	The ammo counter will be displayed on the bottom right of the screen	The game does not run
The ammo decreases when the gun is shot	N/a	Allows the player to keep track of their bullets	The bullets left will decrease and the magazine size will remain the same	The game does not run



I am faced with this issue. It seems that I am possibly missing a reference to a namespace which allows me to use the “TextMeshProUGUI” component. I realised that I need to add the “TMPro” namespace whenever I use TextMesh Pro.

```
4    using TMPro;
5
```

Test	Input	Justification	Expected Outcome	Actual Outcome
The ammo counter is displayed on the screen	N/A	Allows the player to keep track of their bullets	The ammo counter will be displayed on the bottom right of the screen	The crosshair is displayed on the bottom right of the screen
The ammo decreases when the gun is shot	Left mouse click	Allows the player to keep track of their bullets	The bullets left will decrease and the magazine size will remain the same	The bullets left decrease and the magazine size remains the same

Score Counter:

```
7    private int score = 0;
```

Within the “ScoreManager” script, I created a new integer variable called “score” and set it to 0 as this is what the player will start with.

```
9       //increases score by certain amount
10      public void IncreaseScore(int amount)
11      {
12          score += amount;
13      }
```

The function “IncreaseScore " simply increases the player’s score by a certain amount, this amount will be the score the player gains for each kill.

```
10      //score related
11      public int scorePerKill = 10;
12      private ScoreManager scoreManager;
13
```

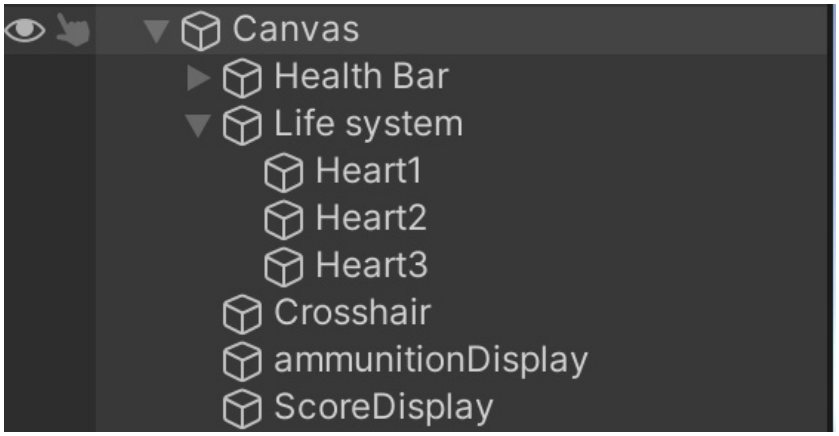
Within the “EnemyHealth” script I introduce a variable called ”scorePerKill”, I also created a new variable that will hold the “ScoreManager” script so that I can access the functions from that script.

```
15  void Start()
16  {
17      // sets player's health to full
18      currentHealth = maxHealth;
19      //reference to ScoreManage script
20      scoreManager = GameObject.FindObjectOfType<ScoreManager>();
21  }
```

Within the start function, I set the “scoreManager” to the game object that contains the “ScoreManager” component, this is just a new game object I created that only holds the “ScoreManger” script as a component.‘

```
26
27      if (currentHealth <= 0)
28      {
29          if (scoreManager != null)
30          {
31              scoreManager.IncreaseScore(scorePerKill);
32          }
33
34          Destroy(gameObject);
35      }
```

If the zombie dies, I first increase the score by the number held in the “scorePerKill” variable. This is done using the “IncreaseScore” function from the “ScoreManager” script.



I then created a new text UI object within the canvas using TextMesh pro .

```
9       //reference to TextMesh UI object
10      public TextMeshProUGUI ScoreDisplay;
```

Back in the “ScoreManager” script, I reference the text game object for the score.

```
23      private void Update()
24      {
25          if (ScoreDisplay != null)
26          {
27              ScoreDisplay.SetText("Score: " + score);
28          }
29      }
```

Within the update method, I first checked that “ScoreDisplay” is not equal to null, this checks if the game object is correctly referenced. I then used the “SetText” function to write the text for the UI. This will just be the string ”Score:” followed by the current score of the player.

I then placed the Score counter at the top left corner rather than the top right corner of the canvas. This was where the hearts were previously placed, I switched these positions to prevent the possible situation of the sore being so large that it goes off the screen.

I also reversed the order of the hearts as the hearts were still emptying from right to left, this didn't make much sense. The hearts now empty from left to right which feels a lot more natural.

Test	Input	Justification	Expected Outcome	Actual Outcome
The score counter is displayed on the screen	N/A	Allows the player to know how much progress they have made	The score counter will be displayed on the top left of the screen	The score counter is displayed on the top left of the screen
The score increases when an enemy is killed	N/a	Allows the player to keep track their score	The score will increase when the player kills an enemy	The score increases when the player kills an enemy

Round Counter:

The round counter practically takes the same approach just in different parts of the code.

```
8       private int round = 1;
9       //reference to TextMesh UI
10      public TextMeshProUGUI RoundDisplay;
```

Within the “RoundManager” script, I create and set the “round” variable to 1 rather than 0 as it makes more sense to begin the game from round 1. I also referenced the text UI object “RoundDisplay”.

```
12      //increments round
13      public void IncreaseRound()
14      {
15          round += 1;
16      }
```

I also created a new function called “IncreaseRound”, similar to “IncreaseScore”, this function just increments the round counter by 1.

```
22      //references
23      private RoundManager roundManager;
```

Within the “EnemySpawn” script, I created a new variable called “roundManger” which will hold the “RoundManager” script.‘

```
27      void Start()
28      {
29          currentEnemyLimit = initialEnemyLimit;
30          StartCoroutine(EnemySpawner());
31
32          roundManager = GameObject.FindObjectOfType<RoundManager>();
33      }
```

Within the start function, I set the “roundManager” to a new game object that only contains the “ScoreManager” script as a component.‘

```
61          //wait for all enemies to be killed before restarting
62          yield return new WaitUntil(()=> GameObject.FindGameObjectsWithTag("Enemy").Length == 0);
63
64          //increments the enemy limit
65          currentEnemyLimit += additionalEnemies;
66
67          if (roundManager != null)
68          {
69              roundManager.IncreaseRound();
70          }
```

RIght after all the enemies have been killed, I call the “IncreaseRound” function. This is done at the same time as when the enemy limit is increased.

```
18      private void Update()
19      {
20          if (RoundDisplay != null)
21          {
22              RoundDisplay.SetText("Round: " + round);
23          }
24      }
```

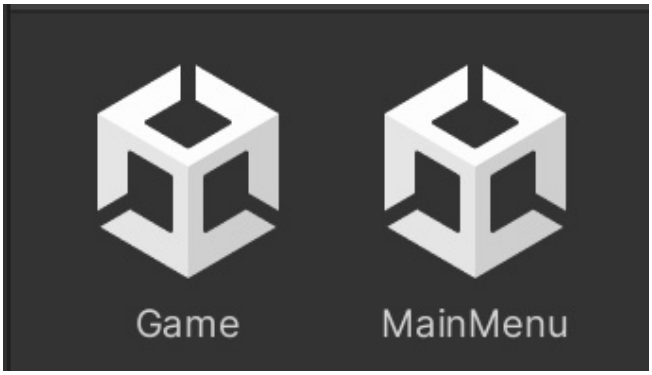
I then updated the round counter similar to the score counter.

This is currently how the round counter is positioned on the screen.

Test	Input	Justification	Expected Outcome	Actual Outcome
The round counter is displayed on the screen	N/A	This visualises the progress made for the player	The round counter will be displayed on the top right of the screen	The round counter is displayed on the top right of the screen
The round increases after all the zombies on the map have been killed	N/A	Allows the player to keep track of their progress	The round will increment after the player has killed all the enemies on the map	The round will increment after the player has killed all the enemies on the map

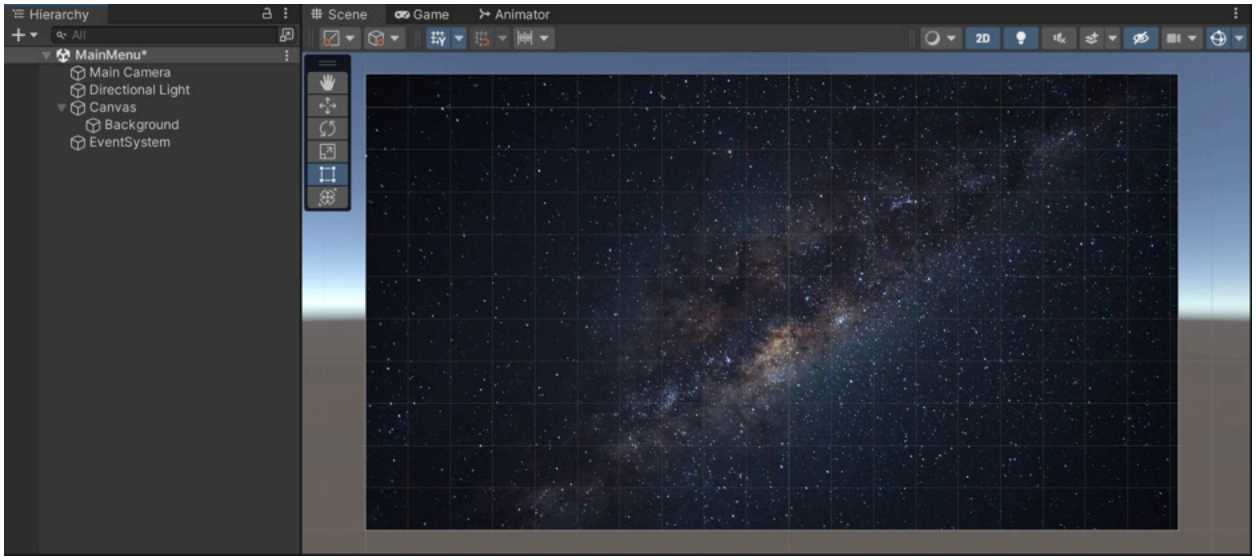
User Interface Screens:

I will now begin designing the user interface screens for the game. I will be constantly referring back to and designing upon the prototype user interfaces I created in the design section of this project. I will begin with designing and creating the main menu screen as this is arguably the most important interface due to how many connections it has with other interfaces (refer to user interface flow diagram in design, page 41).

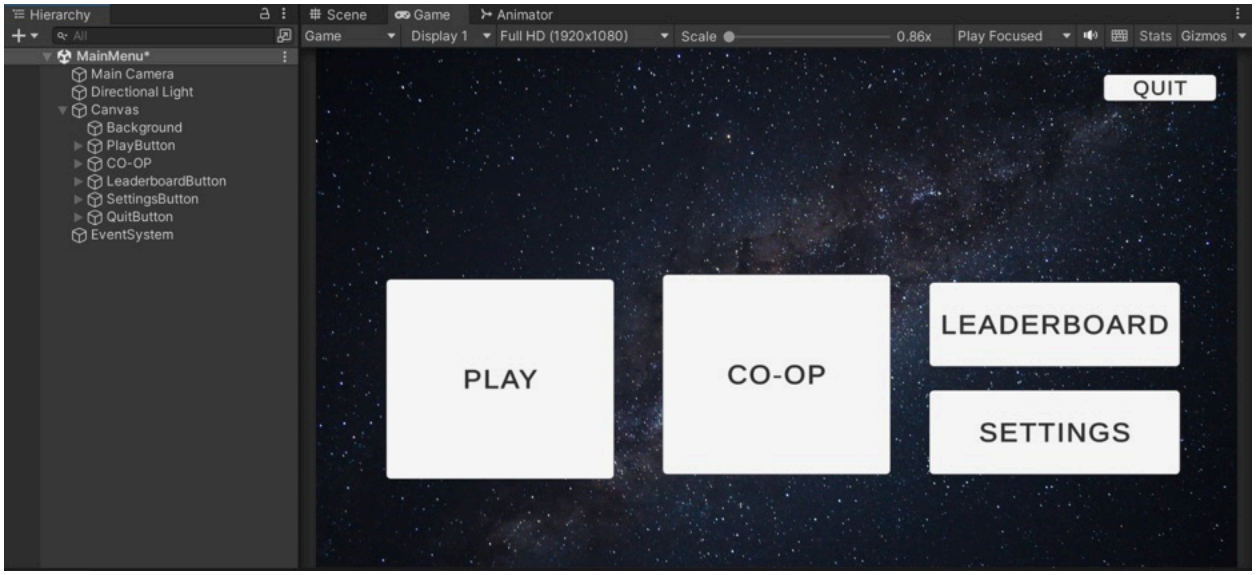


I first renamed the current scene I was using called “SampleScene” to “Game”. I then created a new scene called “Main Menu”, which will hold the canvas for the main menu. Some Interface screens will most likely have its own scene within the game. This allows me to switch between scenes when the player clicks on a button to go to a different interface.

I first created the canvas for the main menu in the new scene. I then added an image child to the canvas and used an anchor preset to set the image to stretch the entire canvas, this means that the image will cover the whole screen regardless of the screen size. I now need to add a source for the image object which will act as the background for the main menu. I decided to just find a space themed image on the internet for the background of the main menu.



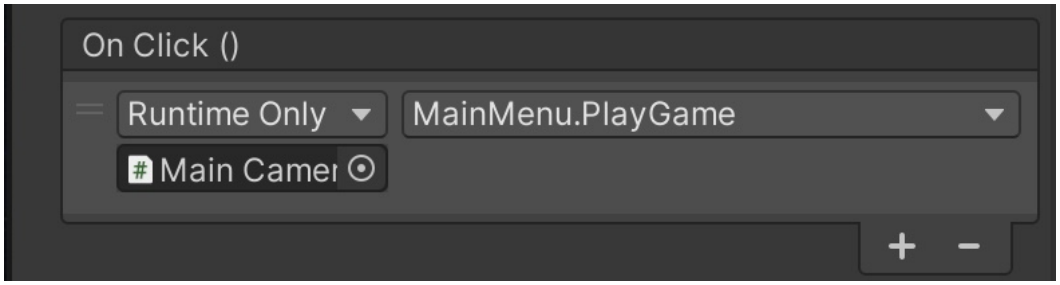
This is currently how the “MainMenu” scene is looking. There is a canvas with the selected image I have chosen as the background for the main menu. The background is a downloaded picture from the internet which has been imported as an asset and turned into a sprite. I will now begin designing the buttons for each option on the main menu as well as the title of the game. I decided to go with the name “Galaxy Gunner” as this was the name advised by my client within the interview(refer to interview in analysis section).



I have now added 5 different options for the game’s main menu. Each of these options were created by adding button objects to the canvas and setting each of their names to their corresponding option. I also decided to base the layout of the main menu UI to the interface designs that were made in the design section of the project.

```
6   public class MainMenu : MonoBehaviour
7   {
8
9       public void PlayGame()
10      {
11          SceneManager.LoadSceneAsync(1);
12      }
13  }
```

I created a new script called “MainMenu” and set it to an empty game object, this is just used to load the game scene from the main menu scene.



On the button that is used to load the game, I set some instructions of what will happen when it is clicked. I will reference the game object with the menu script and then reference the script “MainMenu” as well as the function”PlayGame”. This should allow the main menu scene to switch to the game scene when the player clicks the “Play” button.

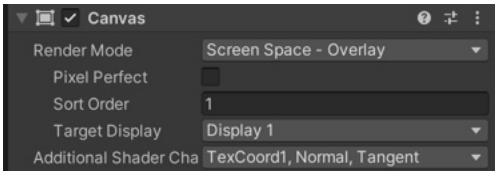
I added the title for the game, “Galaxy Gunner”, by adding a text UI object and centering it on the screen with a suitable size. I also made the quit button red as this button closes the game and is important to not be mixed up with anything else.

Test	Input	Justification	Expected Outcome	Actual Outcome
The Main Menu scene loads	Open the game application	This introduces the user into the game	The main menu scene will load in when the game is opened	The main menu scene loads in when the game is opened
The buttons work properly	Left mouse click on each button	Allows the player to navigate through the interfaces	Each buttons switches to their respective interfaces when clicked	Each button switches to their respective interfaces when clicked

Game Over Screen:

I am now going to begin designing and implementing the game over screen as this takes priority over the rest of the interfaces. This is because the game over screen gives the player a clear notice of when their game has ended and that they will have to start a new game. I am tackling the interface screens on the basis of their priority and importance within the game as I am limited on time and would not be able to implement every UI screen.

I first created a new canvas within the game scene as this will be the game over screen. I then added a new image UI object and named it to be the background. I stretched this background image to fit the whole screen and decided to use a completely black image as the background. I also increase the opacity of the image as this looks better for a game over screen.



I then set the canvas's sort order to 1, this ensures that the game over screen will always appear above the regular game UI. I will now add the buttons and "Game Over" text to the canvas.

For the text, I added a text UI object to the middle of the screen and enlarged it to a suitable size. I then added 2 new button UI objects to "Play Again" and to "Return to Main Menu". This is very similar to the UI design I created back in the design section.

```
6   public class GameOverScreen : MonoBehaviour
7   {
8       public void Setup()
9       {
10          gameObject.SetActive(true);
11      }
12  }
```

I added a new script to the background of the canvas which turns the background on. This also includes the "Game Over" text and the buttons as they are child objects to the background.

```
22      //reference to game over UI
23      public GameOverScreen GameOverScreen;
```

I then referenced the "GameOverScreen" script in the "PlayerHealth" script.

```
55      void Die()
56      {   //removes a life
57          currentLives --;
58
59          if(currentLives <= 0)
60          {
61              GameOverScreen.Setup();
62          }
```

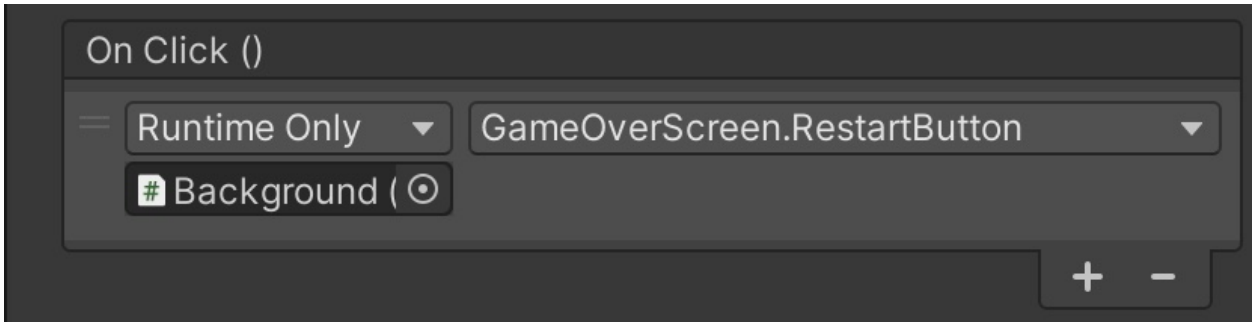
I previously just printed “Game Over” in the console as the game over screen wasn’t developed back then. I have now replaced that with a new statement. When the player has no more lives left, I will refer to “GameOverScreen” script and call the “Setup” function.

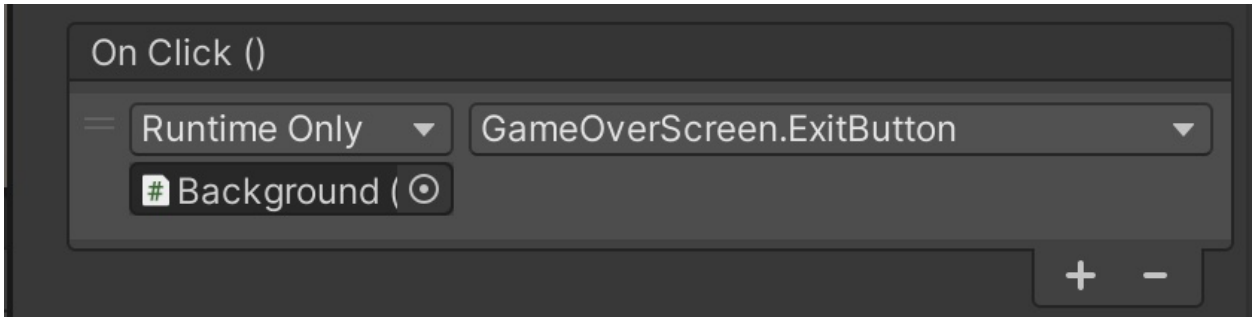
```
4   using UnityEngine.SceneManagement;
```

Back in the “GameOverScreen” script, I added Unity’s “SceneManagement” namespace as this will allow me to control when the different scenes are loaded.

```
13      public void RestartButton()
14      {
15          SceneManager.LoadScene("Game");
16      }
17
18      public void ExitButton()
19      {
20          SceneManager.LoadScene("MainMenu");
21      }
```

I created two functions, “RestartButton” which loads the “Game” scene, and the “ExitButton” which will load the “MainMenu” scene.





Similar to the “Play” button on the main menu screen, I will set instructions on what will happen when the button is clicked. For the “Play Again” button, I will refer back to “GameOverScreen” script in the background object and call the “Restart Button” function. For the “Return to Main Menu” button I will call the “Exit Button” function from the same script.

Test	Input	Justification	Expected Outcome	Actual Outcome
The game over screen load	N/A	This tells the player that their game is over	The game over screen will load as soon as the player loses all their lives	The game over screen loads as soon as the player loses all their lives
The buttons work properly	Left mouse click on each button	Allows the player to navigate through the interfaces	Each buttons switches to their respective interfaces when clicked	The buttons cannot be clicked

It occurs that when I attempt to click the buttons, the cursor disappears and goes back into the game. This prevents me from actually clicking the buttons. I then remembered that the cursor gets locked and made invisible at the start of the game. Therefore I decided to unlock the cursor and also make it visible to the player.

```
10      public void Setup()
11      {
12          // Ensure that the cursor is unlocked and visible when the game starts
13          Cursor.lockState = CursorLockMode.None;
14          Cursor.visible = true;
15          gameObject.SetActive(true);
16      }
```

I added both line 13 and 14 to the “Setup” function in the “GameOverScreen” script. Line 13 unlocks the cursor and line 14 makes it visible.

Test	Input	Justification	Expected Outcome	Actual Outcome
The buttons work properly	Left mouse click on each button	Allows the player to navigate through the interfaces	Each button will switch to their respective interfaces when clicked	Each button switches to their respective interfaces when clicked

Developer-Client Review 3:

These are the requirements from the success criteria that were looked at:

No.	Success Criteria	Justification	☑	Reference
1.	Display the Main menu and settings for audio and visual effects.	This allows the player to start the game at his own accord. The settings for audio and visual effects allows the player to make changes to make the game more enjoyable and personal.	☐	Vampire Survivors
2,	Display the current score in the game.	The current score tells the player how well they are doing in the game. The all-time high score tells the player his best score at the game and gives him a goal - to beat his own high score.	☐	Vampire survivors
10.	The player’s score increases by 10 points when an enemy is killed	This is done to reward the player for killing the enemy, the points will then be added onto their score	☐	COD Zombies
12.	The player has 3	This will give the user the ability to	☐	

	lives	respawn after dying a maximum of 3 times. This makes the game more forgiving as a single life will be too difficult.		
13.	Display the number of lives the player has.	The player will need to know how many lives they have left before the game is over. Icons can effectively be used to display this information.	□	
19.	Increasingly larger waves of enemies are spawned randomly across the map.	This makes the game challenging as the player has to fight more enemies. It also makes the game more interesting as the user wouldn't be able to predict where the enemies will spawn from.	□	COD Zombies
25.	Display a game over screen when the game ends	This is a clear indicator for the player that their game is over. The player can either play again or return to the main menu after this.	□	Vampire Survivors

Me: I have now coded the majority of the requirements from the success criteria as well as your suggestions from the previous review. This mainly includes the player's life system and the score and round counters. I also made a start at the user interfaces such as the main menu and the game over screen.

Client: That's great! I think it's about time you moved onto the next stage if we want to meet the time restraints for this project. Although I did hope the UI for the game would've been a bit more developed, however the core elements of the game have all been implemented well which is good.

Evaluation

This is the final stage of this project, and it will mainly consist of assessing the effectiveness and success of the game and how it has been developed. I will also be justifying why parts of the success criteria were not fully implemented as well as how I would've have approached them if I were to not have any time restrictions.

Post Development Test

Test No.	Description	Test Data	Expected Outcome	Actual Outcome	Video Reference
1	Check if the main menu loads when the game opens.	N/A	Main menu loads with the game title and 3 options.	Works as expected	0:00
1b	Check if player can go to leaderboard screen	Left mouse click on "Leaderboard" button	Game goes to the leaderboard screen	Feature not implemented	N/A
1c	Check if player can go to start a game	Left mouse click on "Play" button	A new game is started	Feature not implemented	N/A
1d	Check if player can go to Co-op screen	Left mouse click on "Co-op" button	Game	Feature not implemented	N/A
1e	Check if player can go to game options screen	Left mouse click on "Game Options" button	Game goes to game options screen	Feature not implemented	N/A
2	Check if score box is displayed on-screen	N/A	Score box is displayed on the top right of the screen	Works as expected	0:07
3	Check if top 5 highest scores are appear on the leaderboard	N/A	The top 5 highest scores are output onto the leaderboard	Feature not implemented	N/A

3b	Check if a new high score is added to the leaderboard	Valid: Finish the game with a score higher than the top 5 Invalid: Finish game with score lower than top 5	Valid: The new high score is inserted into the designated spot on the leaderboard. Invalid: Nothing happens	Feature not implemented	N/A
4	Check if the player's health bar is displayed on-screen	N/A	The player's health bar is displayed at the bottom of the screen	Works as expected	0:07
4b	Check if the enemies' health bars are displayed on-screen	N/A	Each enemies' respective health bar is displayed above each of their heads	Feature not implemented	N/A
5	Check if terrain loads in at the start of the game	N/A	The terrain is fully loaded into the game	Works as expected	0:07
6	Check if the player can look upwards	Move mouse up	The player looks up	Works as expected	0:08
6b	Check if the player can look downwards	Move mouse down	The player looks down	Works as expected	0:10
6c	Check if the player can look to the right	Move mouse to the right	The player looks right	Works as expected	https://youtu.be/CzMwqFDGyuc?si=dqLKQcmeRZM2N4Go&t=13
6d	Check if the player can look to the left	Move mouse to the left	The player looks left	Works as expected	https://youtu.be/CzMwqFDGyuc?si=dqLKQcmeRZM2N4Go&t=14

7	Check if player can move forwards	Valid: press “W” key Invalid: press “L” key	Valid: The player moves forwards Invalid: Nothing happens	Works as expected	https://youtu.be/CzMwqFDGyuc?si=dqLKQcmeRZM2N4Go&t=15
7b	Check if player can move backwards	Valid: press “S” key Invalid: press “P” key	Valid: The player moves backwards Invalid: Nothing happens	Works as expected	https://youtu.be/CzMwqFDGyuc?si=dqLKQcmeRZM2N4Go&t=18
7c	Check if player can move to the right	Valid: press “D” key Invalid: press “U” key	Valid: The player moves to the right Invalid: Nothing happens	Works as expected	https://youtu.be/CzMwqFDGyuc?si=dqLKQcmeRZM2N4Go&t=20
7d	Check if player can move to the left	Valid: press “A” key Invalid: press “E” key	Valid: The player moves to the left Invalid: Nothing happens	Works as expected	https://youtu.be/CzMwqFDGyuc?si=dqLKQcmeRZM2N4Go&t=22
7e	Check if player can jump up	Valid: press spacebar key Invalid: press “W” key	Valid: The player jumps up Invalid: player moves forward	Works as expected	https://youtu.be/CzMwqFDGyuc?si=dqLKQcmeRZM2N4Go&t=23
8	Check if player can sprint	Valid: hold “Shift” + “W”, “A”, “S” or “D” key Invalid: press “O” key	Valid: The player moves faster Invalid: the player does not move faster	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=27

9	Check if the player can shoot	Valid: Left mouse click while in game	Valid: The player shoots their gun	Works as expected	https://youtu.be/CzMwqFDGyuc?si=dqLKQcmeRZM2N4Go&t=39
9b	Check if bullets damages the enemy	Left mouse click at enemy	The enemy's health will decrease when hit with a bullet	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=49
9c	Check if the player can reload their gun	"R" key is pressed	the gun is reloaded	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=61
9d	Check if bullet deals extra damage when it hits the enemy's head	Left mouse click at enemy's head	The enemy's health will decrease more than when a bullet hits the rest of the body	Feature not implemented	N/A
9e	Check if bullet disappears when it hits with the enemy	Left mouse click at enemy	The bullet is destroyed when it collides with the enemy	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=49
9	Check if bullet disappears when it hits with the ground	Left mouse click at ground	The bullet is destroyed when it collides with the ground	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=58

11	Checks if the player gains points for killing an enemy.	Valid: left mouse click on the enemy until enemy health<=0. Invalid: left mouse click at the ground	Valid: 10 points are added to the player's score variable. Invalid: The player's score variable remains the same.	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=67
11b	Check if round increments after killing all the zombies	Kill all the zombies on the map	The round counter is incremented	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=130
12	Check the player starts with 100 health	N/A	When the game starts, the health variable is set to 100	Works as expected	0:07
13	Check if the player starts the game with 3 lives	N/A	When the game starts, the life variable is set to 3.	Works as expected	0:07
14	Check if the number of lives are displayed	N/A	3 heart sprites will appear at the top of the screen	Works as expected	0:07
15	Check if the player is giving spawn protection after respawning	N/A	The player will be invincible from damage for a short period of time after respawning.	Feature not implemented	N/A
16	Check if the enemy performs the chasing animation	Valid: Player moves into sight range of the enemy Invalid: Player is out of sight	Valid: The enemy performs a chasing animation	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=153

		range of the enemy	Invalid: No animation is performed		
16b	Check if the enemy performs the dying animation	Valid: Left mouse click on zombie until health <=0 Invalid: Left mouse click on zombie when bullets in magazine variable = 0	Valid: The enemy performs the dying animation Invalid: No animation is performed	Feature not implemented	N/A
16c	Check if the enemy performs the attacking animation	Valid: The player moves within range of melee attack from enemy Invalid: The player is out of range of melee attack from enemy	Valid: The enemy performs the attacking animation Invalid: No animation is performed	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=174
17	Check if footstep audio is played when player moves around	Valid: “W”, “A”, “S” or “D” key is pressed/held down Invalid: “L” key is pressed/held down	Valid: The player makes footsteps sound when they move Invalid: No sound is played	Feature not implemented	N/A
17b	Check if gunshot audio is played when player shoots	Left mouse click with gun	The gun will have a gunshot sound when shot	Feature not implemented	N/A
18	Check if the enemies spawn randomly across the map	N/A	The enemies are spawned randomly on the map	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQr

					XZz&t=130
18b	Check if the waves of enemies increase in size after each round	N/A	A wave of enemies are spawned, larger than the previous wave	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=130
19	Check if enemy can attack the player	N/A	The player will lose health after getting hit by the enemy	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=174
19b	Check if enemy can kill the player	N/A	The player will lose a life and the heart will become empty	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=179
20	Check if a boss spawns every 5 rounds	Valid: Player is on a round that is a multiple of 5 Invalid: Player is on a round that is not a multiple of 5	Valid: A stronger enemy with more health is spawned amongst the rest Invalid: No boss is spawned	Feature not implemented	N/A
21	Check if the player can pause the game	“Esc” key is pressed	The game should pause	Feature not implemented	N/A
21b	Check if player can unpause the game	“Esc” key is pressed when game is paused	The game should resume in the same state as it was when it was paused	Feature not implemented	N/A

22	Check if the player can save the game currently being played	Left mouse click on the “Quit and Save game” option in the pause menu in-game	The game should quit to the main menu and a new game save is stored	Feature not implemented	N/A
22b	Check if old game save can be loaded	Choose respective game save and left mouse click on “start game” option	The game should load with correct game data and player progress	Feature not implemented	N/A
22c	Check if the player can save multiple different games	N/A	The new game save should store separately from the rest	Feature not implemented	N/A
23	Checks if the player can access the shop	Valid: press the “G” key on the keyboard Invalid: Press the “L” key on the keyboard	The shop should load with all the correct items as well as their price Invalid: Shop doesn't load	Feature not implemented	N/A
23b	Check if player receives purchased item	Valid:Left mouse click on desired item Invalid: Right mouse click on desired item	Valid: The player instantly receives the item Invalid: The player does not receive the item	Feature not implemented	N/A
24	Check if game over screen is displayed	N/A	The game over screen is displayed	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=194
24b	Check if the player can start a new game from this screen	Left mouse click on “Play Again” button	The game starts again with the same chosen difficulty	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaym

					gsqxntKQrXZz&t=196
24c	Check if the player can return to main menu from this screen	Left mouse click on “Main Menu” button	The game goes to the main menu screen	Works as expected	https://youtu.be/CzMwqFDGyuc?si=iaymgsqxntKQrXZz&t=220
25	Check the player is able to join another person’s game	N/A	The player joins the other person’s game	Feature not implemented	N/A
26	Check if red tint appears on screen when hit by enemy	Get attacked by enemy	A red tin will appear on the screen to indicate being hit by the enemy	Feature not implemented	N/A

Success Criteria

Criteria No.	Success Criteria	Has success criteria been met?
1.	Display the Main menu and settings for audio and visual effects.	Partially
2.	Display the current score in the game.	Yes
3.	Display a leaderboard with the different scores achieved by different people.	No
4	Display the health bar of the players and enemies	Partially
5	A terrain map for the game	Yes
6	The player can look around freely	Yes
7	Player can move in all 4 directions as well as jump upwards	Yes

8	The player can sprint to move faster	Yes
9	The player can shoot their gun at the enemy	Yes
10	The player deals more damage with their weapon if they hit the zombie in the head.	No
11	The player's score increases by 10 points when an enemy is killed	Yes
12	The player starts with 100 health	Yes
13	The player has 3 lives	Yes
14	Display the number of lives the player has.	Yes
15	The player is given spawn protection upon respawning.	No
16	Dying animation for the enemy	No
17	Attacking animation for the enemy	Yes
18	Chasing animation for the enemy	Yes
19	Sound effect for gunshots	No
20	Increasingly larger waves of enemies are spawned randomly across the map.	Yes
21	The enemy can attack and kill the player	Yes
22	Spawn a Boss enemy with extreme health and damage every 5 rounds	No
23	Ability to pause the game	No
24	Ability to save the game	No
25	Player can use the in-game shop to upgrade weapon and physical attributes	No
26	Display a game over screen when the game ends	Yes
27	The screen will have a red tint for 3 seconds after the player takes damage.	No

28	The player regenerates their health after not taking damage for 10 seconds	No
29	The zombie despawns 5 seconds after it falls to the ground once killed.	Partially
30	The player can play with another person as a teammate.	No

I had only partially met criteria 1 as the main menu interface loads although the “Play” option is the only one that works. The “leaderboard”, “Settings”, “CO-OP”‘ and “Quit” options don’t work. This was mainly due to not having enough time to implement the feature, however it's also due to the fact I found the interfaces hard to implement and design.

Criteria 2 was easily met as this just required displaying the score variable as a text UI object on the canvas.

Criteria 3 was not met as the feature was not implemented. If I had more time to work on the development of the project, this feature could easily have been implemented. This would’ve simply required me saving the player’s achieved score and name and inserting it into the right position of an array of highscores.

Criteria 4 was partially met as I had only implemented the player’s health bar but not the enemy’s. Although this isn’t too much of an issue since having a health bar on each enemy would have created a lot of visual clutter, especially since I will have possibly hundreds of enemies being spawned.

Criteria 5 was met by designing the map using the terrain and terrain tool within Unity. I ended up creating a flat ground encircled by tall mountains. These mountains also act as a barrier to prevent the player or enemies walking off the map. I then added a rocky texture to the terrain to imitate that space theme.

Criteria 6 was easily met as I just implemented a simple first person camera system. Depending on the mouse input, the camera is simply rotated, I also clamped the vertical ends of the camera to prevent inverting

Criteria 7 was also accomplished, depending on the keyboard input, a vector is added to the player’s position causing it to move. For the jumping ability, I just added an upwards force to the player's rigidbody.

Criteria 8 was easily met. This was done by creating a new sprint speed variable, if the player holds the “Shift” key, the regular speed variable then gets updated to the sprint speed variable.

Criteria 9 was achieved by instantiating a bullet object and adding a force to it every time the player left mouse clicks. If the player was to hold the left mouse button, the bullet was continuously being instantiated with a delay between each bullet, before a force was added. I also ensured that the bullet travels towards the middle of the screen.

Criteria 10 had not been met as I did not have enough time to implement this feature. However I also felt this feature would've made the game a lot more complicated.

Criteria 11 - 14 had all been met. Criteria 11 was easily met as this just required the score variable to be incremented each time an enemy was killed. Criteria 12 and 13 simply required creating and setting the variables for the lives and the player's health. Criteria 14 required using an empty and filled heart sprite image on the canvas, and changing this depending on whether the player has that life or not.

Criteria 15 would have been a good feature to implement however due to time limitations I was not able to do this. This feature would've required disabling the "TakeDamage" function for the player for a certain period of time to prevent the player from taking any damage. I could've also temporarily changed the value of the damage inflicted by the enemy to 0 and return it back to normal after a certain amount of time.

Criteria 16 required me to implement a dying animation for the enemy, however I found it difficult to find a suitable death animation for the enemy from the asset store. I am also not familiar with any animation software so learning this for just a single animation wouldn't be a good way to spend my time. Especially since there are other more important features that could be implemented instead.

Criteria 17 and 18 had both also been implemented. Although it took some time to implement as I found animating the enemy object with correct timings was quite difficult. The animator controller was also quite confusing to begin with and took a lot of time to familiarise myself with.

Criteria 19 was not met due to the time limits for the project. Furthermore, this criteria didn't have much priority since it wouldn't have added much to the functionality of the game. I am also very unfamiliar with using audio in Unity.

Criteria 20 and 21 were also met. Criteria 20 required spawning the enemies in with different positions on the map. This was accomplished by finding random x and z coordinates for each zombie within a certain area on the map. After every wave of enemies, an additional amount is added to the enemy limit before they are instantiated, leading to increasingly larger waves of enemies. The zombie was then instantiated with a unique position on the map everytime. Criteria 21 simply required adding a box collider to the enemy's attacking hand; every collision on the player collider would call the "TakeDamage" function for the player.

Criteria 26 was accomplished by adding another canvas within the game scene and just putting it above the game canvas when the player lost all their lives. I also added buttons that allowed the player to either play again or to return back to the main menu.

Criteria 29 was only partially completed as the zombie disappears instantly rather than 5 seconds after dying, This was mainly due to the fact that I did not add a dying animation to the enemy, so I decided to just destroy the game object which was the best solution at the time.

The rest of the requirements were not completed mainly due to the fact that I did not have enough time to implement all these features.

Criteria 22 would've required me to instantiate an enemy game object with extreme health and damage every 5 rounds. This would not have been too hard to implement as I would just create a new function that instantiates the boss enemy and call it every multiple of 5 rounds. I would also make the enemy slightly different so that the player could distinguish between a regular and boss zombie, this could be making the boss slightly bigger than the rest.

Criteria 23 would require me to switch from the game canvas to the pause menu canvas, another way I could've done this was to just create a new background within the game canvas and turn this on when the "esc" key is pressed. However, I would also need to freeze a lot of things in the game, this would include the player movement, enemy NavMesh AI and animations.

Criteria 24 was easier than I thought it would be to implement at first. It would've just required me to save the position of the player, the progress in the game and any other player stats such as health and lives, etc. This would be saved onto a binary file on the computer. When the player tries to load a previously saved game, the saved binary file is used to update the game variables so that it seems as if the game is in the same state as when the player left it.

Criteria 25 will be difficult to implement because it will take a lot of work to create something like a shop as well as some sort of currency for the game. I am also not sure how I can develop an inventory for the player or whether I can create new items in the game.

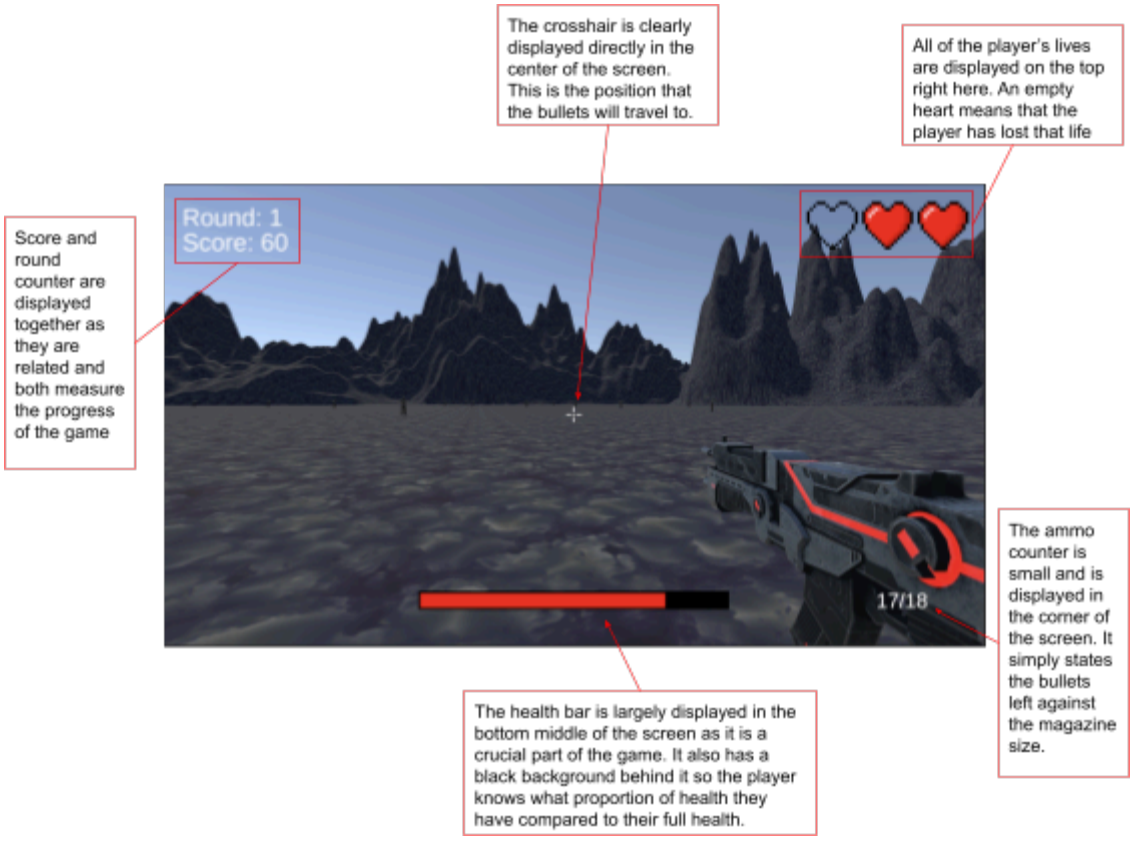
Criteria 27 and 28 are quite specific features which aren't too complicated. Criteria 27 would have required a red canvas background with high opacity to appear for a short time and then disappear. This implementation would've been very similar to that of the game over screen except the background only appears for a limited amount of time. Criteria 28 would require me to continuously check whether the "TakeDamage" function was called on the player. If it hasn't been called in over 10 seconds and the player's health isn't full, I would then regenerate the player's health.

Criteria 30 would be a very hard feature to implement on my own and I am not even too sure on how I can implement something like this. As these features start to dig into the idea of

networking and servers and all the other issues that come with it. I believe that this feature might be out of my scope as a single developer under these short time restrictions.

Usability Features:


The “Game Over” text is clearly displayed with a large font size to inform the player the game is over
The interface is very simple with only two possible options and the game title
GAME OVER
Play Again
Return to Main Menu
The two buttons allow the player to either immediately start playing another game or to return to the main menu



Alpha Test:

I decided to conduct an alpha test with all my stakeholders for the game. This will hopefully inform me of any errors or bugs that I have not encountered yet myself. I will give my stakeholder about a week to play around with the game and note down any observable issues with the game. This also includes obscure bugs which mysteriously appear even when the code is correct and makes sense. I will then review all the reports from each of the stakeholders.

Samuel Mandoza(Client):
Hey man, I'm glad we've already got to the alpha testing stage. I got a couple issues here and there throughout my playthrough last week. For starters, I think some zombies' hitboxes get turned off occasionally as a few zombies would just be completely invincible to hundreds of bullets. What confuses me is that the previously invincible zombie would turn back to normal after running around for a bit. In addition, the enemy's attack animation doesn't seem to always match with when the player takes damage. Finally, I still feel the game is lacking the space theme, I feel a space skybox would definitely help a lot. The game is still very enjoyable and slightly addicting.

Donte O'Neill:
Hey Ryan! I really enjoyed playing the game throughout last week and I'm excited to see how the game will perform and expand as time goes on. However, I did find a few noticeable issues with the game which I hope you could work on. I found the bullets don't always pass through the crosshair, this is especially evident when quickly strafing left to right with the character. The bullets don't seem to line up with the crosshair and almost have a large delay to them. I also found the hitbox for the zombie seems to turn off occasionally, I had noticed this when getting attacked by one of them. It's also missing some user interfaces and sound effects but other than that the game is in a great state.

Lexie Holt:
Hi Ryan, I've been playing the game for about a week now and I have a few things to talk about. First, I have to say I love the simplicity of the game, since I never end up staying with a game when they get too complicated. Although I feel that the game could have some sort of storyline to keep it interesting. Anyways, I didn't find any bugs necessarily, but I did think that the UI for the game was quite underdeveloped. Furthermore, the movement for the player felt quite blocky and especially when moving diagonally. I believe if you were able to address these issues the game would feel amazing, not to say that it's not great as of now.

Maya Wilson:
Hey, I just finished testing this game out and I can see a lot of potential for improvement mainly regarding the UI for the game. I also thought that a leaderboard would be a good feature to implement. I did also find some problems with the game. I quickly recognised how some bullets were flying out of trajectory for no apparent reason, my first thought was that the player's movement might be throwing off the bullets since this issue didn't occur when the player was standing still. The zombie also sometimes seemed to stop taking damage from the bullets, yet I wasn't able to understand what was causing this.

<u>Review of alpha test:</u>
I believe there is a positive consensus for the game despite the feedback of the lacklustre UI and missing features. However I will mainly be focusing on any bugs and problems that were found since this was an alpha test. I recognised that there is a probable issue with the enemy box collider since 3 people had mentioned the enemy not taking damage. However, I was confused when Samuel mentioned that the enemy would randomly return back to normal since my initial thought was that the enemy was not instantiated properly, possibly leading to missing colliders. I will have to investigate further to understand the root cause of the issue.

The bullets from the gun being "delayed" is probably due to the gun having a low shooting force. Increasing the shooting force of the gun should probably solve this issue. However, bullets being occasionally thrown off trajectory and not passing through the crosshair is most likely due to how the movement system interacts with the shooting system. Furthermore, the "blocky" feeling of the player movement is most definitely because the movement system was implemented by translating the player's coordinates. This means diagonal movement would technically be vertical and horizontal movement at the same time. Using the rigidbody of the

player and adding a force based on the direction of movement would feel a lot smoother. However it would have also been much harder to implement as I would have to take the drag and friction of the player into account.

I think this alpha test was quite useful as I was able to learn about some unprecedented issues with the game. I am also glad that the game doesn't have a ton of small bugs which would just end up being annoying to fix.

Beta Test Questionnaire:

For the beta test I gathered 10 people, including my stakeholders, to look at the game as a whole. They will need to evaluate things like the usability, experience and performance of the game. I will then provide them with a questionnaire where they can share their opinions about certain aspects of the game.

Galaxy Gunner Questionnaire

This survey will be used as part of a beta test to evaluate the success of "Galaxy Gunner". Please take your time and answer the following question honestly as your responses will be anonymously recorded.

1. How much did you play the game during the beta test?

- ◯ A great deal
- ◯ A lot
- ◯ A moderate amount
- ◯ A little
- ◯ None at all

2. How satisfied are you with the UI for the game?

- ◯ Very satisfied
- ◯ Satisfied
- ◯ Neither satisfied nor dissatisfied
- ◯ Dissatisfied
- ◯ Very dissatisfied

3. Would you recommend/play this game with to/a friend?

- ◯ Yes
- ◯ No

4. How complicated did the game feel ?

- ◯ A great deal
- ◯ A lot
- ◯ A moderate amount
- ◯ A little
- ◯ None at all

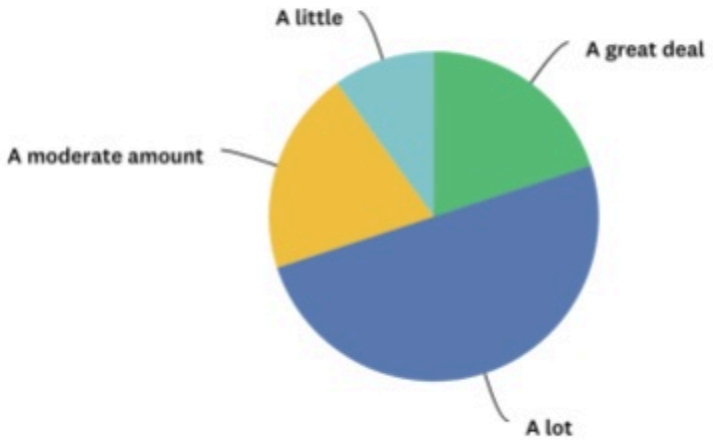
5. How would you rate your experience playing Galaxy Gunner out of 5

- ◯ 1(worst)
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5(best)

Done

How much did you play the game during the beta test?

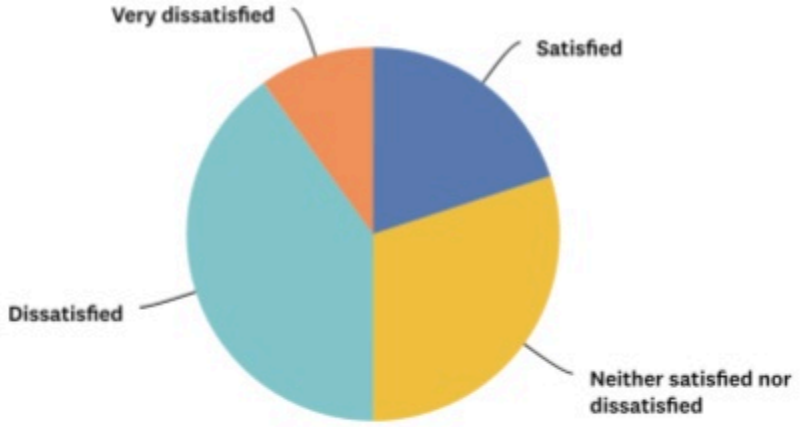
Answered: 10 Skipped: 0



It seems as though most people played the game for a good amount of time throughout the beta test session. This is good as it will mean the answers to the following question would be more accurate and reliable.

How satisfied are you with the UI for the game?

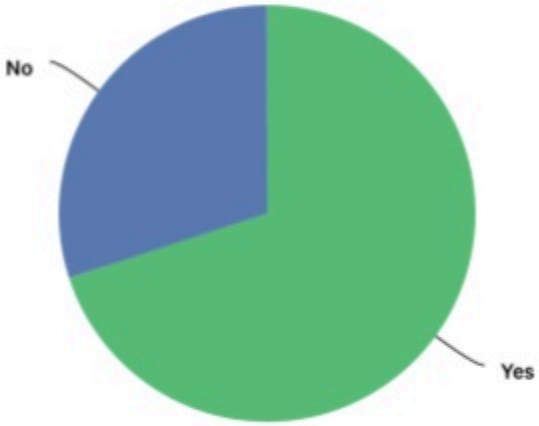
Answered: 10 Skipped: 0



I am not surprised by these results as I did not end up implementing the majority of interfaces that I had initially planned out. This left the UI with buttons that didn't work and minimal functionality which would be a very bad look for any game.

Would you recommend/play this game with to/a friend?

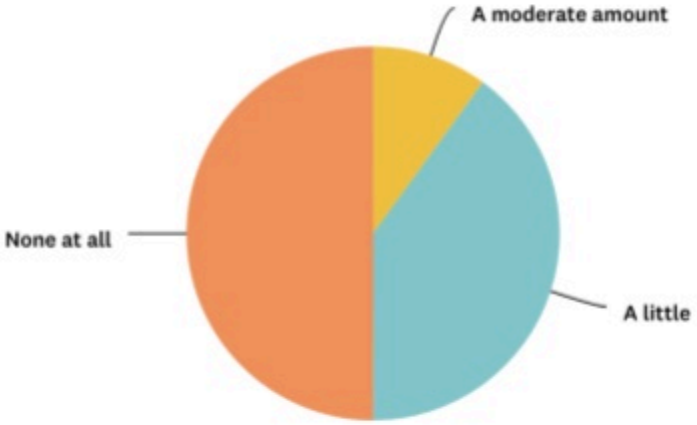
Answered: 10 Skipped: 0



The results of this question seems like there is possibly a huge demand for some sort of multiplayer feature for the game. This gives the game a direction to move towards in the future. It also means that the game is likely to spread tremendously in awareness from word of mouth. This will be really good for the game in the long run as the game may be able to gather more possible users.

How complicated did the game feel ?

Answered: 10 Skipped: 0



The majority answers here led me to believe that the game is quite simple, which was what I was aiming for. I feel like the simplicity of the game would allow it to reach a much larger demographic of gamers which is always good when starting out the game. Of course the game will get more complex overtime as more features are added, but hopefully by this time, the game will already have an established fan base.

The majority of answers here are again positive, which is a great relief as I feel that I was able to successfully create a game for my client. Although there was one person who rated the game quite poorly, this was fair as I feel like the game was still missing a lot of features and isn't even close to being considered a fully finished game.

Maintenance

Maintenance is the process of managing the game following its initial release. I currently haven't added any specific features to the game that helps with its maintenance, although this is something I'd definitely implement in the future. Things like an automated error response algorithm or something that allows the user to report any bugs or glitches to me or any other future developers on a team, would be extremely helpful. This gives the developers something to focus their attention on. Furthermore, consistent updates and additional features for the game will keep current players engaged for a much longer period of time and possibly even attract new players to the game.

I also kept in mind the maintainability of the code I wrote during the development stages of the project. I did this by including detailed comments within the code where necessary, also using suitable variable and function names. This will prove to be very useful if I were to have other developers join into the project. They will be able to quickly understand the code I've already

written and be able to quickly begin building upon this code. Furthermore, the use of many functions makes the code modular and allows for isolated testing. The modular aspect of this code can also provide reusable components that are already guaranteed to work, which ends up being a huge time save for developers.

Approach to limitations

I will now address the limitations I introduced during the Analysis section and explain how I would've approached each of them.

Requirements	Justification
The gun will have recoil	Even though it adds realism to the game, it will be difficult to code and isn't necessary for the game to function.

To implement this feature it would require me to add an extra force to the instantiated bullets other than the one that sends the bullet flying forwards. I would have to do this for every bullet and set up a pattern for the recoil. The gun will also need to return to its original position after all the recoil. This seems too difficult to implement and I would much rather spend my time developing features with a higher priority.

Requirements	Justification
Different parts of the game will have specific music tracks	This would provide a better atmosphere in the game. It can also create different emotions for the player allowing him to be more immersed into the game.

To accomplish this requirement, I would first need to find suitable music tracks for my game that are preferably copyright free, or make my own music which would unfortunately take too long. I will then need to be able to switch the music being played depending on the scene the game is currently in.

Requirements	Justification
The player will have multiple extra movement abilities and techniques such as sliding and grappling	These could improve the enjoyability of the game as the player is more in control, however it will be extremely hard to implement within the time limits I am given.

This will not only take too long to implement, but would also be quite difficult as well. This is due to the different things I will have to take into consideration when the player is flying through the sky after grappling for example. I will obviously need to consider the physics of how this would work, but I would also need to provide animations for these abilities as well as how they would interact with already implemented features such as shooting.

Requirements	Justification
The player will be able to play the game online with other people.	This makes the game more interesting as the player can interact and play with new people.

This will be a very difficult feature to implement, especially for the scale of the project I am creating. I may need to have access to a server, which allows me to communicate with players over the internet. I would need to implement a matchmaking algorithm to join different players together, these players would also need to be authenticated as real people so that there would be not any security issues or players that are possibly cheating.

Requirements	Justification
The game will have many realistic and detailed graphics and animations	This would make the game more immersive and also attract more people to play the game.

To accomplish this I would need to have access to and run a decent graphics and animation development software. This would also take a lot of time to complete as I will need to familiarise myself with how to use the software. The game already has a few animations within the game, making extra custom animations does not seem worth it for the time being.

Appendix

Code Listing

PlayerMovement.cs

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

public class PlayerMovement : MonoBehaviour
{
   public float speed = 5.0f; // Determines the speed of the player
   public float sprintSpeed = 10.0f; //Determines the speed of the player when
sprinting
   public float jumpForce = 5.0f; //Determines the jump height of the player
   public bool isOnGround = true;
   private float horizontalInput;
   private float forwardInput;
   private Rigidbody playerRb;
```

```
    // Start is called before the first frame update
    void Start()
    {
        playerRb = GetComponent<Rigidbody>(); // allows access to the player's
rigidbody component
    }

    // Update is called once per frame
    void Update()
    {
        //Get player input for horizontal and forward movement
        horizontalInput = Input.GetAxis("Horizontal");
        forwardInput = Input.GetAxis("Vertical");

        //get forward and right directions from cam orientation
        Vector3 forwardDirection = Camera.main.transform.forward;
        Vector3 rightDirection = Camera.main.transform.right;

        forwardDirection.y = 0f;
        rightDirection.y = 0f;

        //Normalize to ensure consistent speed in all directions
        forwardDirection.Normalize();
        rightDirection.Normalize();

        // Calculate the movement direction based on camera orientation
        Vector3 moveDirection = forwardDirection * forwardInput + rightDirection *
horizontalInput;
        moveDirection.Normalize();

        float currentSpeed = speed;

        if (Input.GetKey(KeyCode.LeftShift))
        {
            currentSpeed = sprintSpeed;
        }

        // Move the player
        transform.Translate(moveDirection * currentSpeed * Time.deltaTime);

        //Lets the player jump
```

```
        if(Input.GetKey(KeyCode.Space) && isOnGround)
        {
            playerRb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
            isOnGround = false;
        }
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Ground"))
        {
            isOnGround = true;
        }
    }
}
```

PlayerCam.cs

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

public class PlayerCam : MonoBehaviour
{
    // sensitivites for horizontal and vertical mouse movement
    public float SensX;
    public float SensY;
    //orientation of the player model that follows the camera
    public Transform orientation;
    //X and Y rotation of the camera
    float xRotation;
    float yRotation;

    // Start is called before the first frame update
    private void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;//keeps the cursor at the center of
the screen
        Cursor.visible = false;//makes the cursor invisible
    }

    // Update is called once per frame
```

```
    private void Update()
    {
        // to get mouse input
        float mouseY =  Input.GetAxisRaw("Mouse Y") * Time.deltaTime * SensY;
        float mouseX =  Input.GetAxisRaw("Mouse X") * Time.deltaTime * SensX;

        //update rotation based on mouse input
        yRotation += mouseX;
        xRotation -= mouseY;

        // to rotate cam and orientation
        transform.rotation = Quaternion.Euler(xRotation, yRotation, 0);
        orientation.rotation = Quaternion.Euler(0, yRotation, 0);

        //clamps vertical rotation
        xRotation = Mathf.Clamp(xRotation, -90f, 90f);


    }
}
```

MoveCam.cs

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

public class MoveCam : MonoBehaviour
{
    public Transform cameraPosition;

    // Update is called once per frame
    private void Update()
    {
        transform.position = cameraPosition.position;
    }
}
```

PlayerHealth.cs

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

```
using UnityEngine.UI;

public class PlayerHealth : MonoBehaviour
{
   //Health  variables
   [SerializeField] private float maxHealth;
   private float currentHealth;
   public HealthBar healthBar;

   //Life variables
   public int maxLives;
   public int currentLives;
   public Image[] hearts;

   //reference to heart sprites
   public Sprite emptyHeart;
   public Sprite filledHeart;

   //reference to game over UI
   public GameOverScreen GameOverScreen;

   // Start is called before the first frame update
   void Start()
   {
       // sets player's health to full
       currentHealth = maxHealth;

       //sets slider to full
       healthBar.SetSliderMax(maxHealth);

       //sets player's lives to full
       currentLives = maxLives;

       //sets all heart sprites to filled
       foreach (Image heart in hearts)
       {
           heart.sprite = filledHeart;
       }
   }

   public void TakeDamage (float amount)
   {
```

```
        currentHealth -= amount;
        healthBar.SetSlider(currentHealth);
        if (currentHealth <= 0)
        {
            Die();
        }

    }

    void Die()
    {
        currentLives --;
        // Updates the index to avoid out of range error
        if (currentLives >= 0)
            hearts[currentLives].sprite = emptyHeart;

        if (currentLives <= 0)
        {
            GameOverScreen.Setup();
        }
        else
        {
            currentHealth = maxHealth;
            healthBar.SetSliderMax(maxHealth);
        }
    }
}
```

EnemyAI.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class EnemyAI : MonoBehaviour
{
    //references
    public NavMeshAgent enemy;
    public Transform player;
```

```
    public LayerMask whatIsGround, whatIsPlayer;
    public Animator enemyAnimator;
    public BoxCollider boxCollider;

    //For Patrolling
    public Vector3 walkPoint;
    bool walkPointSet;
    public float walkPointRange;


    //For Attacking
    public float timeBetweenAttacks;
    public float damage;
    bool alreadyAttacked;

    //states
    public float sightRange, attackRange;
    public bool playerInSightRange, playerInAttackRange;

    private void Awake()
    {
        player = GameObject.Find("Player").transform;
        enemy = GetComponent<NavMeshAgent>();
        enemyAnimator = GetComponent<Animator>();
        boxCollider = GetComponentInChildren<BoxCollider>();
    }
    //Update is called once per frame
    private void Update()
    {
        //Check for sight and attack range
        playerInSightRange = Physics.CheckSphere(transform.position, sightRange,
whatIsPlayer);
        playerInAttackRange = Physics.CheckSphere(transform.position, attackRange,
whatIsPlayer);

        //Conditions for each state
        if(!playerInSightRange && !playerInAttackRange) Patrolling();
        if(playerInSightRange && !playerInAttackRange) ChasePlayer();
        if(playerInSightRange && playerInAttackRange) AttackPlayer();
    }

    private void Patrolling()
```

```
    {
        if (!walkPointSet) SearchWalkPoint();

        if (walkPointSet)
            enemy.SetDestination(walkPoint);

        //calculates distance to walkpoint
        Vector3 distanceToWalkPoint = transform.position - walkPoint;

        //Walk point reached
        if(distanceToWalkPoint.magnitude < 1f)
            walkPointSet = false;

    }

    private void SearchWalkPoint()
    {
        //finds random number within range
        float randomZ = Random.Range(-walkPointRange, walkPointRange);
        float randomX = Random.Range(-walkPointRange, walkPointRange);

        //calculates new walkpoint
        walkPoint = new Vector3(transform.position.x + randomX, transform.position.y,
transform.position.z + randomZ);

        if (Physics.Raycast(walkPoint, -transform.up, 2f, whatIsGround))
            walkPointSet = true;
    }

    private void ChasePlayer()
    {   //makes enemy chase player by its position
        enemy.SetDestination(player.position);
    }

    private void AttackPlayer()
    {
        if(!alreadyAttacked)
        {
            if (!enemyAnimator.GetCurrentAnimatorStateInfo(0).IsName("Z_Attack"))
            {
                //triggers the attack animation
                enemyAnimator.SetTrigger("Attack");
```

```
                //make sure enemy doesn't move
                enemy.SetDestination(transform.position);
                //make sure that the enemy looks at the player
                transform.LookAt(player);
            }

            alreadyAttacked = true;
            //causes enemy to attack after specified time
            Invoke(nameof(ResetAttack), timeBetweenAttacks);
        }
    }

    private void OnTriggerEnter(Collider collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            player.GetComponent<PlayerHealth>().TakeDamage(damage);
        }
    }

    //resets the enemy attack
    private void ResetAttack()
    {
        alreadyAttacked = false;
    }

    //turns the box collider on
    private void EnableAttack()
    {
        boxCollider.enabled = true;
    }

    //turns the box collider off
    private void DisableAttack()
    {
        boxCollider.enabled = false;
    }

}
```

EnemyHealth.cs

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

public class EnemyHealth : MonoBehaviour
{
   [SerializeField] private float maxHealth;
   private float currentHealth;

   //score related
   public int scorePerKill = 10;
   private ScoreManager scoreManager;

   // Start is called before the first frame update
   void Start()
   {
       // sets player's health to full
       currentHealth = maxHealth;
       //reference to ScoreManage script
       scoreManager = GameObject.FindObjectOfType<ScoreManager>();
   }

   public void TakeDamage (float amount)
   {
       currentHealth -= amount;

       if (currentHealth <= 0)
       {
           if (scoreManager != null)
           {
               scoreManager.IncreaseScore(scorePerKill);
           }

           Destroy(gameObject);
       }
   }
}
```

EnemySpawn.cs

```
using System.Collections;
```

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

public class EnemySpawn : MonoBehaviour
{
   public GameObject Enemy;

   //max no. of enemies
   public int initialEnemyLimit;
   public int currentEnemyLimit;
   //enemeies added after each wave
   public int additionalEnemies;


   //range of x and z values
   public float xPositionMin;
   public float xPositionMax;
   public float zPositionMin;
   public float zPositionMax;

   //references
   private RoundManager roundManager;


   // Start is called before the first frame update
   void Start()
   {
       currentEnemyLimit = initialEnemyLimit;
       StartCoroutine(EnemySpawner());

       roundManager = GameObject.FindObjectOfType<RoundManager>();
   }

   IEnumerator EnemySpawner()
   {
       while (true)
       {
           //current no. of enemies
           int enemyCount = 0;

           while (enemyCount < currentEnemyLimit)
           {
```

```
                //generates random x and z value
                float xPosition = Random.Range(xPositionMin,xPositionMax);
                float zPosition = Random.Range(zPositionMin,zPositionMax);

                //activates enemy game object
                Enemy.SetActive(true);

                //spawns in enemy
                Instantiate (Enemy, new Vector3(xPosition, 3, zPosition),
Quaternion.identity);

                //time betwween each enemy spawn
                yield return new WaitForSeconds(0.05f);
                enemyCount += 1;
                //deactivates enemy game object
                Enemy.SetActive(false);

            }
            //wait for all enemies to be killed before restarting
            yield return new WaitUntil(()=>
GameObject.FindGameObjectsWithTag("Enemy").Length == 0);

            //increments the enemy limit
            currentEnemyLimit += additionalEnemies;

            if (roundManager != null)
            {
                roundManager.IncreaseRound();
            }
        }
    }
}
```

BulletCollisions.cs

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

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

```
//max no. of enemies
public int initialEnemyLimit;
public int currentEnemyLimit;
//enemeies added after each wave
public int additionalEnemies;


//range of x and z values
public float xPositionMin;
public float xPositionMax;
public float zPositionMin;
public float zPositionMax;

//references
private RoundManager roundManager;


// Start is called before the first frame update
void Start()
{
    currentEnemyLimit = initialEnemyLimit;
    StartCoroutine(EnemySpawner());

    roundManager = GameObject.FindObjectOfType<RoundManager>();
}

IEnumerator EnemySpawner()
{
    while (true)
    {
        //current no. of enemies
        int enemyCount = 0;

        while (enemyCount < currentEnemyLimit)
        {
            //generates random x and z value
            float xPosition = Random.Range(xPositionMin,xPositionMax);
            float zPosition = Random.Range(zPositionMin,zPositionMax);

            //activates enemy game object
            Enemy.SetActive(true);
```

```
                //spawns in enemy
                Instantiate (Enemy, new Vector3(xPosition, 3, zPosition),
Quaternion.identity);

                //time betwween each enemy spawn
                yield return new WaitForSeconds(0.05f);
                enemyCount += 1;
                //deactivates enemy game object
                Enemy.SetActive(false);

            }
            //wait for all enemies to be killed before restarting
            yield return new WaitUntil(()=>
GameObject.FindGameObjectsWithTag("Enemy").Length == 0);

            //increments the enemy limit
            currentEnemyLimit += additionalEnemies;

            if (roundManager != null)
            {
                roundManager.IncreaseRound();
            }
        }
    }
}
```

GunProjectile.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
 public class GunProjectile : MonoBehaviour
{   //bullet object
    public GameObject bullet;

    public float shootForce;
    //Gun stats
    public float timeBetweenShooting, reloadTime;
    public int magazineSize;
    public int bulletsLeft, bulletsShot;
    public bool allowButtonHold;
```

```
    private bool readyToShoot, shooting, reloading;

    //reference camera
    public Camera PlayerCam;
    public Transform attackPoint;

    //Graphics
    public TextMeshProUGUI ammunitionDisplay;

    private bool allowInvoke = true;

    private void Awake()
    {
        // make sure magazine is full
        bulletsLeft = magazineSize;
        readyToShoot = true;

    }

    private void Update()
    {
        myInput();

        //set ammo display
        if (ammunitionDisplay != null)
            ammunitionDisplay.SetText(bulletsLeft + "/" + magazineSize);
    }

    private void myInput()
    {
        //check if allowed to hold down button to shoot
        if (allowButtonHold) shooting = Input.GetKey(KeyCode.Mouse0);
        else shooting = Input.GetKeyDown(KeyCode.Mouse0);

        //Reloading
        if (Input.GetKeyDown(KeyCode.R) && bulletsLeft < magazineSize && !reloading)
Reload();
        //Automatically reload when shooting with no ammo
        if (readyToShoot && shooting && !reloading && bulletsLeft == 0) Reload();

        //Shooting
```

```
        if (readyToShoot && shooting && !reloading && bulletsLeft > 0)
        {
            //Set bullets shot to 0
            bulletsShot = 0;

            Shoot();
        }


    }

    private void Shoot()
    {
        readyToShoot = false;

        //Find the exact hit position of bullet using a raycast
        Ray ray = PlayerCam.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
        RaycastHit hit;

        //Check if ray hits something
        Vector3 targetPoint;
        if (Physics.Raycast(ray, out hit))
            targetPoint = hit.point;
        else
            targetPoint = ray.GetPoint(75);

        //calculates direction from attackPoint to targetPoint
        Vector3 direction = targetPoint - attackPoint.position;

        //instantiate bullet
        GameObject currentBullet = Instantiate(bullet, attackPoint.position,
Quaternion.identity);

        currentBullet.tag = "Bullet";
        //Rotate bullet to shoot direction
        currentBullet.transform.forward = direction.normalized;

        //add forces to the bullet
        currentBullet.GetComponent<Rigidbody>().AddForce(direction.normalized *
shootForce, ForceMode.Impulse);

        //bullets in mag decrease by 1
```

```
        bulletsLeft--;
        //bullets shot increase by 1
        bulletsShot++;

        //Invokes ResetShot function
        if (allowInvoke)
        {
            Invoke("ResetShot", timeBetweenShooting);
            allowInvoke = false;
        }
    }

    private void ResetShot()
    {
        //allows shooting and invoke again
        readyToShoot = true;
        allowInvoke = true;
    }

    private void Reload()
    {
        reloading = true;
        Invoke("ReloadFinished", reloadTime);
    }

    private void ReloadFinished()
    {
        bulletsLeft = magazineSize;
        reloading = false;
    }
}
```

HealthBar.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class HealthBar : MonoBehaviour
{
    public Slider healthSlider;
```

```
    // updates the value of the slider
    public void SetSlider(float amount)
    {
        healthSlider.value = amount;
    }

    //sets the slider to the max value
    public void SetSliderMax(float amount)
    {
        healthSlider.maxValue = amount;
        SetSlider(amount);
    }
}
```

RoundManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class RoundManager : MonoBehaviour
{
    private int round = 1;
    //reference to TextMesh UI
    public TextMeshProUGUI RoundDisplay;

    //increments round
    public void IncreaseRound()
    {
        round += 1;
    }

    private void Update()
    {
        if (RoundDisplay != null)
        {
            RoundDisplay.SetText("Round: " + round);
        }
    }
}
```

ScoreManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class ScoreManager : MonoBehaviour
{
   private int score = 0;
   //reference to TextMesh UI object
   public TextMeshProUGUI ScoreDisplay;

   //increases score by certain amount
   public void IncreaseScore(int amount)
   {
       score += amount;
   }

   private void Update()
   {
       if (ScoreDisplay != null)
       {
           ScoreDisplay.SetText("Score: " + score);
       }
   }
}
```

MainMenu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{

   public void PlayGame()
   {
```

```
        SceneManager.LoadSceneAsync(1);
    }
}
```

GameOverScreen.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;



public class GameOverScreen : MonoBehaviour
{
    public void Setup()
    {
        // Ensure that the cursor is unlocked and visible when the game starts
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
        gameObject.SetActive(true);
    }

    public void RestartButton()
    {
        SceneManager.LoadScene("Game");
    }

    public void ExitButton()
    {
        SceneManager.LoadScene("MainMenu");
    }
}
```