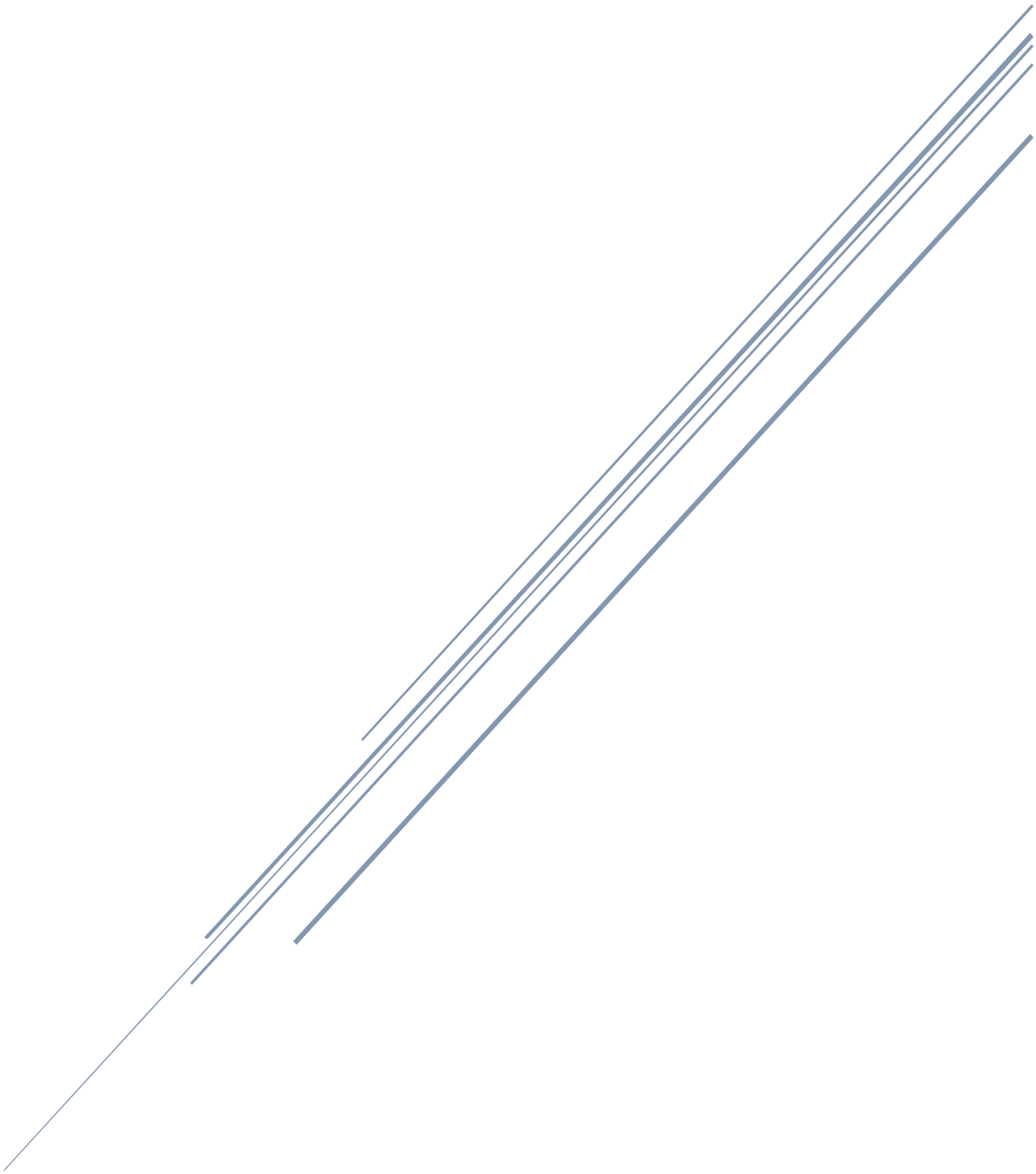
FPS GAME IN UNITY

By Rohan Kanani



University of Leicester

CO3201 Computer Science project

Declaration

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s).Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s).I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Rohan Kanani

Signed: 

Date: 03/05/24

Table of Contents

1 Abstract

This report provides an overview of the design and development of a first-person shooter (fps) game created using the unity development engine and C# programming language.

For this project, I developed a 3D open world zombie survival game which incorporates various game mechanics and secret objectives that the player can complete to either escape the game or survive to reach a higher score. The game also includes various personalisation settings such as resolution options, sensitivity slider and more to allow for different player preferences and hardware compatibilities.

The objective behind this project was to create an engaging fps game which incorporates graphics, user interactions and Ai. The background motivation behind this project stems from a personal interest in the industry combined with the knowledge, that the interest in this industry is also growing, meaning a higher demand of new releases [1].

The development process involved various steps such as design and creation of a 3D game level, collection of suitable assets and programming of various interactable elements. This is all achieved using the tools provided by unity game engine.

The main conclusion of this project shows the effectiveness of using unity to create a 3d fps game and the possibility of a full release given more time and resources. The game was evaluated based on user feedback from stakeholders and a comparison from the original requirements listed from the interim report.

2 Introduction

2.1 Aims

Through analysis and observation of the current gaming trends, I've noticed a gap in the gaming market for an immersive and challenging First Person Shooter (FPS) survival game. My motivation stems from an interest in the FPS genre, combined with a growing demand amongst gamers I've talked to about what they think is missing in the fps genre. A game like this is important as it provides entertainment and joy to players by providing a challenging and strategic experience for gamers to play and develop their skill at through repetition. Therefor the aim of this project is to create a fps survival game that fills this gap. This will be achieved through a series of well-defined objectives.

2.2 Objectives

1. Gather 3D assets from the unity asset store that align theme of the game
2. Design and construct an immersive 3D map using unity's level construction tools and assets

3. Develop and implement various game mechanics which are crucial to an fps survival game such as weapons, movement system and more(further detailed in requirements)
4. Create a user interface
    a. Design and create a user-friendly main menu that aligns with the games theme
    b. UI elements that provide the user with necessary information such as health and ammo
    c. Control information on screen
5. Testing
    a. Test and document any bugs for every game mechanic implemented
    b. Fix bugs found from tests
    c. Balancing game mechanics based on testing and user feedback

2.3 Methodology

I will use the agile methodology to develop my game as it is an iterative approach that ensures rapid delivery of small segments. It is also adaptable for adjusting requirements and allows for constant reassessing of plans. Agile is also suitable for a small team such as one person. Due to all the reasons stated therefore I believe that the agile methodology is the most suitable approach for me and this project.

3 Survey of Literature

In this section I will conduct research on all elements of my project including the game engine, game design, game ai, and game mechanics

3.1 Game engine

Before development begin, a suitable game engine must be selected as it decides the features which can be included in the game. The game engine serves as a software framework which provides a list of tools which aid in the whole development process of the whole game. For development of a fps game, I have chosen unity due to multiple key factors which include:

Graphics rendering – Unity allows for 3d modelling which can be used for creating of a 3d environment which the game can be played on. This is essential as it is the backdrop of the game and without models the game would have no visuals and have no level to be played on. One of the main advantages of Unity when it comes to 3d modelling is that it can run on multiple platforms with online and offline modes on windows, mac, and linux. Unity also provides good light rendering at a reduced cost of performance while developing and playing [2].

Physic capabilities – Unity allows for realistic simulations of movement collisions and interactions with the game map. Unity uses something called colliders. Colliders are invisible

objects which can be attached to game objects, they simulate physical properties of the material that they supposedly represent [3].

Complexity – compared to other engines unity is known as an easier to learn engine as there are countless resources teaching users how to use it. Unity also consists of other user-friendly features such as a simple, clear, and easily configurable user interface, constant updates to the platform, and a wide variety of assets which are available to use for free from the unity asset store [4].

Performance – Since the scope of my game compared to other full releases is much smaller unity is the ideal engine as it is typically used for smaller projects and by smaller development teams. According to [4] unity struggles with games which have a large and complex world, since the scope of my map isn't at this level I shouldn't have to worry about unity's Impact on performance.

Cost – unity offers a free student and hobbyist tier which can be used for personal and commercial use [5].

3.2 Unity compatibility

Since I will be developing my game using the unity engine, I must ensure I have a development environment which is compatible with the engine. The unity website states the following minimum requirements:

Minimum requirements for windows
Os: Windows 7 (SP1+), Windows 10 and Windows 11, 64-bit versions only.
Cpu: X64 architecture with SSE2 instruction set support
Gpu: DX10, DX11, and DX12-capable GPUs [6]

Since the requirements are not demanding I am able to use the engine with ease without any performance issues during development.

3.3 What makes a good game

According to a journal I found on game design [7], there are multiple key objective criteria that must be considered when creating a good game. Some criteria listed in this journal include:

- Freshness and replay ability – the game should try to be different each time it's played
- No 'king maker effect' – a game loses its appeal if the player at any stage has no chance of 'winning'
- Creative control – any game not based on chance should give the player an opportunity to affect its outcome
- Uniformity – the theme, format and graphics of the game should be consistent
- Winning chances – the player must have a theoretical winning chance

- No early elimination – the player should be always involved in the game

While development I will keep all these criteria in mind so I can ensure the game remains engaging to the players.

3.4 Game Ai

Since my game will contain enemy ai I have conducted some research on different types of ai and ai elements. [8]

One element of ai I must consider is what movement types I want to use. Movement types describe the way the enemy moves in a combat situation. According to a journal I found on enemy npc I could utilise many different types such as:

- Flanking intensive – enemy will move to attack from unexpected directions
- Passive – npc will not move while attacking
- Slow push – npc will slowly move towards the player in a straight direction
- Rush – npc will make a dash towards the player without regard to safety
- Cautious – npc is opting to move around the map and attack from a range without getting too close to the player

Although I may not be able to incorporate all these different types of movement types in my project, I will be able add an appropriate amount and combine some of these movement types with each type of ai my game will contain. Types that I don't add can also be added in future iterations of my game if wanted or needed.

Another element I must consider is attack frequency and type. Attack frequency is how often the ai will attack and deal damage to the player and type is how they will attack. Two types of attack types I have considered are melee and range. The frequency of these attacks can be randomised or set. Since I'm taking an agile approach to development, I will decide the frequency during development and testing. Further attack types may also be added in future iterations if needed as well.

3.5 Other solutions

Some existing solutions in this genre include:

- Call of Duty: Black ops 2, Treyarch Inc., 2012.
- Back4Blood, Turtle Rock Studios, 2021
- Dead Space, EA Redwood Shores, 2008.
- Half-Life 2, Valve Software, 2004.
- Killing floor 2, Tripwire Interactive, 2016
- Halo: Combat Evolved, Bungie Studios, 2001.
- Halo 3: ODST, Bungie Studios, 2009.

These games have set the industry standard for a good FPS game and through playing them I have seen a wide variety of game mechanics which have contributed to the success of these solutions. By analysing these existing solutions, I have gained knowledge on what

players value and don't value in fps games and hope to implement and improve some of these within my project.

4 Requirements

From the above research and past experiences, I have documented project requirements for my game to be functional. I have split these requirements into three different types:

- Essential – requirements which are essential for my game to work as the bare minimum.
- Recommended – The elements are highly required for a good game
- Optional – These are requirements that enhance the overall game but are not necessary

4.1 essential requirements

- The player must be able to walk forward
- The player must be able to walk backwards
- The player must be able to walk left
- The player must be able to walk right
- The player must be able rotate in every direction
- The player must have a gun
- The player must be able to shoot
- The game must have an enemy
- The enemy must be defeated after being shot by the gun
- The game must have a playable surface (map)

4.2 Recommended requirements

- The player should have health
- The enemy should be able to move
- The enemy should be able to find my player
- The enemy should be able to damage my player via collision
- The game should end when enough damage is taken to players health
- The gun should have a set amount of ammo that can be shot
- The gun should have a magazine that reloads from ammo
- The gun should only work if there is ammo in the magazine
- The enemy should respawn in sets of waves
- The map should have different terrains (levels)
- The map should have textures such as concrete, grass, etc…
- The map should have assets scattered over it such as bushes, trees, cars, etc…
- Shooting should make shooting sounds
- Walking should make walking sounds
- Running should make running sounds
- Enemy should make noises

- My player should be able to jump
- My player should be able to crouch

4.3 Optional requirements

- My player may have a stamina limit for running
- The game may have a visual representation for total ammo
- The game may have a visual representation for magazine ammo
- The game may have a visual representation for health
- The game may have a visual representation for what wave they are on
- The wave of enemy may get progressively harder
- The enemy may have animation when moving
- The game may have a start option
- The game may have a quit option
- The game may have a controls option
- The game may have a pause option
- The game may have an un-pause option
- The game may have a credits option
- The player may be able to adjust the mouse sensitivity
- The player may be able to adjust the game volume

I have also noted down some non-functional requirements. These are requirements that will describe how the system should perform and ensure good usability

4.4 Non-functional requirements

- The game should run at minimum 30fps on hardware from the last 5 years
- The game should load in less than 60 seconds
- The game should run on window 10/11 platform
- The games controls should be easy and obvious to learn and use
- The game should have a simple user interface

5 Design

In this section I will describe how I will implement my requirements into my project using text, class diagrams, flowcharts, stepwise refinement, and user interface plans.

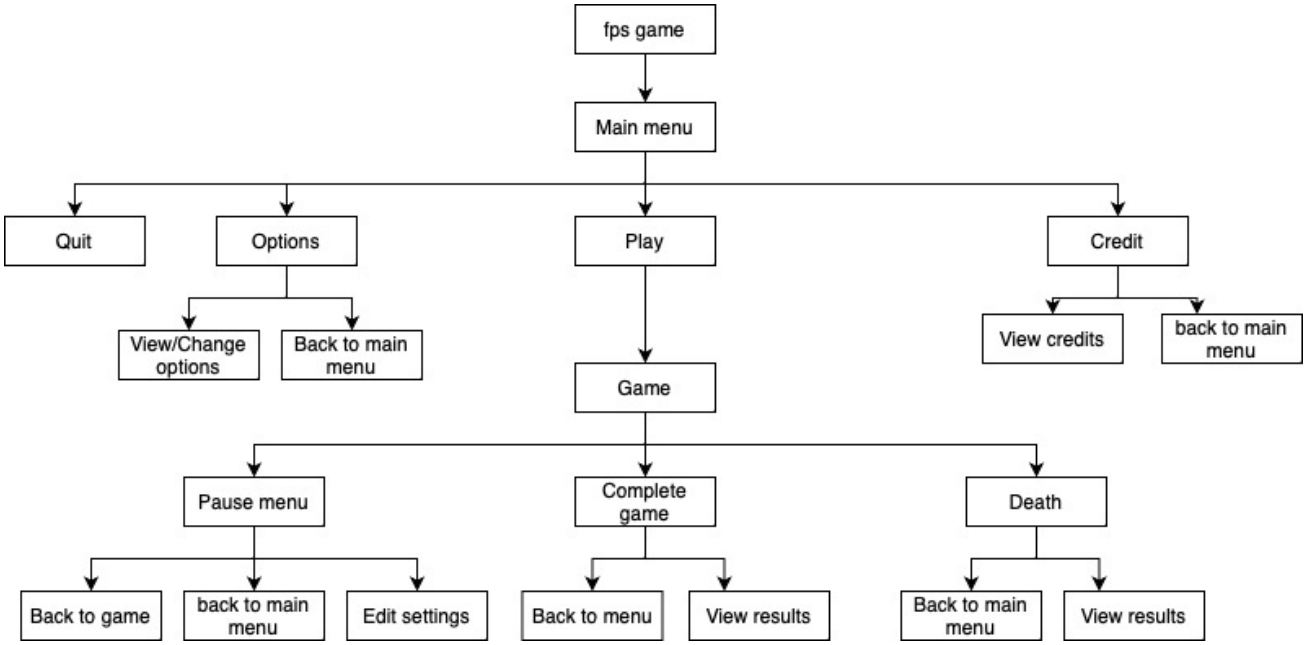
5.1 Stepwise refinement

I will break down my problem into smaller problems until they can be read and interpreted with ease. This will also help me plan out how I will tackle the problem when it comes to creating each element of my game and where I should start. I will do this by performing stepwise refinement (top-down model) with sections of my game. The models will contain a hierarchy of steps and as you go down the hierarchy each step should be easier to diagnose and solve. Due to my game having multiple large sections I will show multiple top-down models for each major section.

5.1.1 Game

Figure 1. Modular view of game shows my full game broken down with each scene that it will contain and where it leads to.

Figure 1. Modular view of game



Level 0:
Main menu – central hub of the game where everything eventually leads back to

Level 1:
Quit – exits application
Options – takes you to options menu
Play – starts the game
Creds – takes you to credit page

Level 2:
View/change options – edits options like sensitivity, volume
Back to main menu – takes back to level 0 (main menu)
View credits – displays credits for all assets used

Level 3:
Game – game level, where the player plays the game

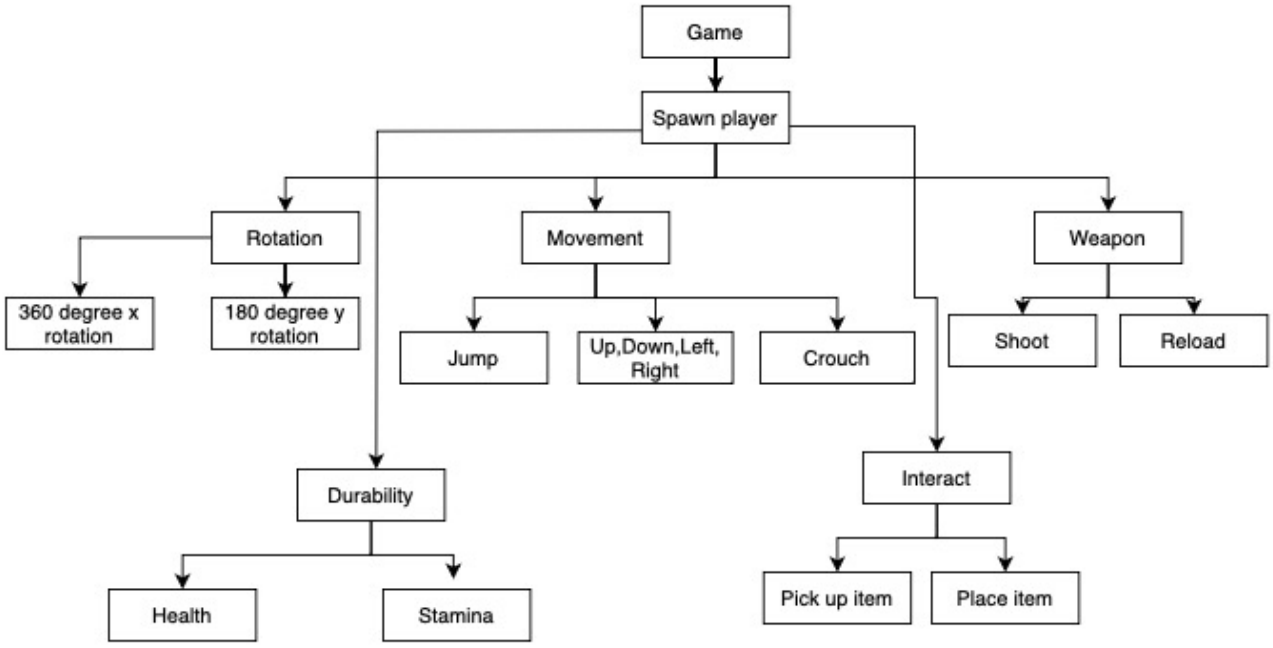
Level 4:
Pause menu – freezes game to show options
Complete game – game ends and shows options and information
Death – death screen shows information and options

Level 5:
Back to game – returns player to game
Edit settings – edit and apply settings
View results – displays results such as wave, points, items

5.1.2 Player

Figure 2 shows a top-down model of the player broken down into smaller steps

Figure 2. Modular view of player



Level 1:
Player spawns in the game

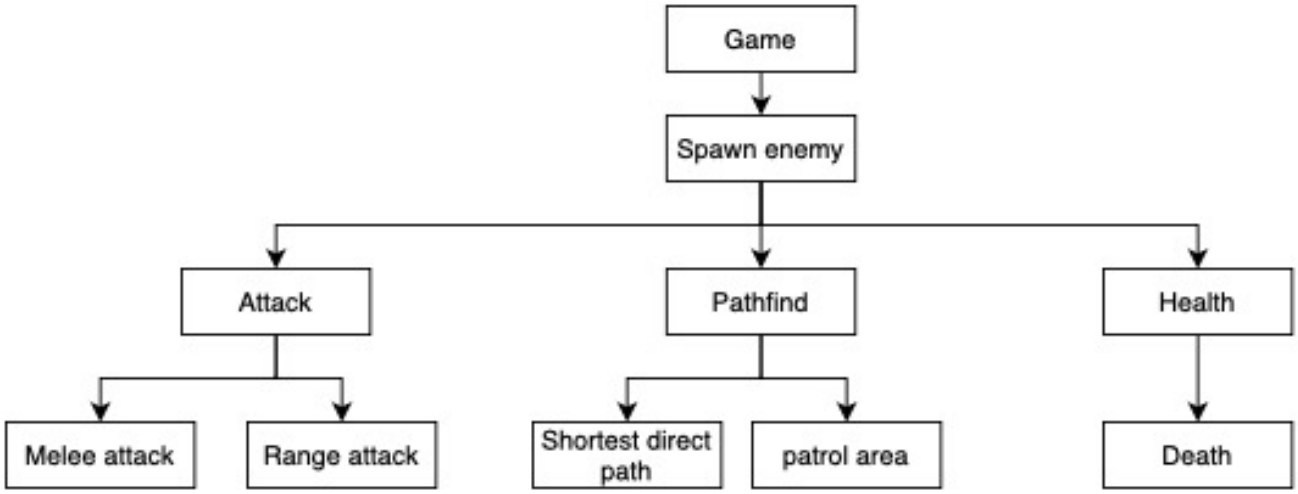
Level 2:
Rotation – player can rotate
Movement – player move on the map
Weapon – player has a means to defeat the enemy
Durability – stats which change depending on scenarios
Interact – interact with the world

Level 3:
Rotate on x axis – turn left and right
Rotate on Y axis – turn up and down
Jump – move on y axis
Up, Down, Left, right – move on x axis
Crouch – decrease size
Shoot – means of defeating enemy
Reload – replenish ammo in gun
Health – current player health
Stamina – means to sprint around, jump, and crouch
Pick up item – pick up items found on map
Place item - place items on map if needed

5.1.3 Enemy

Figure 3 shows the top-down model of the enemy ai broken down into smaller steps.

Figure 3. Modular view of enemy



Level 1:
Spawn enemy – enemy is spawned

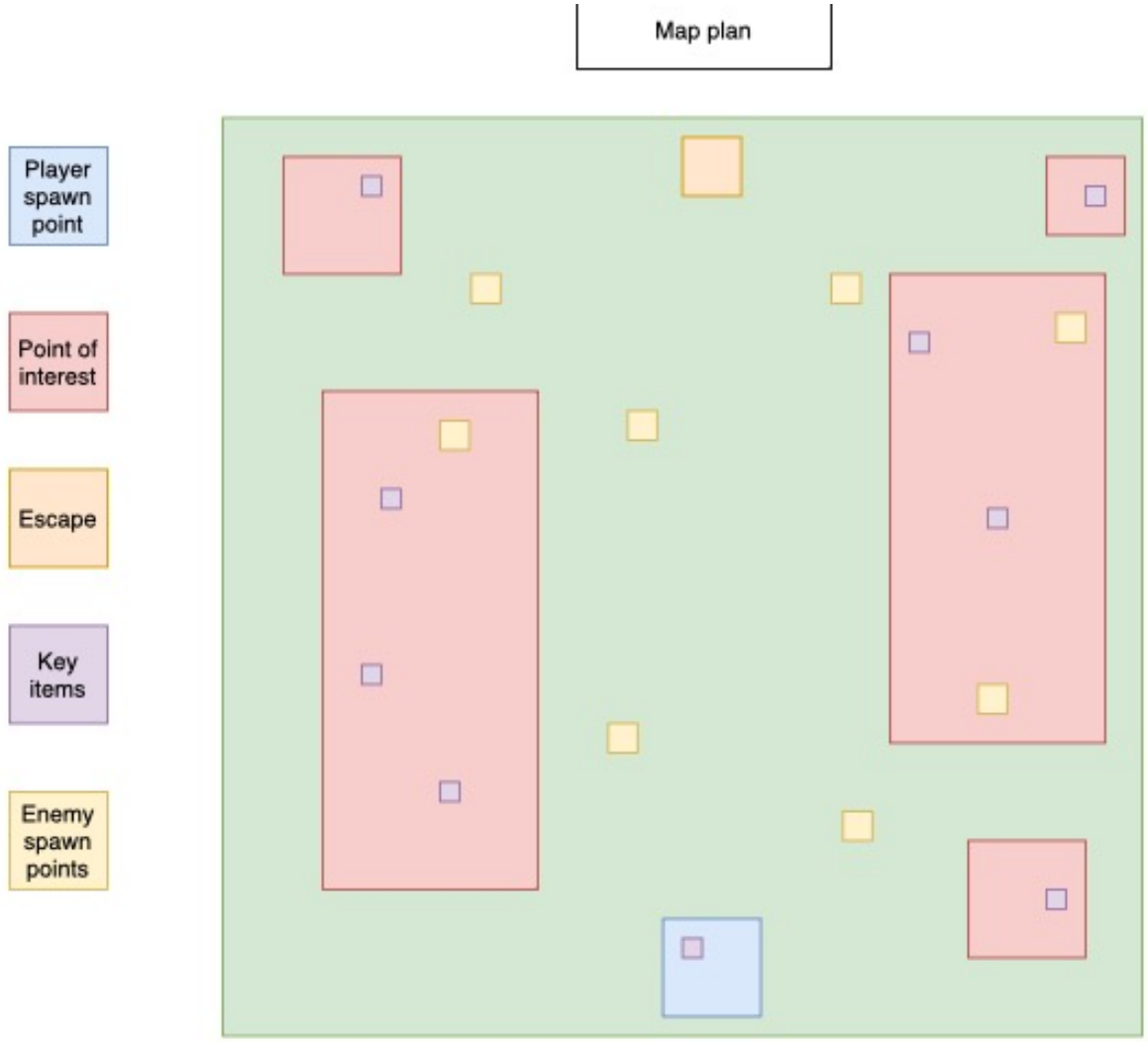
Level 2:
Attack – means to attack the player
Pathfind – track and follow the player
Health – counts health from being attacked

Level 3:
Melee attack – close quarters attack when enemy is at the player
Range attack – distance attack on player
Shortest direct path – rushes directly to player
Patrol area – patrols area and tracks player if they enter that area
Death – remove enemy from map

5.2 Level design

Since it is essential for my game to have a level for where it is played, I have made a basic top-down plan for roughly how my map will look shown in Figure 4. The abstraction performed makes it easier for me to plan out how I want my map to look without having to worry about visuals at this stage.

Figure 4. Map plan



Player spawn point – where the player will spawn when the game starts
Point of interests – buildings/areas which contain key items such as weapons and items
Escape – where the player can escape if they have fulfilled the requirements
Key items – possible locations of items
Enemy spawn points – where the enemies can spawn when the waves start

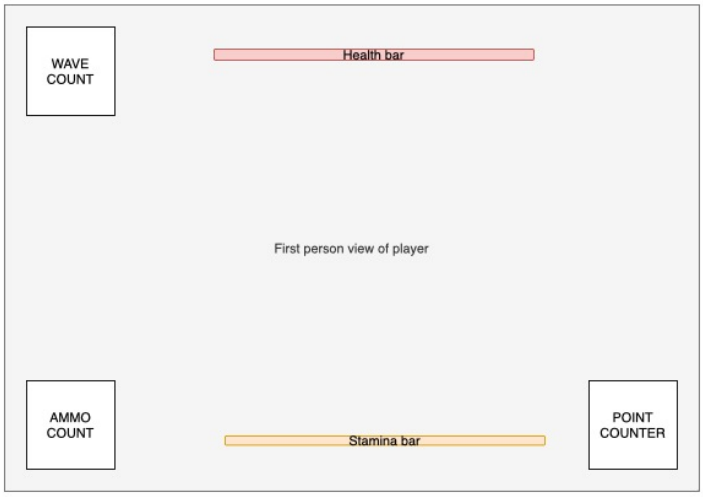
Using unity's level creation tools which allow for placement and manipulation of objects and terrains I should be able to translate this plan into a fully functional game environment.

5.3 user interface design

In this section I will plan out how my user interface designs will look for each scene of my game including the players point of view, pause menu, and main menu. I will keep the designs as simple as possible to ensure a clear and intuitive user interface experience in my project. I will stick as closely as possible to these designs when implementing them to my project. After collecting feedback from stake holders on the user interface new designs can be made and implemented to my project on any future iterations of it.

5.3.1 Player POV

Figure 5. Player view

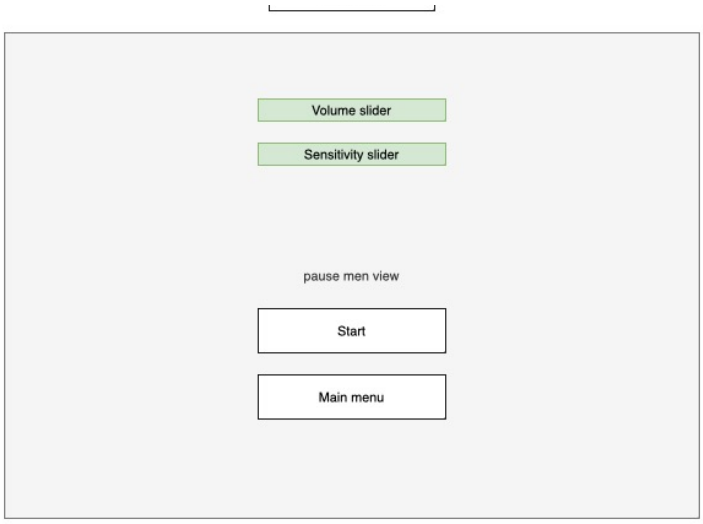


Justification:

- Health bar and stamina bar at the top and bottom of the screen. Clear representation of the players current stats.
- Wave Count tells the player the current wave
- Ammo count keeps track of the ammo in the gun (eg: 7/30)
- Point count keeps tracks of how many points the player has

5.3.2 Pause view

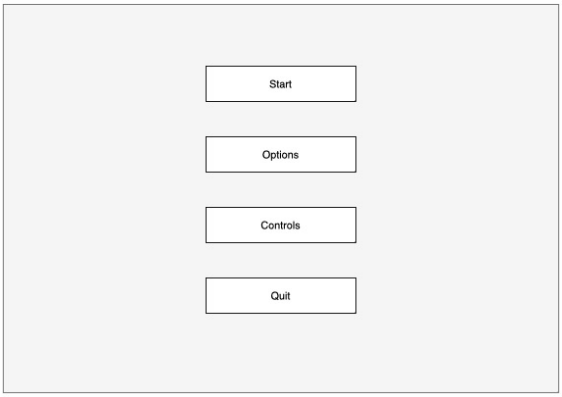
Figure 6. Pause view



- Volume and sensitivity slider available in the pause menu. These should be here as players should be able to customise these on the go to suit their preferences.
- Start button un pauses the game.
- Main menu stops the game and sends you back to main menu

5.3.3 Main menu view

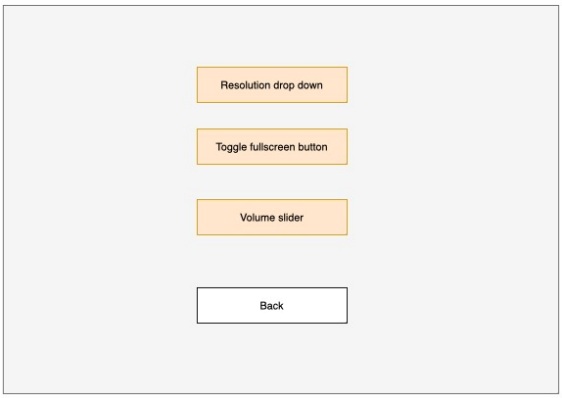
Figure 7. Main menu view



- Main menu buttons correspond to their following pages and changes the screen/scene to them when pressed.

5.3.4 Options menu

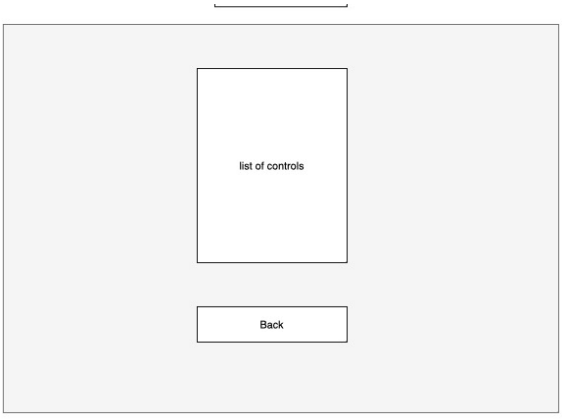
Figure 8. Options view



- Options menu contains customisable settings for the game. Other options may appear here if further requirements are added after the completion of the ones previously listed.

5.3.5 Controls menu

Figure 9. Controls menu view



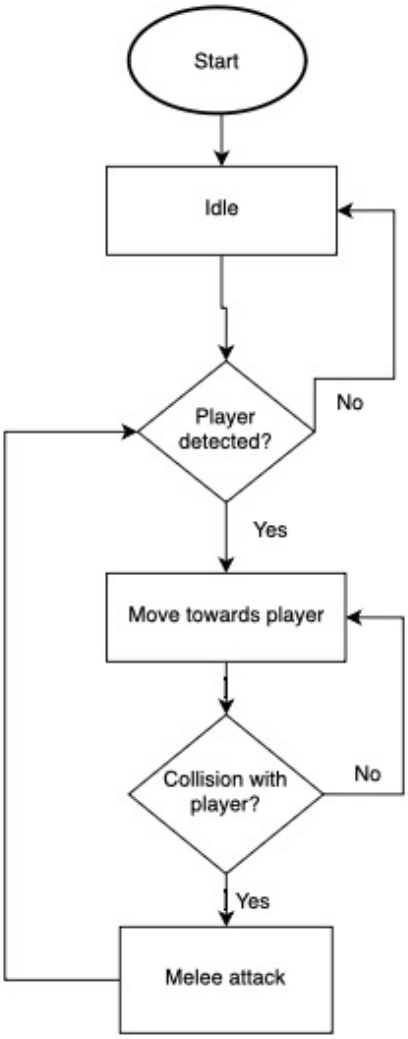
- Simple screen which just shows a list of all the controls to play the game.

5.4 Enemy ai

I have aimed to incorporate at least two different enemy ai variants into my project with different behaviours and attack patterns. Enemy 1 will be a simple rush type enemy that uses melee attacks when they collide with the player. Enemy 2 is planned to be a combination of a slow push and cautious type enemy with both melee and ranged attacks (Enemy types explained in survey of literature section page 6). I have represented both these types of enemies in a continuous flowchart which only ends once the enemy is defeated.

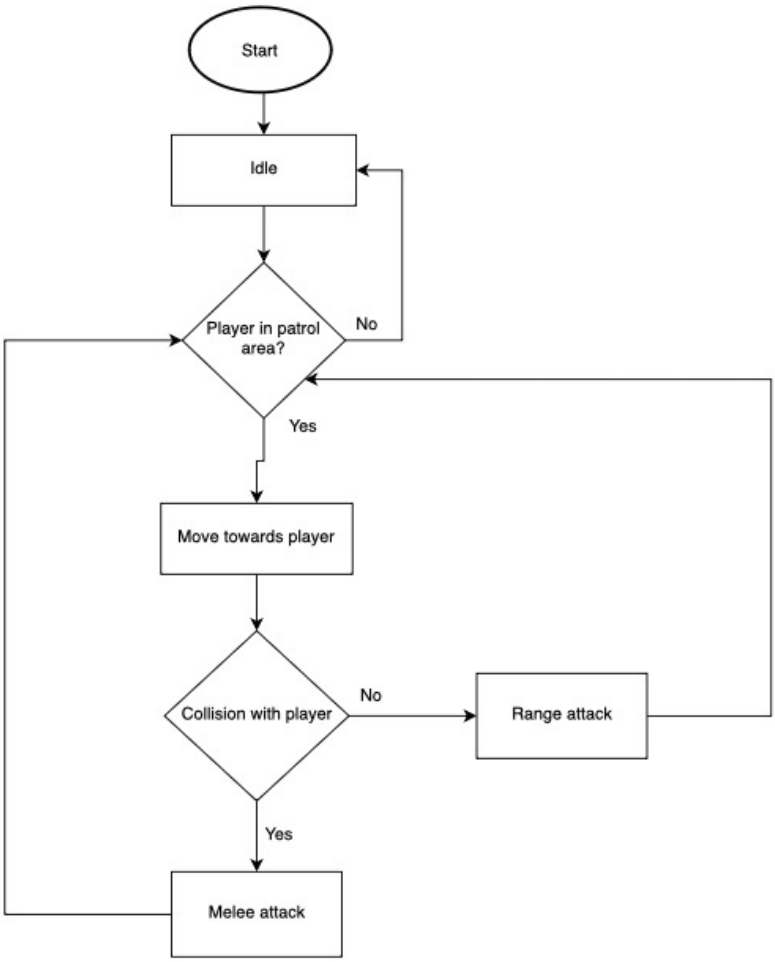
Enemy 1:

Figure 11. Enemy 1 flowchart



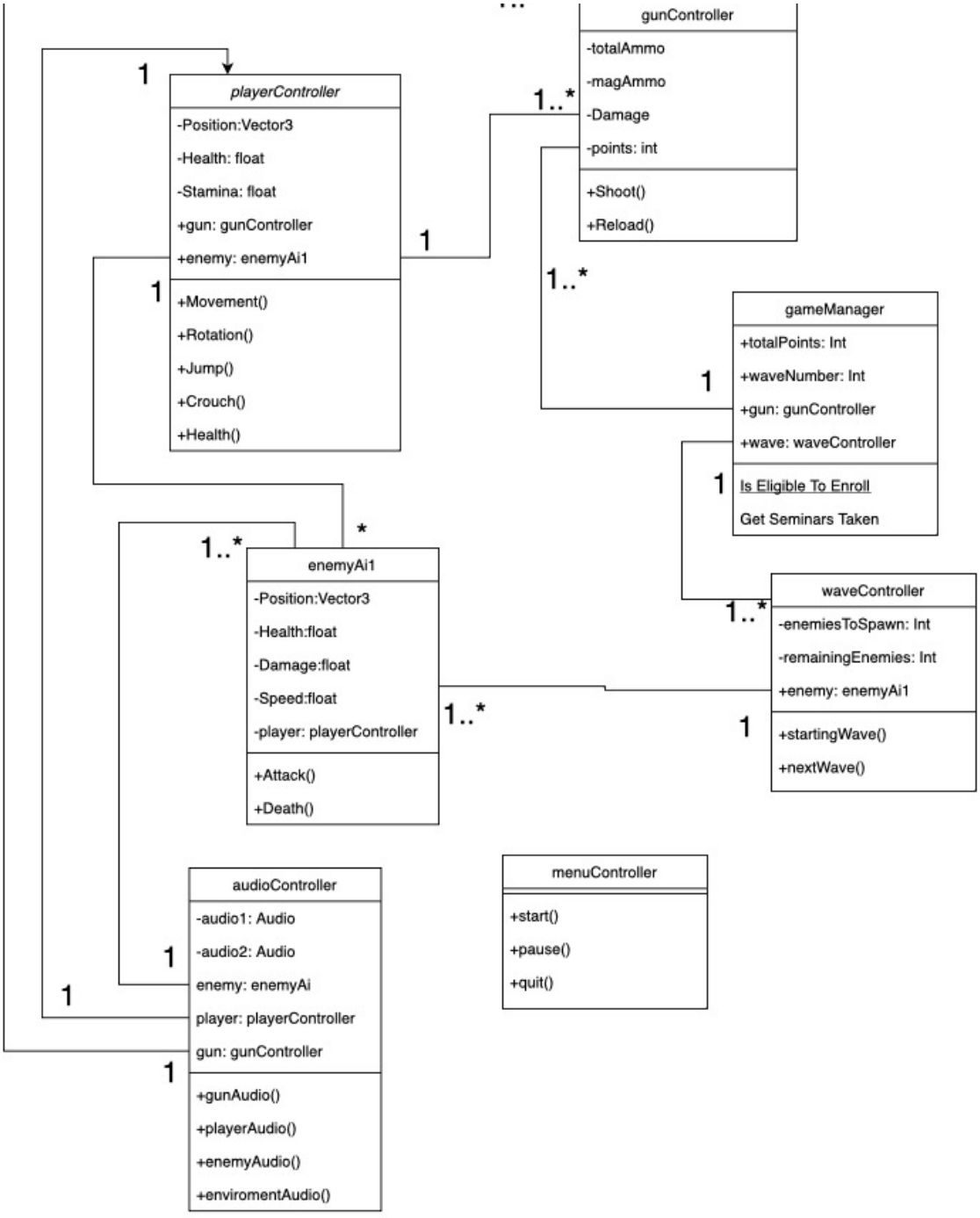
Enemy 2:

Figure 10. Enemy 2 flowchart



5.5 class diagram

Figure 12. Class diagram



The class diagram shown as Figure 12 shows a simplified overview of the major classes present in my project. Smaller and less important classes, attributes and relations have been omitted for readability.

5.5.1 Relationships

playerController and audioController has a 1 to 1 relationship. One player interacts with one audioController. This is because there is one player in my game and one audioController which handles all the audio in my game.

enemyAi1 and audioController has a many to one relationship. Many enemies each have their own audioController to handle each of their sounds.

gunController and audioController has a 1 or many to one relationship. One or many guns will have one audio controller which handles their sounds.

playerController and gunController has a 1 to 1 or many relationship. One player handles one or many guns depending on if the player uses different guns.

playerController and enemyAi1 has a 1 to 1 or many relationship. This is because 1 player can be affected by 1 or more enemies when they deal damage to the player. One player can also deal damage to one or many enemies.

enemyAi1 and waveController has a 1 or many to one relationship. One waveController interacts with at least one or many enemies when they spawn them during the wave.

waveController and gameManager has a 1 to 1 relationship. One waveController tells the gameManager what wave the game is on,

gameManager and gunController has a 1 to 1 or many relationship. One gameManager interacts with at least one or many guns depending on if multiple guns are used.

5.5.2 Class functions

playerController – handles all the methods needed for the player. This includes movement, rotation, weapon handling, and interacting with the game environment.

enemyAi1 – handles all the necessary methods for the first type of enemy ai such as its movement type, health, and attack method. Further enemy ai classes have been omitted from the class diagram as they will interact with the other classes the same way enemyAi1 class will.

gunController – Handles all the needs of the gun such as shooting, reloading, switching weapons, and adding points to point counter. Just like enemyAi1 other weapon classes have been omitted as they will interact with the other classes in the same way.

waveController – handles spawning of waves of enemyAi1 (and other enemy classes). The class also notifies other classes which are running of what wave the game is currently on.

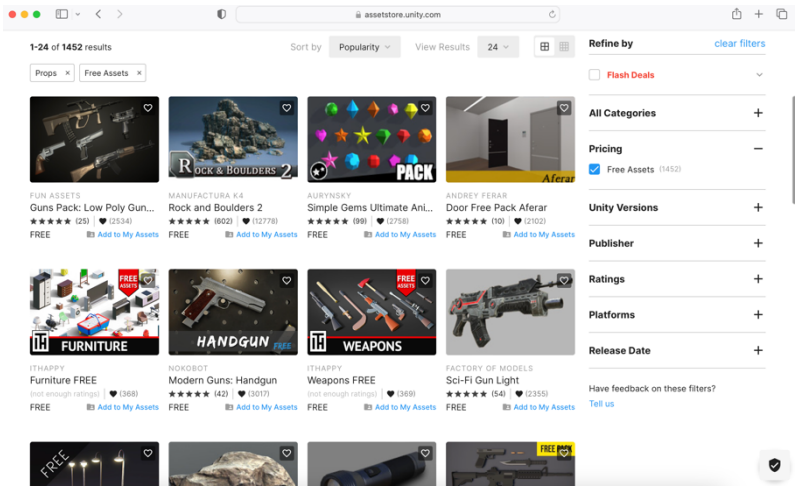
gameManager – handles the games background interactions such as counting wave numbers and how many points the player has.

audioController – handles the audio of the game and when to play them.

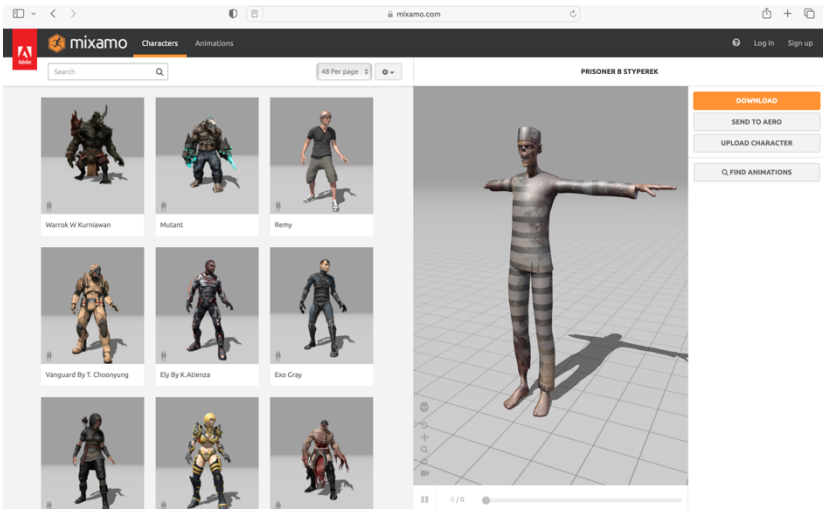
menuController – deals with switching scenes between the main menu and the actual game. Main menu has no relation between any class as it should be in a separate scene which is programmed to load other scenes such as the game itself.

5.6 Assets and animations

The development of assets and animations for every aspect of my game is substantial and due to the project time constraints, it is unrealistic to create all of them myself. Therefor I will source my assets and animations from established and reputable sources.
The main sources which I will aim to use are the Unity asset store [9] and Mixamo [10]. The unity asset store contains a large variety of free and purchasable assets for public use, specially designed for unity projects including 3d models, textures, animations, and audio. Mixamo, on the other hand is a free service which specialises in character models and animation rigging for those models. Every model and animation found on this platform is highly customisable which means they can be easily tailor them to fit my project needs.
All assets and animations sourced will be credited in a credits page in my game and in the implementation section of the report.



Unity asset store [9]



Mixamo [10]

5.7 Extra requirements

These are low priority requirements I will come up with during the implementation phase. Any work done on these extra requirements will be done after completing all previously listed essential, recommended, and optional requirements. These will be documented in the critical analysis section of the report.

6 Implementation

This section will focus on the implementation of the project.

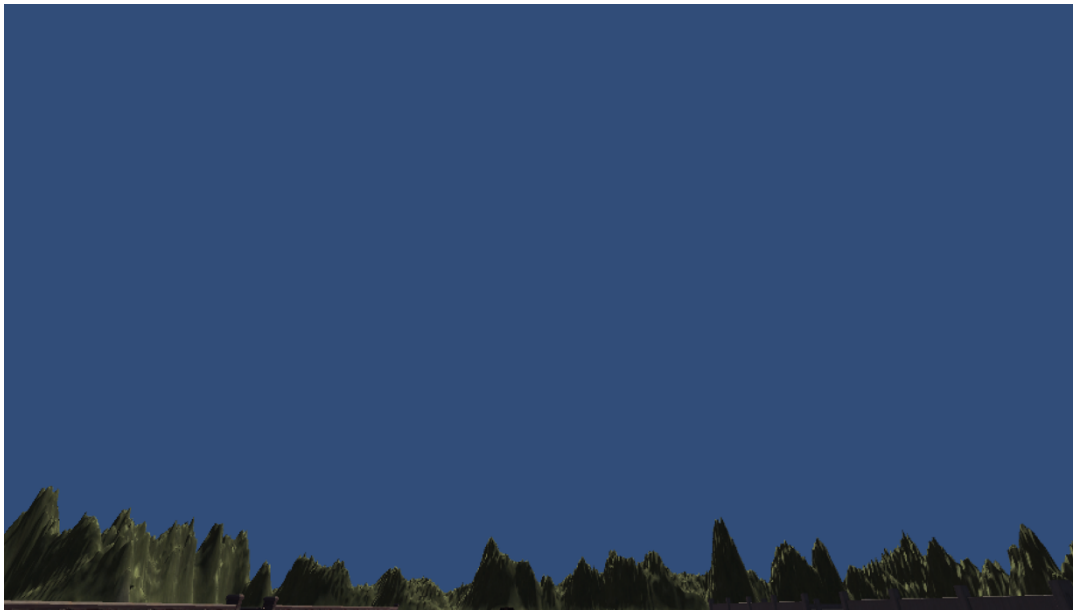
6.1 Game environment

6.1.1 Skybox

A skybox is a box of textures which resembles a sky. It is essentially a background for a 3D environment which the player can never reach [11]. By default, the sky box is blank (Figure 13) and needs a texture to be applied to it ().

Before skybox texture applied:

Figure 13. Empty skybox



After texture applied:

Figure 14. Textured skybox

The texture was found on the Unity asset store and was applied to my project via the unity lighting tools. All assets that used will be declared in a separate section of this report. This skybox not only adds a backdrop to my game it also provides lighting and shadows to my map which adds a level of immersion in a first-person perspective game.

6.1.2 Game map

After applying the sky box, I used unity's terrain tools to create basic plain map with a grass texture painted all over it. I also added mountains around the perimeter which act as barrier to prevent the player from walking off the map. I also added a house to act as a spawn point for the player when the game starts. This map was left blank for most of the development to allow me to focus on higher priority requirements as at minimum it was essential for me to have a playable surface for my game.

Figure 16. Early top-down view of map:

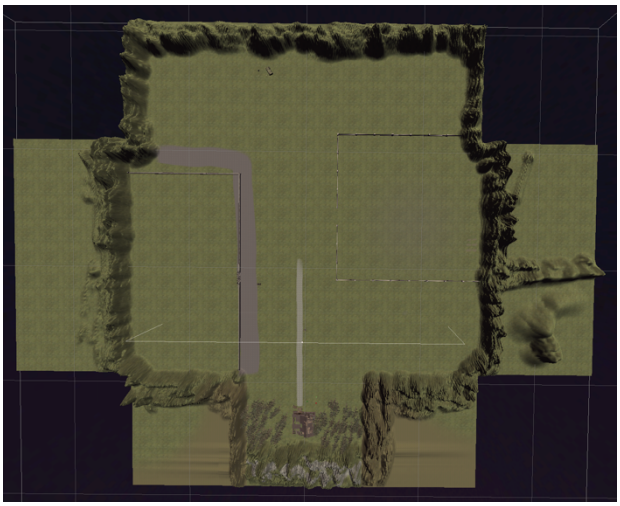
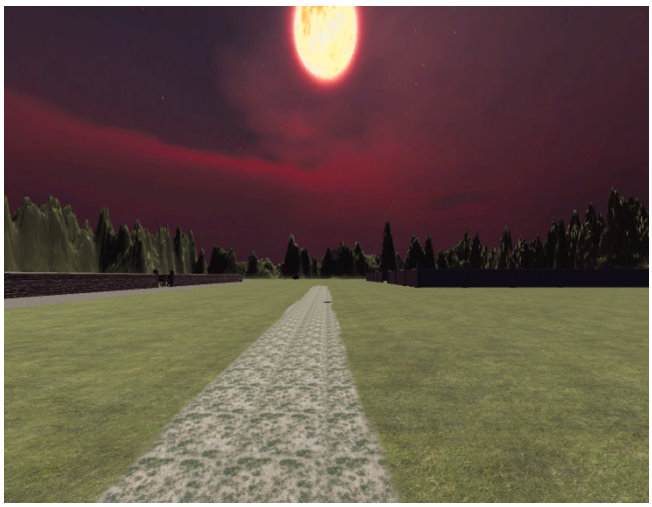


Figure 15. First person view of map



Map plan (for reference)

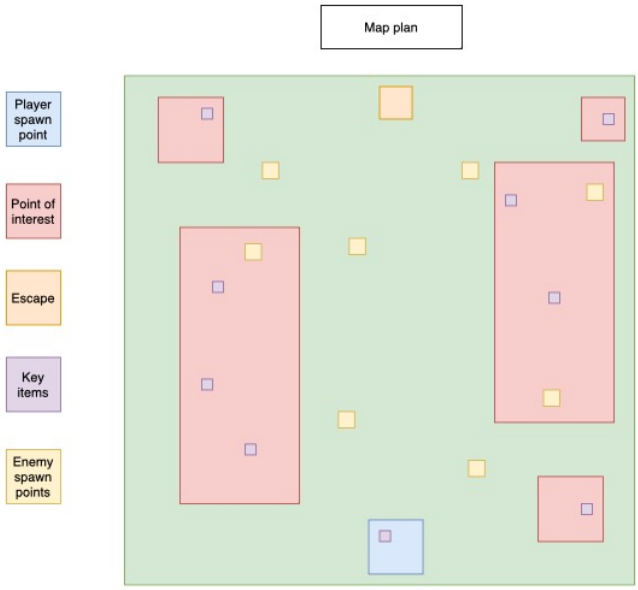


Figure 17. Final map view

Comparing my initial plan to my final implementation of my map, I have maintained mostly the same layout with slight differences in sizing of each area. While the initial plan highlighted spawn points on the map, in the final version these are still present but have been made invisible to prevent the player knowing where the enemies are going to be spawning from. Key items are also present on the final map but are made hidden to add a level of discovery to the game however, I have placed streetlights which give off actual lighting effects to give the player hints to which areas they should go to, to find any key items.

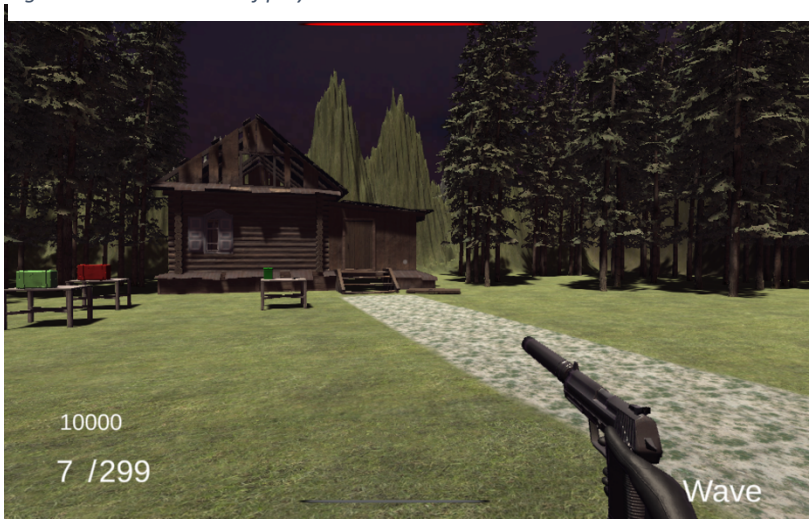
6.2 Player

This section will go over the various game mechanics implemented in my project such as movement, rotation, shooting, and more.

Since my game is first person there is no reason to incorporate an actual player model with animations since the user would not be able to see them. This means that I was just able to use a basic game object such as a cylinder to represent the player. A camera which is a device that allows the player to view the game world is then attached to the cylinder to create the first-person perspective of the game. This is shown in Figure 18Figure 18. The camera view is shown in Figure 19.

Figure 18. Scene view of player

Figure 19. Camera view of player



6.2.1 Movement

Methods of movement included in my system include walking, running, jumping, and crouching. Walking is limitless essential method of movement which the player can do to traverse the environment at a steady pace without needing to use up any stamina. This was the first step of implementing movement within my game as it is the most basic requirement for navigation. The player can walk forward, left, backwards, and right using the industry standard w, a, s, d keyboard keys respectively. Running is also present which is a limited fast burst of movement which uses the players stamina until the player stops running or is depleted. The player can run by holding left shift in combination with any walk key. Jumping in my game is activated by pressing the space bar. Jumping allows the player to get to higher places on the map such as through windows or on top of other game objects such as cars or bushes. next, the player can crouch by holding c. This makes the player small and allows them to fit into small places on the map such as broken fences or through small windows. The final element of movement present in my game is rotation. This is essential as without it the player can only move in four directions but with the addition of this requirement it allows the player to in any 360-degree angle by rotating to the angle you want to move to and then using any movement keys. Rotating is simply done by moving the mouse to the direction you want to go to. Combining all these methods of movement together allows for fluid movement around the game environment.

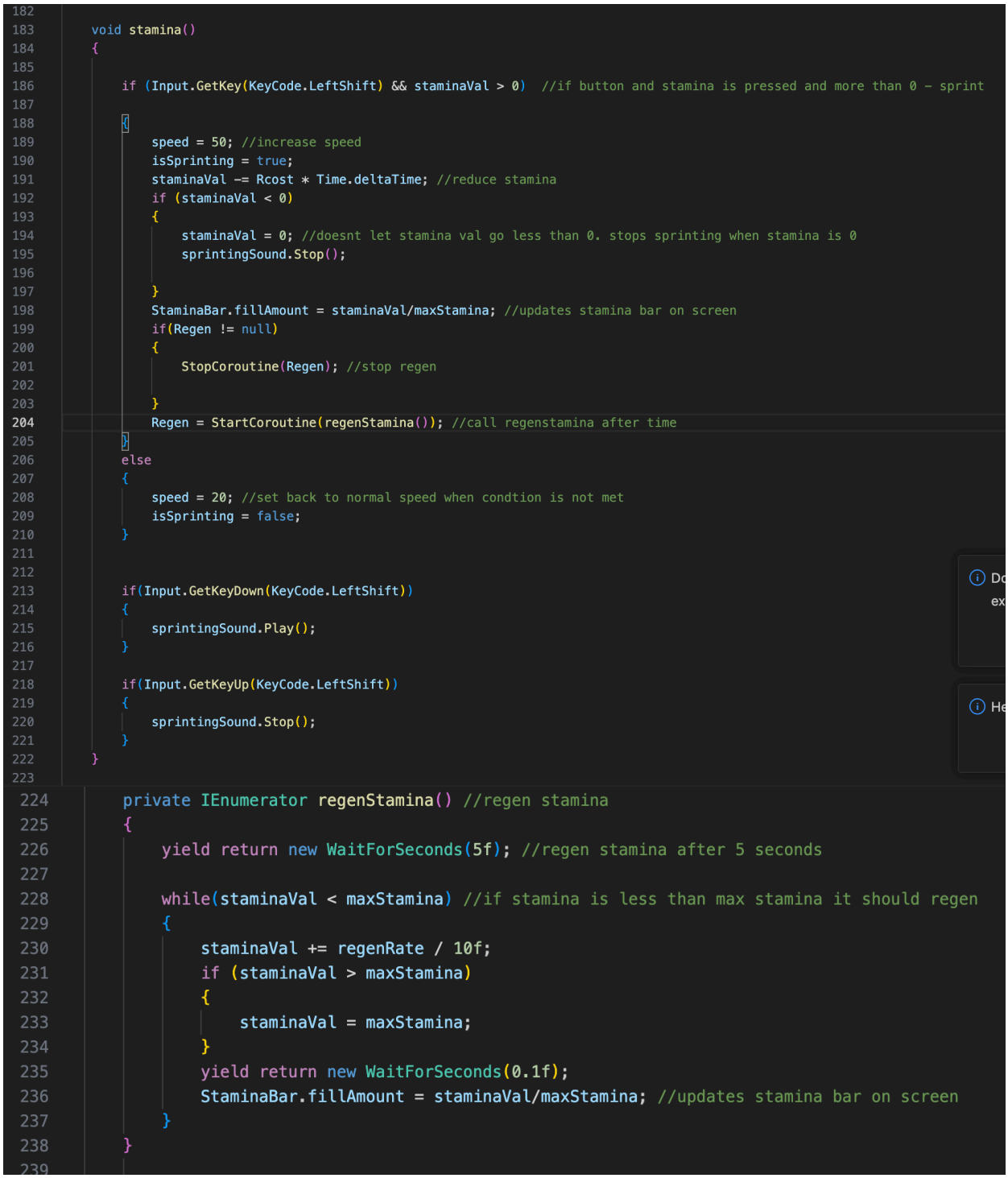
6.2.2 Stamina

As mentioned, running in my game consumes the players stamina. This is done to prevent them from being able to endlessly run forever and encourages strategic use of their stamina resources before they run out. Stamina is regained slowly after the player stops running. The rate of regeneration is set and should be tweaked in any future iterations based on user feedback to adjust to a rate that is liked by users. Stamina in my game is tracked via a stamina bar (Figure 20) presented at the bottom of the player's screen. This bar is dynamic and updates every frame while stamina is being used (Figure 21).

Figure 20. Stamina bar full

Figure 21. Stamina bar after running for some time

Figure 22. Stamina and stamina regen code part of movementController class



```
182
183     void stamina()
184     {
185
186         if (Input.GetKey(KeyCode.LeftShift) && staminaVal > 0)  //if button and stamina is pressed and more than 0 - sprint
187
188         {
189             speed = 50; //increase speed
190             isSprinting = true;
191             staminaVal -= Rcost * Time.deltaTime; //reduce stamina
192             if (staminaVal < 0)
193             {
194                 staminaVal = 0; //doesnt let stamina val go less than 0. stops sprinting when stamina is 0
195                 sprintingSound.Stop();
196
197             }
198             StaminaBar.fillAmount = staminaVal/maxStamina; //updates stamina bar on screen
199             if(Regen != null)
200             {
201                 StopCoroutine(Regen); //stop regen
202
203             }
204             Regen = StartCoroutine(regenStamina()); //call regenstamina after time
205         }
206         else
207         {
208             speed = 20; //set back to normal speed when condtion is not met
209             isSprinting = false;
210         }
211
212
213         if(Input.GetKeyDown(KeyCode.LeftShift))
214         {
215             sprintingSound.Play();
216         }
217
218         if(Input.GetKeyUp(KeyCode.LeftShift))
219         {
220             sprintingSound.Stop();
221         }
222     }
223
224     private IEnumerator regenStamina() //regen stamina
225     {
226         yield return new WaitForSeconds(5f); //regen stamina after 5 seconds
227
228         while(staminaVal < maxStamina) //if stamina is less than max stamina it should regen
229         {
230             staminaVal += regenRate / 10f;
231             if (staminaVal > maxStamina)
232             {
233                 staminaVal = maxStamina;
234             }
235             yield return new WaitForSeconds(0.1f);
236             StaminaBar.fillAmount = staminaVal/maxStamina; //updates stamina bar on screen
237         }
238     }
239
```

Figure 22 shows my implementation of a stamina system outlining how stamina is consumed and regenerated. It includes the function stamina that handles using stamina when sprinting, which is triggered by holding left shift and having a stamina value of over zero. When the player stops sprinting the regen stamina method is then called (line 207). The function regenStamina (line 224) is a special kind of

method “IEumerator” in C# which allows you to pause the execution of code. This is necessary to prevent stamina regeneration to happen instantly when the player stops running. Line 226 shows that the method waits 5 seconds before the rest of the method is executed. This number can be altered to fit player feedback if the regen speed needs to be tweaked.

Each class created in unity automatically contains an “update” method. This method is a default method which gets called during every frame of the game. All persistent game mechanics which need to be active at all times should be called within this method including all movement mechanics such as sprinting. Since stamina first checks if left shift is being held it prevents the rest of the method from being called every frame as it is checking if it is being pressed every frame and then finally executes the remaining code once it detects that it is being held.

6.2.3 Health system

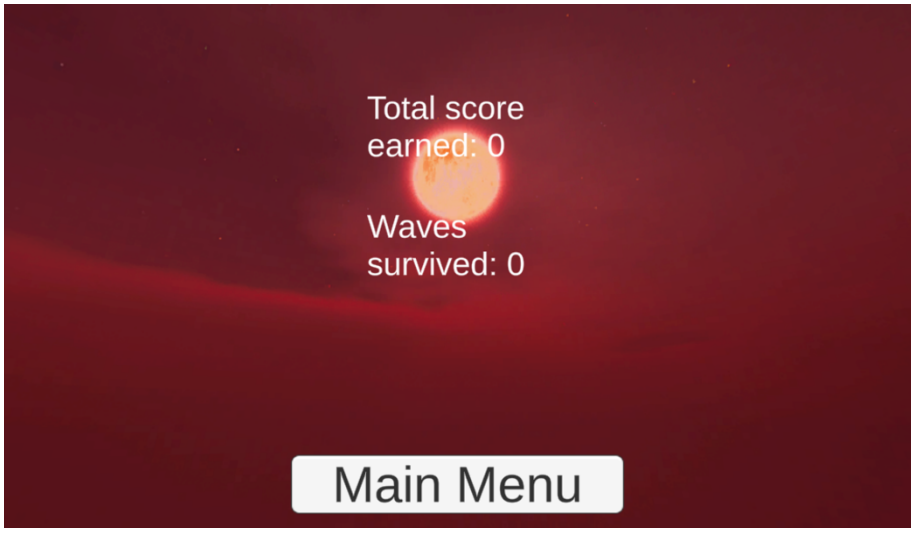
Similarly, to stamina the players health is tracked using a health bar at the top of the players’ screen shown in Figure 23. The health bar decreases every time the player takes damage (Figure 24) until the player eventually dies. The players health just like stamina also automatically regenerates overtime after being hit back to its maximum amount.

Figure 23. Player health bar full

Figure 24. Health bar after taking points of damage

One the health hits zero and the player dies the game ends and sends the player to a death screen which is shown below as Figure 25. Currently the death screen shows number of waves survived by the player and the total score they have earned during the game. The screen also includes a button which takes the player back to the main menu.

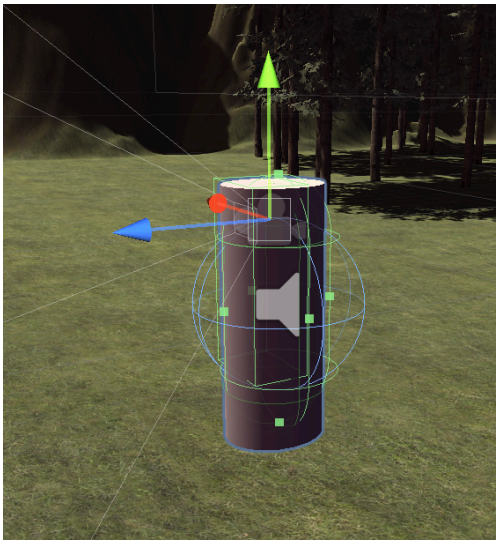
Figure 25. Death screen



6.2.4 Player audio

Audio sources in unity work by adding an audio component to a game object (Figure 26). Each audio component can hold one audio clip. To play multiple audio clips you can add multiple audio components to the same game object. The audio in game will then originate from where the components are located giving the sense of special audio effects.

Figure 26. Audio source attached to player



I have recorded several clips to use within my game. The player has a total of eight randomised audio clips that play when specific actions take place. There are four random pain sounds that play when the player takes damage, three different death sounds that can play when the player dies and a looping sprinting sound that plays while the player is running. I recorded multiple sounds for each scenario to avoid repetitiveness in the game's audio.

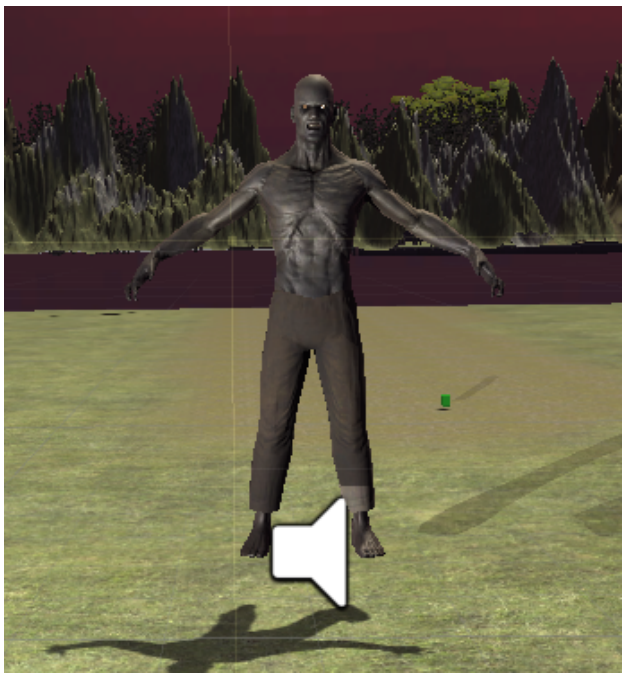
6.3 Enemy

I have implemented three different enemies within my game.

6.3.1 Enemy one

Figure 27 shows the first enemy I implemented. This enemy is quick rush type enemy which navigates to the player quickly and deals low damage.

Figure 27. Enemy 1



So that the enemies move and attack fluidly they need different animation states for every action they take. Unity provides an animator tool called a blend tree which allows you to set animation states and smoothly transition between them based on parameters in the code such as Boolean values. Figure 28 shows the blend tree for Enemy 1 and Enemy 2.

Figure 28. Enemy 1 and Enemy 2 animation blend tree

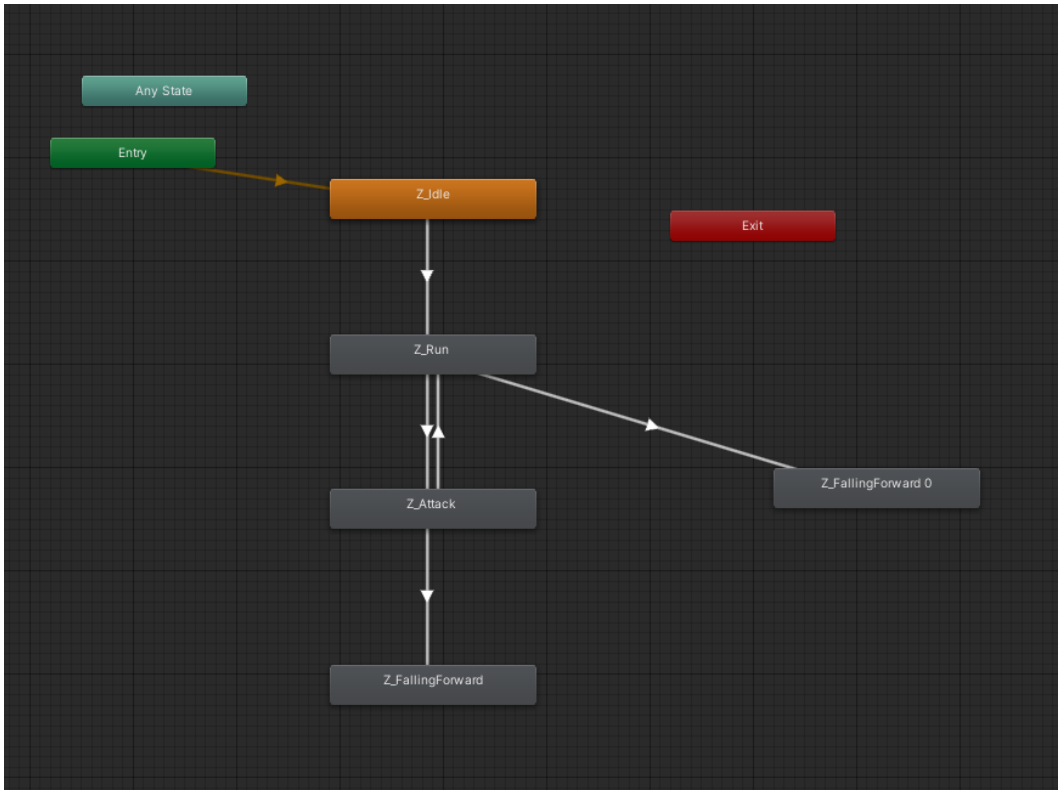


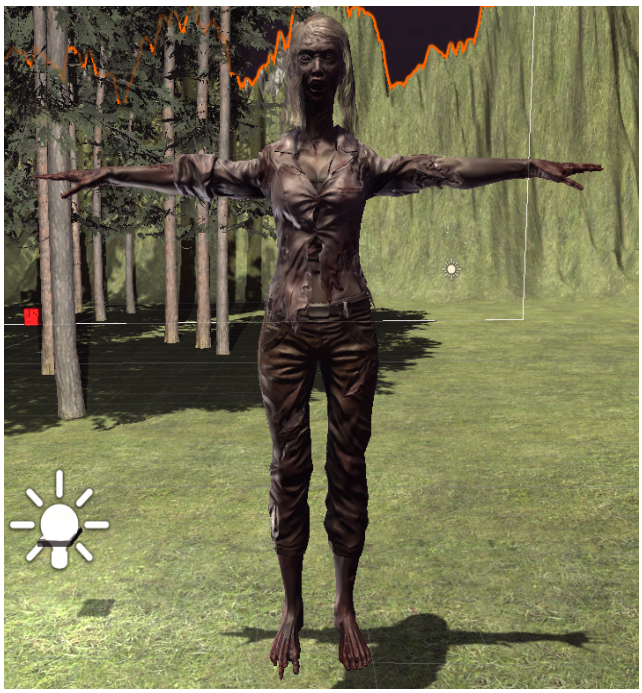
Figure 28 animation states and branches

- Entry to idle – entry is the default state. It transitions to the first animation state in the tree. In this case it is the idle animation state. This is simply just an idling animation for the enemy.
- Idle to run – the condition to move from idling to the moving animation is if isRunning == true. This means that the Enemy running animation will start playing if the animator detects if this condition is true in the enemy ai class.
- Z_Run to Z_FallingForward – the condition to move between these states happens if isAlive == false. The enemy ai class tracks if the enemy is defeated while running and if they are this animation triggers.
- Z_Run to Z_Attack – the condition to move between these states is if isAttacking == true. This stops the running animation and plays an attacking animation if it detects that damage is being dealt to the player.
- Z_Attack to Z_Run – Once the enemy stops attacking and isAttacking == false the animator switching back to the running animation.
- Z_Attack to Z_FallingForward – If while attacking the enemy dies it will stop the attacking animation and switch to the death animation.

6.3.2 Enemy two

The second enemy I implemented is shown in Figure 29. This enemy is a slow push type. This enemy is slower and deals much more damage than compared to enemy one. This enemy also has its own animations which differ from enemy one's animations.

Figure 29. Enemy 2



The blend tree for enemy two is the same as enemy ones but with different animations of the same type.

6.3.3 Boss enemy

The third enemy I add is a boss enemy. This enemy is a combination of a passive and slow push type. This enemy patrols an area and uses a combination of range and melee attacks on the player. Once the player leaves the patrol area the boss enemy goes back to its starting position and starts idling. I also gave this enemy a health bar which works similarly to the players health bar to show that it's an uncommon boss enemy. This enemy is shown in Figure 30 below.

Figure 30. Boss enemy

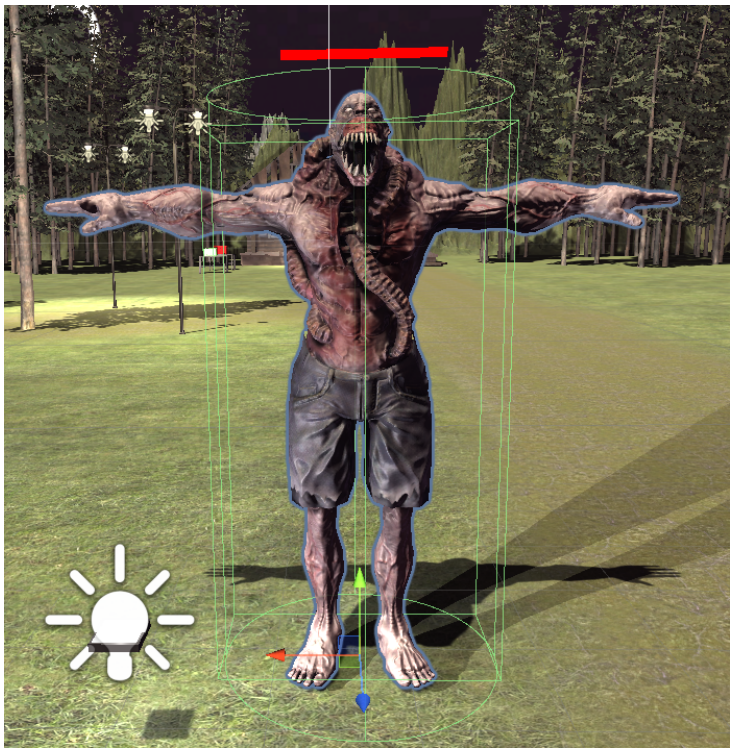


Figure 31. Boss enemy blend tree

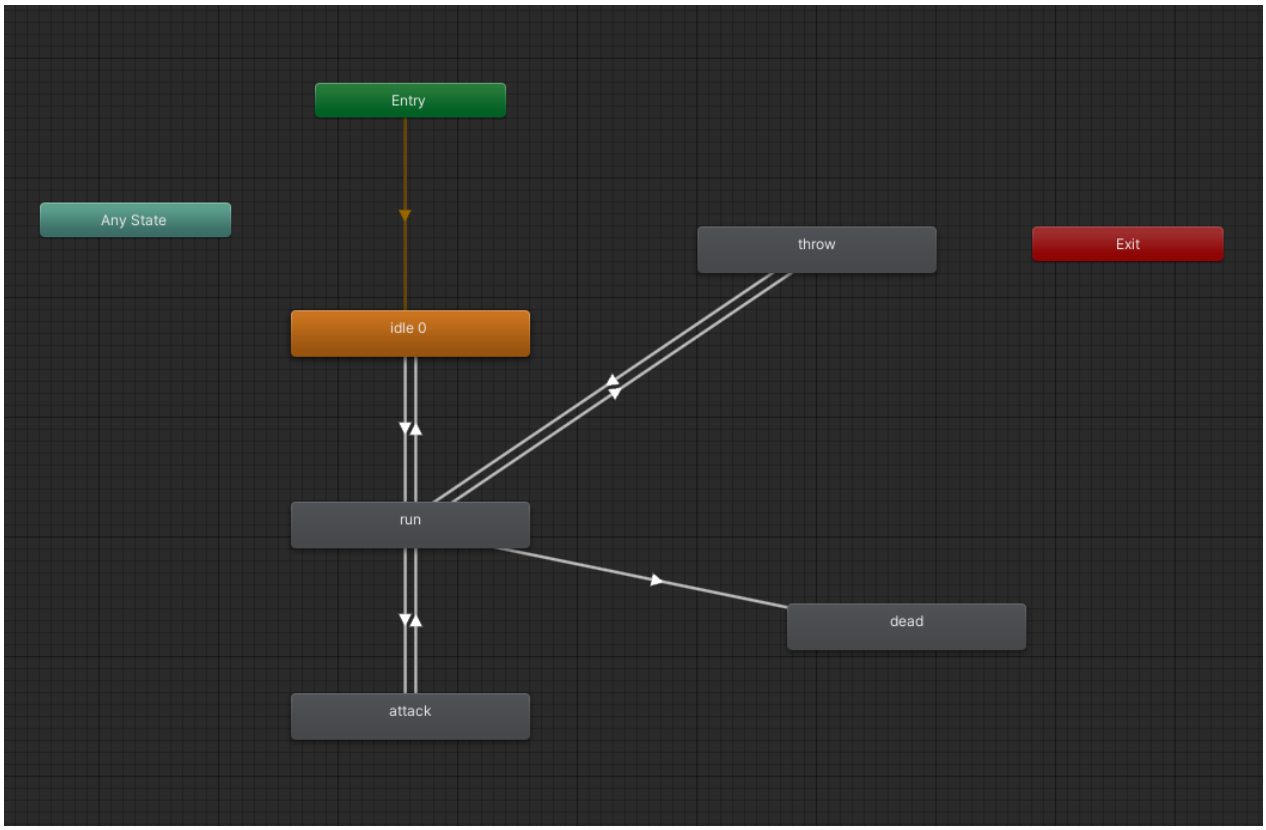


Figure 31 animation states and branches

- Run to throw – triggers throw animation when range attack is initiated
- Throw to Run – triggers running animation when range attack ends
- Run to dead – triggers death during running animation if enemy dies while running
- Run to attack – stops running then plays melee attack animation
- Attack to run – starts running when melee attack is done

6.3.4 Ai navigation

Unity provides an ai navigation package which is downloadable within the engine which contains tools for ai pathfinding [12]. The two tools I have used from this package are nav mesh surface and nav mesh agents. The nav mesh surface is a component that you apply to the game object that you use as a surface to define walkable and non-walkable area. Walkable areas for ai are highlighted blue in the scene view to show walkable areas. The nav mesh surface for my game is shown in Figure 33. Nav mesh agent components are then applied to the enemy game objects. This component then lets you alter various parameters for ai navigation such as speed and stopping distance which is shown in Figure 32. I can then reference these components in my enemy classes and instruct them to path find to my player.

Figure 33. Nav mesh surface

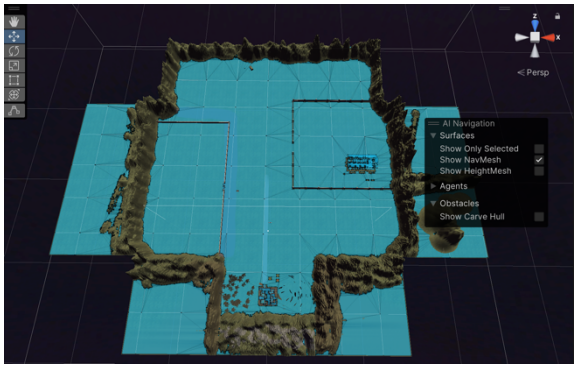


Figure 32. Nav mesh agent on enemy

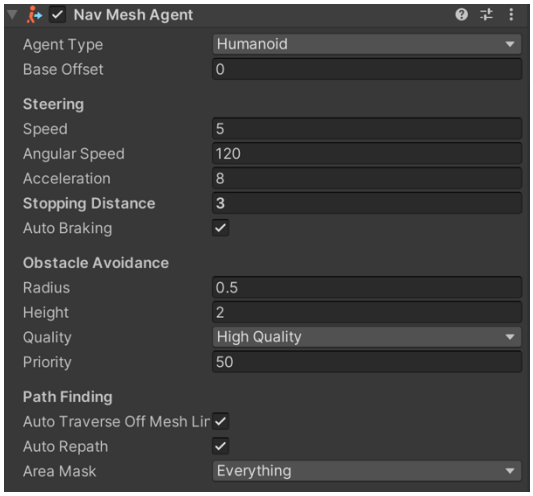


Figure 34. Pathfinding in action

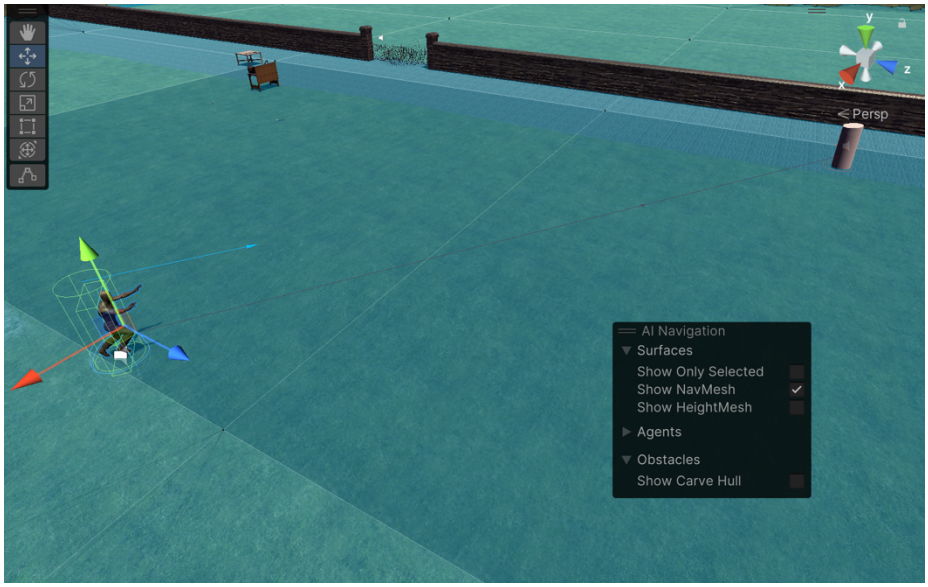


Figure 35. Commented snippet from enemyAi class showing navigation

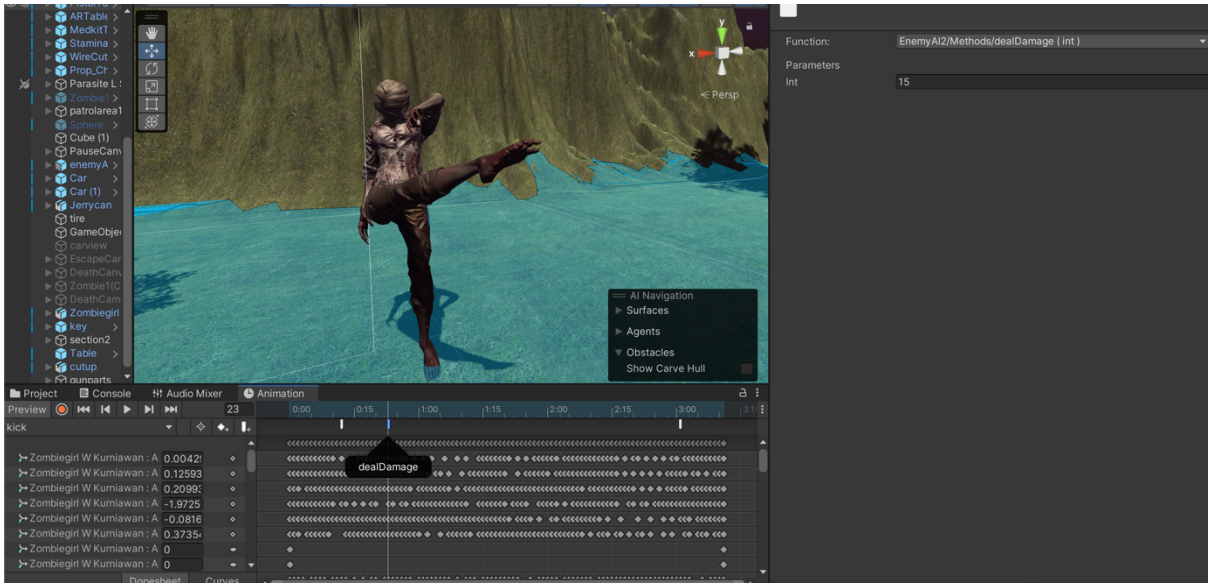
```
43
44      public void find()
45      {
46          enemy.SetDestination(player.position); //set destination to player
47          animator.SetBool("isRunning", true); // Set isRunning to true in animator
48
49          //check distance to the player
50          float distanceToPlayer = Vector3.Distance(transform.position, player.position);
51
52          if (distanceToPlayer <= attackRange && enemy.remainingDistance <= enemy.stoppingDistance)
53          {
54              //enemy is close enough to attack
55              animator.SetBool("isRunning", false); //stop running
56              animator.SetBool("isAttacking", true); //start attacking
57
58          }
59          else
60          {
61              //player is not within attack range
62
63              animator.SetBool("isRunning", true); //start running
64              animator.SetBool("isAttacking", false); //stop attacking
65              alreadyAttacked = false;
66
67          }
68
69
70
71      }
```

Figure 35 shows an example of ai navigation and controlling the blend tree within my enemyAi class. Line 46 is setting the enemies destination to where the player is at. The method itself is being called in the update method meaning that this position is being updated during every frame of the game. This is necessary as the players position need to be updated every time they move, which in an fps would be constantly. Line 47 is an example of setting a condition in the enemies blend tree. The first state in blend tree in Figure 28 is “idle”. It Then moves to “Run” if a condition is met. Line 47 shows this condition being met as “isRunning” is being set to true. Line 55 shows this condition being switched back to false when the enemy needs to stop moving. Figure 35 also shows a snippet of how attacking is implemented. Line 52 checks if the player is within the enemies attack range and if so plays the relevant animations needed.

6.3.5 Attacks

My enemies have a total of four attacks and animations that play. Each of the three enemies have their own melee attack and the boss enemy has the addition of a range attack. As mentions, Figure 35 shows an example of how the animations and attacking are programmed however dealing damage is missing. Damage is dealt in its own method which simply just minuses from the players health variable. This public method is then called within the animator at a specific moment during the animation. This is to ensure that the damage is only dealt after the physical attack animation plays and not when the whole animation ends. This is shown in Figure 36. I have also implemented attack noises which play alongside attack animations.

Figure 36. dealDamage method being called in animator



6.4 Weapons

I implemented three different guns within my game. A pistol which the player starts with, an assault rifle (AR) that the player can buy with in game points and a buildable sniper rife that the player can get if they find all its parts and build it at the work bench. Each weapon contains its own ammo and reload system which is separate from each other.

6.4.1 Pistol

The pistol, shown in Figure 37, is found in the house the player spawns in. It is located on a table and is available for the player to pick up (Figure 38). The player is unable to leave the spawn house unless they do this. Once they pick it up the door falls to the ground and allows the player to leave the spawn house and more around the map freely. This also triggers the first wave to begin. The pistol itself is a medium damage, and low fire rate gun.

Figure 37. Pistol

Figure 38. Pistol on table ready to pick up

6.4.2 Assault rifle (AR)

The AR in my game (Figure 40) is an automatic gun that has a high fire rate but low damage per hit. The player can obtain this weapon by finding it on the map and purchasing it with points (Figure 39).

Figure 40. AR

Figure 39. Buy AR

6.4.3 Buildable weapon

The final weapon implemented in my game is a buildable sniper rifle (Figure 41). The player can obtain this weapon by finding three different weapon parts around the map and then building it at a work bench. Figure 42 shows the buildable weapon parts together but in the iteration of the game given to players they will be scattered around the map. Figure 44 and Figure 43 respectively show the weapon on the work bench before and after being built.

Figure 41. Buildable weapon

Figure 42. Weapon parts together

s

Figure 44. Weapon before being built

Figure 43. Weapon after being built

Figure 45.Gun3PartsController script

```
28      public void OnTriggerStay(Collider other) //check if collison with parts
29      {
30          if(other.gameObject.tag == "gunscope") //collison with part
31          {
32              pickUpText.SetActive(true); //enable pick up text
33
34              if (Input.GetKey(KeyCode.E)) //pick up part
35              {
36                  hasPart1 = true; //tracks that part is collected
37                  hasparts(); //runs has parts
38                  pickUpText.SetActive(false); //disables text after pick up
39                  part1.SetActive(false); //disables game object after pick up
40              }
41
42          }
43
44             if(other.gameObject.tag == "gunmag") //same as above but with other part
45          {
46              pickUpText.SetActive(true);
47
48              if (Input.GetKey(KeyCode.E))
49              {
50                  hasPart2 = true;
51                  hasparts();
52                  pickUpText.SetActive(false);
53                  part2.SetActive(false);
54              }
55          }
56
57             if(other.gameObject.tag == "gunbarrel") //same as above but with other part
58          {
59              pickUpText.SetActive(true);
60
61              if (Input.GetKey(KeyCode.E))
62              {
63                  hasPart3 = true;
64                  hasparts();
65                  pickUpText.SetActive(false);
66                  part3.SetActive(false);
67              }
68          }
69      }
70
71
72        public void OnTriggerExit(Collider other) //disables all text afer leaving area
73      {
74          pickUpText.SetActive(false);
75
76      }
```

Figure 46. craftgun script

```
39      // Update is called once per frame
40      void Update()
41      {
42
43          if (gunBuilt == true) //if gun has been built
44          {
45              pickuptext.SetActive(true);
46              if (Input.GetKey(KeyCode.E)) //pick up gun
47                  {
48                      tableGun.SetActive(false);
49                      Destroy(pickuptext);
50                      hasgun = true;
51
52                      pistol.SetActive(false); //when new gun is picked up disable other guns
53                      ar.SetActive(false);
54                      newGun.SetActive(true);
55                      bar.SetActive(true);
56                  }
57              }
58      }
59
60        public void OnTriggerStay(Collider other)
61      {
62          if(other.gameObject.tag == "Player")
63          {
64
65              if(g.hasAllParts == true)
66              {
67                  buildText.SetActive(true);
68                  if (Input.GetKey(KeyCode.E)) //build gun
69                  {
70                      Destroy(buildText);
71                      buildText.SetActive(false);
72                      scope.SetActive(true);
73                      mag.SetActive(true);
74                      barrel.SetActive(true);
75                      StartCoroutine(wait()); //wait a second to prevent picking up and building to happen at the same time
76                  }
77              }
78              else
79              {
80                  missingPartsText.SetActive(true);
81              }
82          }
83      }
84
85          IEnumerator wait() //wait a second
86      {
87          yield return new WaitForSeconds(1);
88          gunBuilt = true;
89      }
```

Collecting the parts of the gun and crafting it are controlled by two different classes. Figure 45 and Figure 46 respectively show a snippet of code from these classes.
Line 28 from Figure 45 shows a “onTriggerStay” method. This is a method which constantly runs while the player is in a certain area which is defined in the engine. Line 30 is condition statement which is checking if the player has collided with a certain game object which as a specific tag in this case it is checking if the player has collided with “gunscope”. If true text appears on the players user interface with an option to pick up the part with the “e” key. Once picked up the game object disappears from the map and a Boolean is triggered in line 36 noting that the player has the part. This is repeated two more times for the remaining parts. Line 72 shows another method “onTriggerExit”. This method is ran once after the player leaves the area. Within this method disables any text on the screen. This is so the pickup text isn’t permanently on the player’s screen.
The onTriggerStay method in Figure 46 is responsible for building the weapon on the crafting table. Line 62 in this method checks if the player is next to the crafting table and if so checks if they have collected all the parts from the previous script. If also true the gun can be built as seen on Figure 43.

This weapon differs form the previous ones as it is a high damage, low fire rate weapon. This weapon is also different from the other two as it does not use any ammo. Instead, it uses energy which decreases after every shot. The energy bar is clearly shown in Figure 41 as a blue bar at the bottom of the screen. The energy works like the stamina and health system as it recharges overtime after being shot, back to its maximum energy level. This weapon also contains a scope shown in Figure 47 the player can use to aim with unlike the pistol or AR. The scope can be used by holding mouse 2.

Figure 47. Sniper scope

6.3.4 Shooting

To implement shooting with my weapons I used Unity's raycast system [13]. Raycasts are invisible rays that can be shot from a point of origin in a certain direction with a maxim travel length. All my enemies have colliders which detect if they are hit with these raycasts and if so, there health is affected. Since these raycasts are invisible I have implemented two ways the player can tell where they are shooting from, the first is a crosshair placed in the centre of the screen which shows where the gun is aiming, the second is a hit effect which happens if the raycast hits something which is shown in Figure 48. This hit effect is changed to a red dust particle if the raycast hits an enemy.

Figure 49. Crosshair

Figure 48. Raycast hit effect

6.3.5 Ammo and reloading

The pistol and AR both have separate ammo systems. Each gun has two ammo variables, mag ammo and total ammo. Total ammo is the ammo the player has for that gun excluding the mag ammo and the mag ammo is the ammo already in the gun. When shooting the gun uses the mag ammo and once mag ammo is zero the player can't shoot until they reload from the total ammo. Reloading can also happen when the players mag ammo is not at its max amount. Figure 50 and Figure 51 respectively show the ammo of the pistol after being shot once then reloaded. Once total ammo is zero the player can't reload and if the mag ammo is zero, they then can't shoot.

Figure 50. Pistol ammo after being shot once

Figure 51. Pistol after reloading

6.3.6 Gun animation

I implemented a weapon sway animation while the player is moving. This means that the gun will bounce up and down. This is done in code by programming the weapon to bounce up and down when it detects that the player is moving. I did this in code as it allowed me to customise how I wanted the sway to look versus how a premade animation would look. Similarly, to weapon sway I have also implemented a reload animation done is code as well. This lowers the gun from the player's screen until the reload is done then brings it back to the original position.

6.3.7 Gun audio

Each gun is programmed to play an audio clip every time they shoot and reload. The same audio clips are used for all weapons but are tweaked using the audio tools within unity to change the pitch of each sound.

6.4 Wave system

The wave system implemented in my game spawns a mix of enemy one and enemy two in each wave with the number of enemies increasing per wave. Each enemy spawned is also given a random speed so that the enemies don't cluster up together when navigating to the player. The waves are spawned in specific areas of my map depending on where my player is at the time, if the player moves between specified areas of my map the place the wave is spawned will change as well, for example if wave five is currently being spawned and the player is currently located in area one and then moves to area two the rest of wave five will then be spawned in area two.

6.4.1 Spawn areas and Spawn points

To set up the dynamic spawning of enemies depending on where the player is I sectioned off areas of my map into six different areas which are shown in Figure 52.

Figure 52. Spawn areas

Within each of these areas there are up to five individual spawn points where the enemies are instantiated. This is done by creating an array of spawn points (Figure 53) within these areas which contain invisible game objects which act as markers for the spawn points.

Figure 53. Array of spawn points

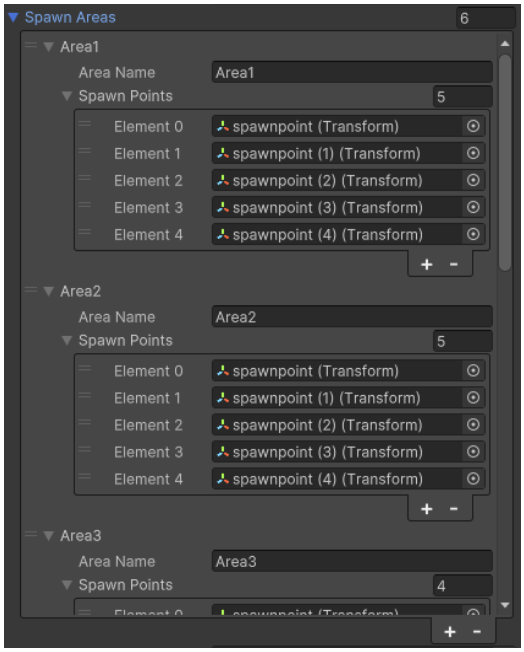


Figure 54. Choosing spawn location from WaveController class

```
    Transform[] GetSpawnPointsForArea(string areaName) //spawn points in array. returns array of spawnpoints in areas on the map
    {
    foreach (var area in spawnAreas) //loops through each area in spawnAreas
    {
        if (area.areaName == areaName) //checks if current area matches area name
        {
            return area.spawnPoints; //if true return correct spawn points
        }
    }
    return null; //no spawn area found
    }

    void OnTriggerStay(Collider other) //runs while player is inside a box colliders
    {

        //all areas of the map
        //each area has 1-5 spawn points
        //when staying in area "currentPlayerArea" variable is changed to the area the player is in

    if (other.CompareTag("Area1"))
    {
        currentPlayerArea = "Area1";
    }

       if (other.CompareTag("Area2"))
    {
        currentPlayerArea = "Area2";
    }

          if (other.CompareTag("Area3"))
    {
        currentPlayerArea = "Area3";
    }

          if (other.CompareTag("Area4"))
    {
        currentPlayerArea = "Area4";
    }

          if (other.CompareTag("Area5"))
    {
        currentPlayerArea = "Area5";
    }

          if (other.CompareTag("Area6"))
    {
        currentPlayerArea = "Area6";
    }

    }

}
```

Figure 55. Spawning the wave from waveController class

```
51      IEnumerator SpawnWave() //spawn wave slowly
52      {
53          int spawnCounter = 0; //enemy spawned
54
55          while (enemiesRemainingToSpawn > 0) //spawn enemies until done
56          {
57              enemiesRemainingToSpawn--; //decrement amount of enemies to spawn
58              spawnCounter++; //increment amount spawned
59
60              Transform[] spawnPointsInArea = GetSpawnPointsForArea(currentPlayerArea); //get array of spawn points based on player location
61              if (spawnPointsInArea != null && spawnPointsInArea.Length > 0) //if spawn found
62              {
63                  Transform spawnPoint = spawnPointsInArea[Random.Range(0, spawnPointsInArea.Length)]; //select random location in area
64                  GameObject enemyToSpawn = spawnCounter % type2SpawnRate == 0 ? enemy1 : enemy2; //choose both type on enemies
65                  GameObject spawnedEnemy = Instantiate(enemyToSpawn, spawnPoint.position, spawnPoint.rotation); //instantiate enemy
66
67                  if (enemyToSpawn == enemy1)
68                  {
69                  spawnedEnemy.GetComponent<EnemyAI>().OnDeath += OnEnemyDeath; //get enemy from engine
70                  }
71                  else if (enemyToSpawn == enemy2)
72                  {
73                      spawnedEnemy.GetComponent<EnemyAI2>().OnDeath += OnEnemyDeath; //get enemy from engine
74                  }
75               }
76
77              yield return new WaitForSeconds(1.0f); //wait 1 second between every spawned enemy
78          }
79      }
```

Figure 54 shows a snippet of code from the waveController class which shows how the spawn points are retrieved and how areas are selected. The GetSpawnPointsForArea method at line 102 is responsible for retrieving the spawn points within an area of the map. The foreach loop at line 104 searches each area in the array of spawnAreas and checks if the areaName matches the current area the player is in. If it finds the area it returns the spawn points else, it returns null. The onTriggerStay method at line 114 is responsible for detecting when the player enters an area which then changes the currentPlayerArea variable to reflect their position.
Figure 55 then shows code from the waveController class which spawns the wave from the selected location. Line 60 retrieves the spawn points by calling the GetSpawnPointsForArea method with the currentPlayerArea as a parameter.

6.5 Power ups and Purchasable items

This section will cover all the items within my game that the player is able to get and purchase with points.

6.5.1 Points

Every time the player hits an enemy with bullets, they earn a set number of points which they can use to purchase items or doors with. The point counter is located at the bottom left of my screen which is noticeable in Figure 19 which shows the number “10000”. These points will also be crucial for my player to unlock all areas of my game.

6.5.2 Stamina upgrade

I created a simple box in blender which I have used for my stamina upgrade kit game asset which is shown in Figure 56. When the player purchases the upgrade, their maximum stamina increases by 50 percent. This therefore allows them to run for longer.

Figure 56. Stamina upgrade

6.5.3 Health upgrade

I used the same asset I created for the stamina upgrade to also act as a health upgrade game object just with a different colour which is shown in Figure 57. Similarly, to the stamina upgrade when the player purchases the health upgrade their maximum health increases from 30 to 50. This therefore simply allows the player to take more damage before dying,

Figure 57. Health upgrade

6.5.4 Ammo boxes

Since the pistol and AR have a limited amount of ammo, I needed a way for my player to be able to gain more ammo. To achieve this, I added ammo box assets for this reason. The player can purchase ammo for these weapons at these ammo boxes which are located around the map. Figure 58 shows an ammo box for the AR. The player is able to purchase ammo from here for 800 points for and gain a set number of bullets. Each ammo box is specific to a specific gun for example the ammo box for the pistol is cheaper to buy ammo from compared to the ammo box for the AR.

Figure 58. Ammo box

6.5.5 Locked door

To access an area of my map the player must open a set of double doors which costs points to do so. In this area there are other key items which make it necessary for the player to explore here. Figure 59 and Figure 60 respectively show the doors before and after being opened.

Figure 59. Set of doors before being opened

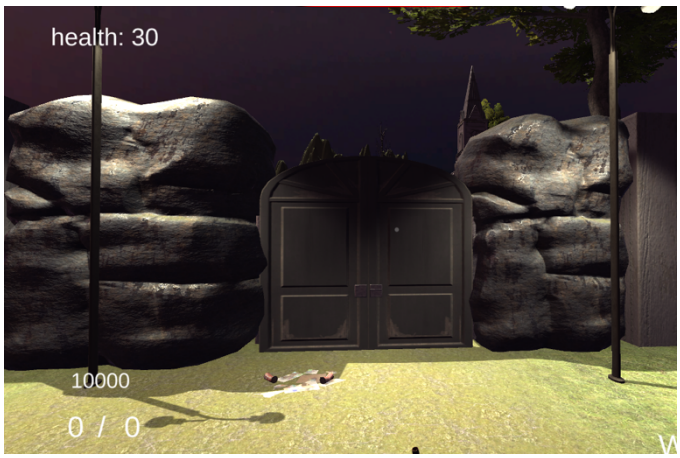


Figure 60. Doors after being opened using points

6.5.6 Saw

Another section of my map is blocked off by barbed wire. To access this section the player must find the saw game object which then allows them to cut the barbed wire which intern opens the blocked area of the map. This barbed wire also inflicts damage to the players health every time they walk into it. Figure 61 shows the saw which the player can pick up for free. Figure 62 shows the barbed wire which is blocking the section of the map.

Figure 61. Saw game object

Figure 62. Barbed wire blocking an area of the map

6.6 Escaping

To finish my game without dying I have implemented a way that the player can escape and end the game. On the opposite side of the map the player can find a car which is shown in Figure 63 with visible missing parts.

6.6.1 Car

Figure 63. Car with missing parts

6.6.2 Car parts

To fix this car I have hidden three car parts around the map which include a tyre, a key, and a fuel can (Figure 64).

Figure 64. Car parts

The tyre and fuel can will be located behind the two separate locked areas of the map. This makes it a requirement for my player to explore everywhere in my game. To obtain the key the player must kill the boss enemy who will then drop it as an item.

6.6.3 Car after being fixed

Once the player collects all the parts, they have the option to add the parts onto the car all at once. This physically changes the look of the car by adding the missing tyre (Figure 65). This also plays two audio clips for fuelling the car and adding the tyre. Once the parts are added it gives the player the option to escape by pressing “E”.

Figure 65. Car after being fixed

6.6.4 End screen

Once the player decides to escape, they get transported into the car. The main camera points to the dashboard of the car which displays some stats of the game and an option to go to the main menu. This screen is shown below in Figure 66. This screen also plays an engine noise to add to the effect of being in the car.

Figure 66. Escape screen

6.7 Menu

I implemented a main menu (Figure 67) and a pause menu (Figure 70) within my game. The main menu is a central hub which contains a variety of pages. The main menu also includes a start button which loads the actual game and an exit button which closes the application. The pause menu is used within the actual game which freezes the game and displays some options.

6.7.1 Main menu pages

Figure 67. Main menu

Figure 68. options page

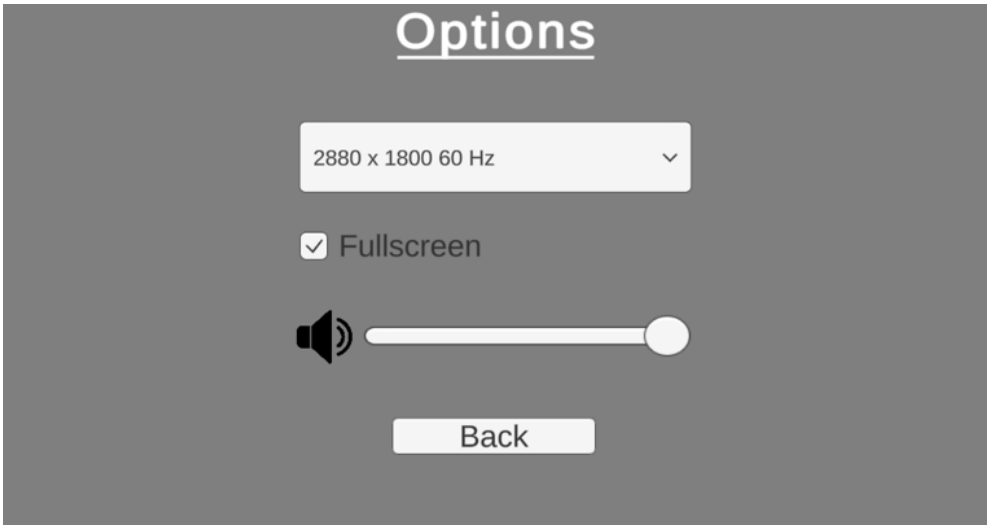


Figure 68 shows the options screen which can be accessed by the main menu. Here the player can change the games resolution via a drop-down menu, toggle full screen of the application on/off and change the master volume of the game.

Figure 69. Controls page

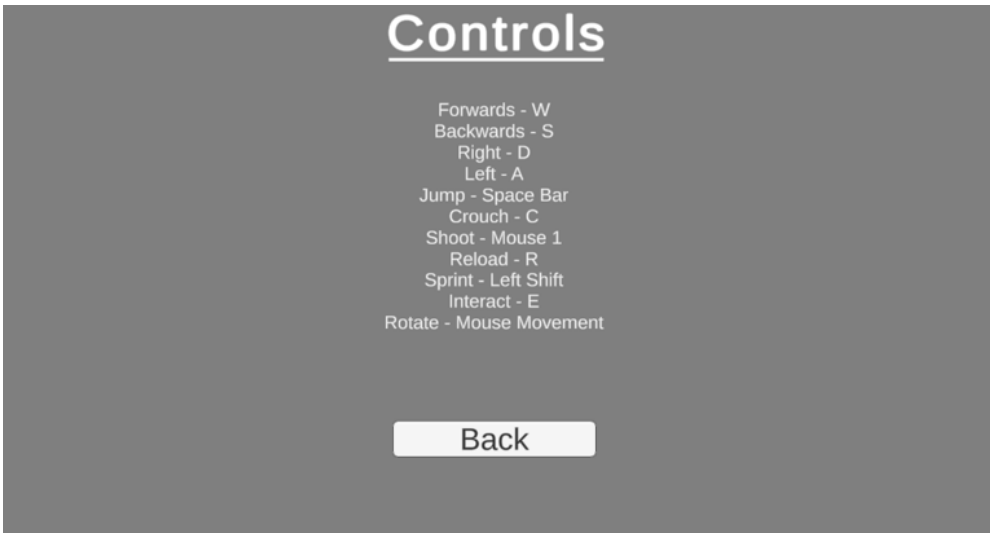
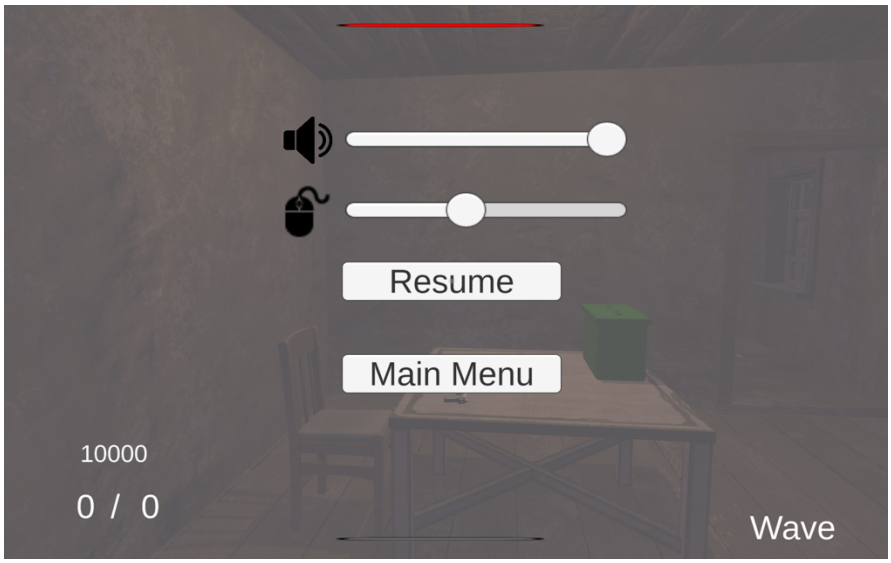


Figure 69 shows my control page which is also accessed by the main menu. This page simply just shows the control keys for all possible actions in my game. Although the controls within my game are similar to other solutions, this is still necessary as newer players may not be familiar with industry standard game controls.

6.7.2 Pause menu

The pause menu contains two sliders. The volume slider which is also in the options menu is also present in the pause menu, both sliders are the same and update each other when used in the main menu and the pause menu. The second slider is a mouse sensitivity slider. This is only present in the pause menu as sensitivity is something that needs to be changed a couple times usually to fit the player's needs.

Figure 70. Pause menu



7 User feedback

To gather user feedback on my system I sent out a build of my game along with a questionnaire to willing participants. The main target audience for testing my system were other university students who participated in the year 2 entertainment technology module. Students who took this course learnt about the Unity game engine and how to make several different types of games. This meant that the people testing my system had a solid understanding of game design and development. The feedback given from this survey was essential in identifying bugs I missed during development and also helped with balancing game mechanics.

7.1 Survey

The survey included two types of questions. The first type is question are about the user's playtime and achievements. These were asked to measure the user's engagement with the game. The second type of questions asked focused on potential improvements for my system. Out of all individuals asked to take part in my research, five people accepted and gave responses.

The following questions were a part of my survey:

“Did you play for at least 30 minutes?”
Yes
No

“How many attempts until you completed the game?”
1
2
3
4
5
6+
didn't complete

“Highest wave survived?”

“How difficult is the game?”
very easy
easy
normal
hard
very hard

“Should the games difficulty stay the same?”
Yes
Should be easier
Should be harder

“How many weapons did you find and use?”
1
2
3

“What are your opinions on the game map and what improvements if any should be added?”

“Are there any game mechanics you disliked and why (eg. weapon system, health system, stamina system)?”

“Is there any game mechanics that you really liked and why?”

“List any features you believe the game should have or would benefit from?”

“Please list any bugs you may have encountered”

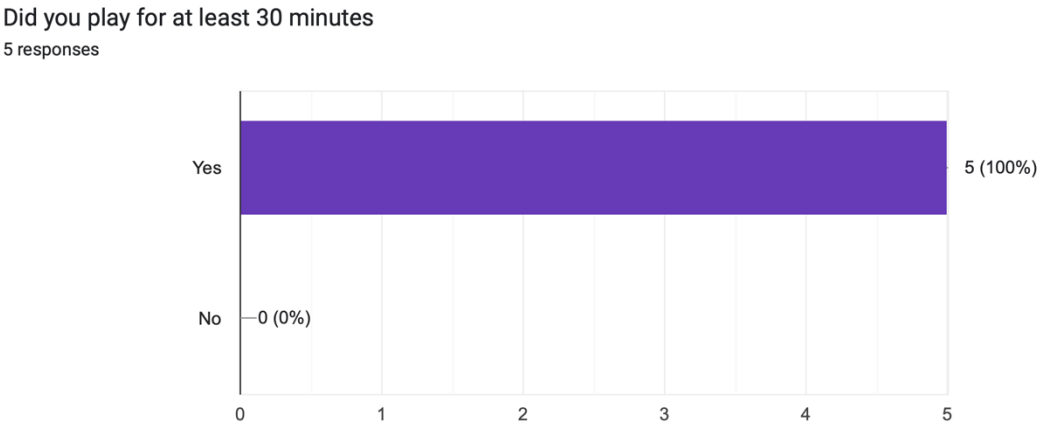
“What mechanics need balancing?”

7.2 Feedback evaluation

In this section I will go through notable responses from my survey which could help future iterations of my game.

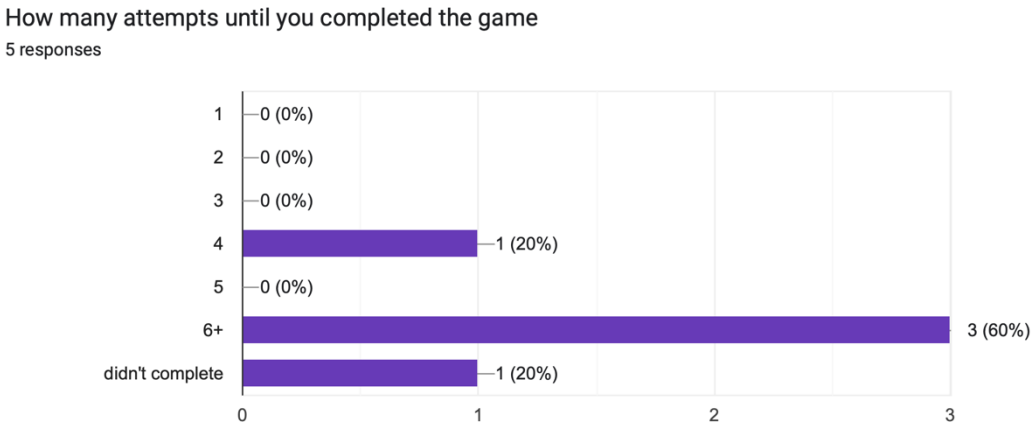
The first question asked in my survey is shown in Figure 71. This question is essential as it tells me that my participants played for enough time which would allow them to give me good feedback and informed feedback. Since all 5 participants played for what I believe to be enough time I can value their responses to a higher degree when considering improvements.

Figure 71. Game time question



The responses given for the question in Figure 72 show that for most players it took them a high number of attempts to complete the game. This indicates that my game may be on the harder side of the difficulty scale. Although this shouldn't be used for measuring if the difficulty is good as some gamers believe harder difficulty is better for games.

Figure 72. How many attempts did it take to escape question



The questions and responses in Figure 73 and Figure 74 allow me to measure if my game being difficult is good or not. Figure 73 tells me that the majority of the five plays felt that the game was a normal difficulty. Three answered normal and two answered hard. Figure 74 then tells me that three people answered that the difficulty should stay the same and two people said it should be easier. I can assume that the people who said that the difficulty was normal also said it should stay the same and the same with people who said the game was hard answered that the game should be easier. Since it's a 60 and 40 percent split between answers I can assume that most people will find my games difficulty fine but since these differences are still close it may be useful for me to balance some mechanics slightly to make my game easier such as increasing player speed by a bit or increasing a specific weapons damage.

Figure 73. How difficult is the game question

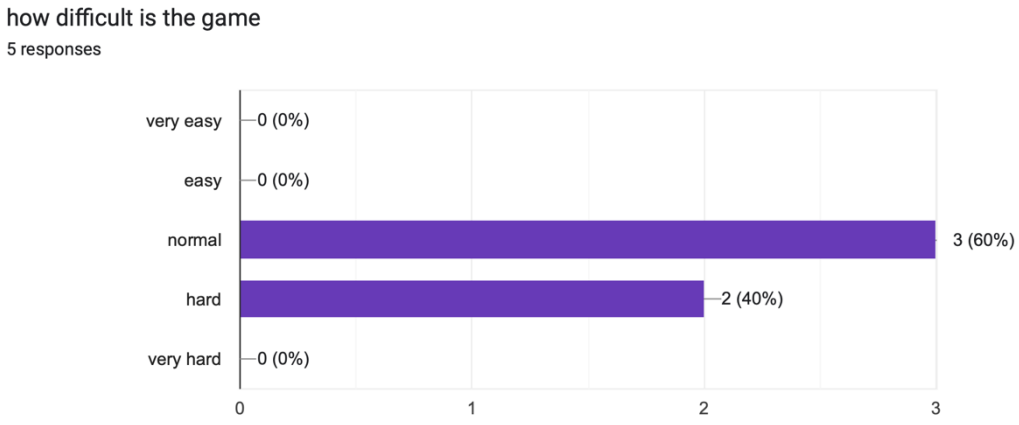


Figure 74. Should the difficulty stay the same question

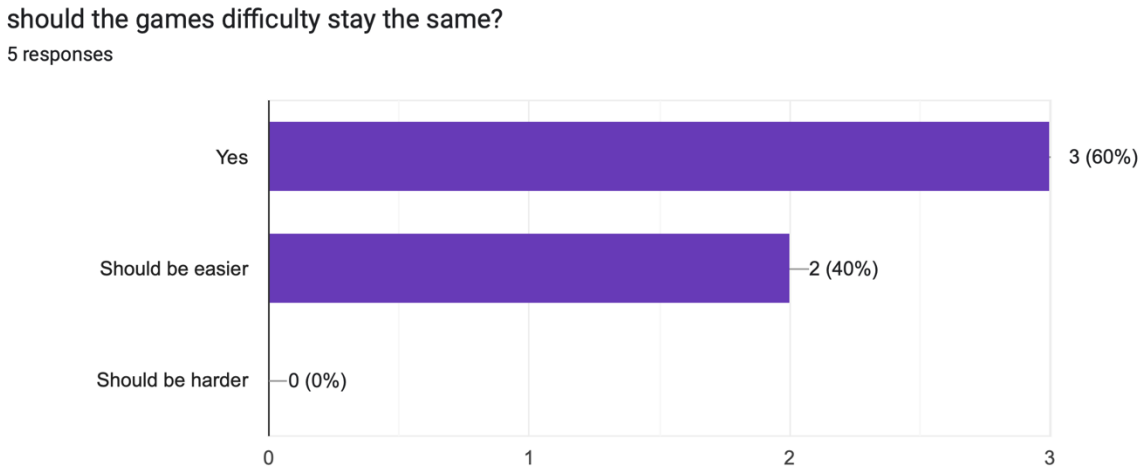
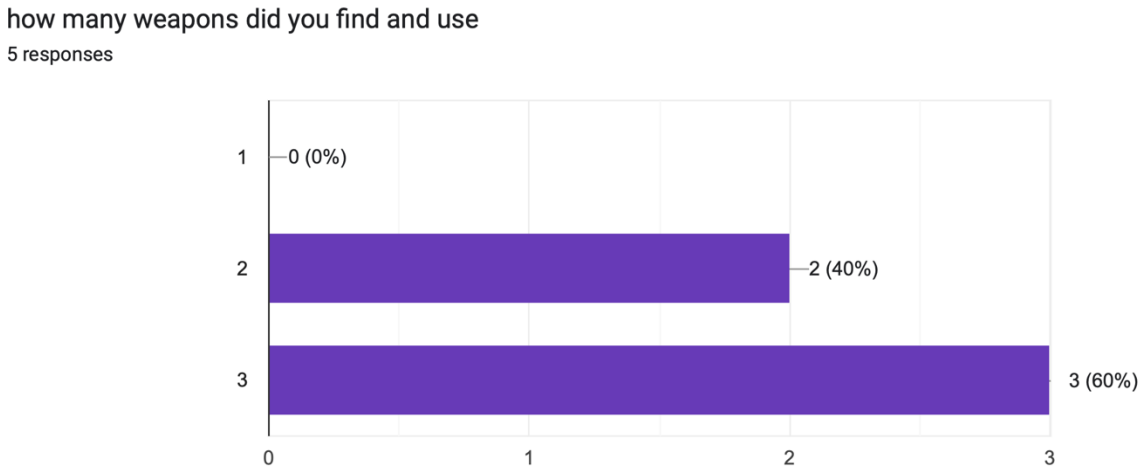


Figure 75. How many weapons found question

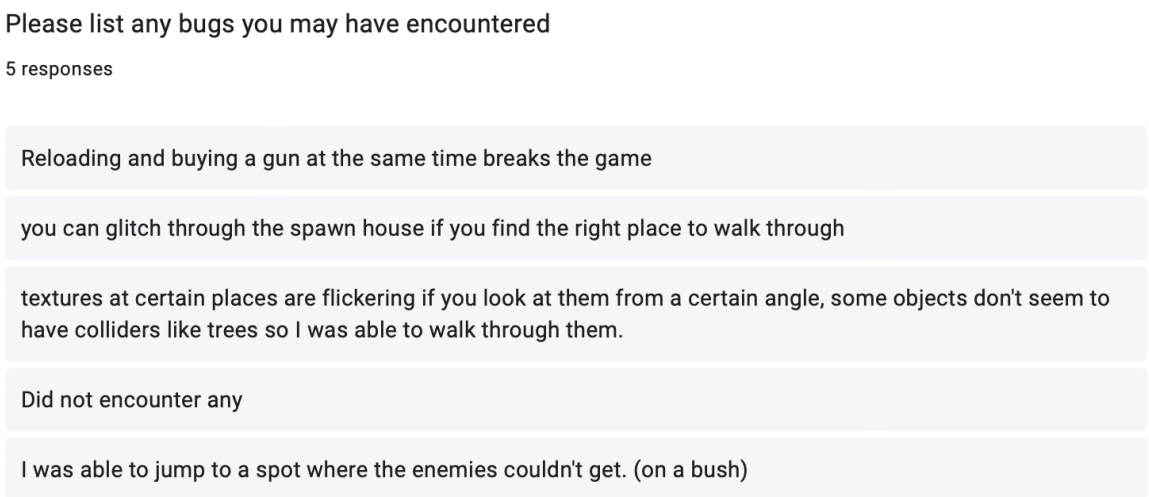


The question asked in Figure 75 tells me about the placements of the weapons in my game. At least two out of three of them were found among the participants. These answers tell me that the general placement of weapons is okay as the majority were located. However, since not all three were found I could implement more obvious hints to help locate them such as street signs that point to specific area or a brighter light at key components.

As expected, some bugs were found by the users. Figure 76 shows the list of bugs reported via the survey. I will go through each bug and how I could potentially solve them in future builds of the game.

.

Figure 76. Bug list from users



Bug report 1 – after testing this myself I can confirm this happens and is not intended. In a future version I think I can fix this by adding a simple condition to buying a weapon which does not allow them to buy it if “reloading == true”.

Bug report 2 – this occurs in a few places especially when the player sprints at game objects which posse colliders. I think I can minimise this by adding multiple colliders to these game objects as a backup collider just in case.

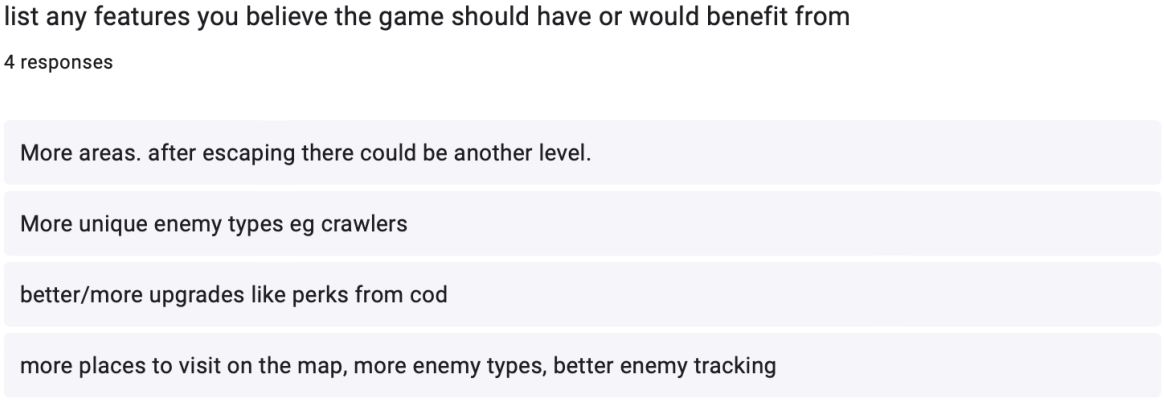
Bug report 3 – texture flickering happened at certain places during development but all I did was delete these textures and re add them back. Further details on where these textures are, would be key to fixing them. The trees were added as a terrain object. These don’t seem to be able to possess colliders and I wasn’t able to figure out how to add them. I could re add these trees as game objects which would then let me add colliders to them.

Bug report 4 – none encountered

Bug report 5 – I should either decrease the jump height or increase the height of the bushes to prevent this.

The final notable part from the responses given from the survey are the suggestions for what to add next. Figure 77 shows the response given.

Figure 77. List of wanted features



All the ideas suggested from the survey seem good to add to a fps game. If any future iterations are developed, I will look at these suggestions first and consider working on them first after bug fixing.

8 Critical Appraisal

8.1 Critical analysis

In this section, the final state of the project will be critically compared to against the original list of requirements. I will go through each requirement and state whether it has been met or not. I will also specify the section of the document where they have each been met.
I will discuss any requirements not met and potential solutions that I could explore to implement them in any future iterations of my project.

8.1.1 Essential requirements

Requirement	Has it been met?	Section of document
The player must be able to walk forward	YES	6.2.1 Movement
The player must be able to walk forward	YES	6.2.1 Movement
The player must be able to walk left	YES	6.2.1 Movement
The player must be able to walk right	YES	6.2.1 Movement
The player must be able rotate in every direction	YES	6.2.1 Movement
The player must have a gun	YES	6.4.1 Pistol
The player must be able to shoot	YES	6.3.4 Shooting
The game must have an enemy	YES	6.3.1 Enemy one
The enemy must be defeated after being shot by the gun	YES	6.3.1 Enemy one
The game must have a playable surface (map)	YES	6.1.2 Game map

8.1.2 Recommended requirements

Requirement	Has it been met?	Section of document
The player should have health	YES	6.2.3 Health system
The enemy should be able to move	YES	6.3.4 Ai navigation
The enemy should be able to find my player	YES	6.3.4 Ai navigation
The enemy should be able to damage my player via collision	YES	6.3.5 Attacks
The game should end when enough damage is taken to players health	YES	6.2.3 Health system
The gun should have a set amount of ammo that can be shot	YES	6.3.5 Ammo and reloading
The gun should have a magazine that reloads from ammo	YES	6.3.5 Ammo and reloading
The gun should only work if there is ammo in the magazine	YES	6.3.5 Ammo and reloading
The enemy should respawn in sets of waves	YES	6.4 Wave system
The map should have different terrains (levels)	YES	6.1 Game environment
The map should have textures such as concrete, grass, etc…	YES	6.1 Game environment
The map should have assets scattered over it such as bushes, trees, cars, etc…	YES	6.1 Game environment
Shooting should make shooting sounds	YES	6.3.7 Gun audio
Walking should make walking sounds	NO	n/a
Running should make running sounds	YES	6.2.4 Player audio
Enemy should make noises	YES	6.3.5 Attacks
My player should be able to jump	YES	6.2.1 Movement
My player should be able to crouch	YES	6.2.1 Movement

8.1.3 Optional requirements

Requirement	Has it been met?	Section of document
My player may have a stamina limit for running	YES	6.2.2 Stamina
The game may have a visual representation for total ammo	YES	Figure 50. Pistol ammo after being shot once
The game may have a visual representation for magazine ammo	YES	Figure 51. Pistol after reloading
The game may have a visual representation for health	YES	Figure 23. Player health bar full
The game may have a visual representation for what wave they are on	YES	Figure 19. Camera view of player
The wave of enemy may get progressively harder	YES	6.4 Wave system
The enemy may have animation when moving	YES	6.3 Enemy
The game may have a start option	YES	Figure 67. Main menu
The game may have a quit option	YES	Figure 67. Main menu
The game may have a controls option	YES	Figure 69. Controls page
The game may have a pause option	YES	Figure 70. Pause menu
The game may have an un-pause option	YES	Figure 70. Pause menu
The game may have a credits option	NO	n/a
The player may be able to adjust the mouse sensitivity	YES	Figure 70. Pause menu
The player may be able to adjust the game volume	YES	Figure 70. Pause menu

8.1.4 Requirements missed

The first requirement I missed is sounds for when the player walks. This was simply missed due to time restraints. As I am using an agile methodology to develop my system, I decided to reassess the importance of this requirement and decided it wasn't as important as other requirements found in the optional requirements list. However, adding this requirement in the future if needed shouldn't be an issue as I did add sprinting noises while running. The addition of walking noises would follow a similar approach.

The second requirement I missed is including a credits page in my game. This was also due to timing restraints. Instead of including a dedicated page in my system I will simply include all assets used in a text document on my git repository with the final submission. If I had more time, I could have implemented this just like with the controls page in my menu as they are both info type pages.

8.1.5 Extra requirements added

As mentioned in the design section, extra requirements were identified during the implementation stage and were added to my project to adapt to evolving project needs. This section outlines and justifies the addition of these extra requirements.

Extra player requirements:

- Health regeneration – originally my plan was to have med kits that the player could use to regenerate their health with but after integrating stamina regeneration I was able to follow a similar approach into the health system as well without having to spend a long time on it, as I had already created the logic of regeneration that I was able to re-implement into other systems.

Extra enemy requirements:

- Added a second type of enemy – once I learned how to create an enemy ai I was able to re-use that knowledge and create another type which acts differently.
- Boss enemy – After I learned how to make a health bar for the player, I realised I could implement the same for an enemy. After giving the health bar to enemies I realised it made them look more intimidating, so I decide do create a third type of enemy and call it a boss enemy and give it a health bar while also removing the health bar from the more common enemies. So, this enemy had a clear difference between the other two I implemented a separate range attack and a separate method of navigation (patrolling).
- Animation - During planning I omitted animations by mistake. After realising this during development I learnt how to use the animator in unity and implemented these into my enemies. Without learning and adding these, my enemies would have given no indication of the actions that they are taking.

Extra gun requirements:

- AR – added second gun after I learned about creating weapons in unity. I wanted each gun to be different, so I made this act differently to the pistol by making it shoot faster and have a higher range.
- Buildable weapon – after learning how to make game objects disappear and appear I was able to figure out how to make a craft able gun. Since I wanted this to be unique, I learnt about canvases in unity and added a scope to the gun. Since I already had the logic of regeneration from the stamina bar, I was able to alter it and implement it into the gun to act as a substitute for the ammo system.
- Gun animation – when moving with the guns it did not feel fluid, so I added movement animations to them. Same with reloading.

Extra game requirements:

- Escaping – once I learnt how to create the buildable weapon I thought if adding a similar system but with a car. If all car parts are found the player can find a secret escape route to take to end the game without dying.
- Locked doors – added these to prevent very quick speed runs of my game

8.1.6 Project results

Next, I will evaluate my project results and discuss what went well and what went badly in view of my original aim and objectives.

The main aim of my project was to create a fps survival game. I believe that I have accomplished this task as I have implemented various first-person shooter mechanics such as movement, shooting, and enemies. Another aim was to ensure an immersive and challenging game. I believe this was also achieved through the development of creating a 3d environment using premade assets and creating challenging wave system which spawns enemies.

What went well:

- I believe the overall creation of a 3d game went well. My past experiences with game development during the entertainment technology module taught me how to create 2d games, which meant I had to learn new skills when switching to a 3d game.
- I was able to implement a wide variety of game mechanics which ended up being successful enough to add to the game.
- I was able to test my game with a couple of different computers and it seems to run fine on all semi modern systems.

What went badly:

- Asset collection was a challenge as there were only a limited amount of free assets which fit the atmosphere of my game. For example, I scrapped an idea for a crawler zombie as I was not able to find a free crawling attack animation.
- I found it difficult to handle physics in my game. The main issue I kept encountering was that my player kept being able to walk through solid objects which have colliders.
- Map design was a challenge as I struggled to come up with a good-looking map. This also relates the asset collection issue as high-quality map assets usually came with a price tag.
- Testing could have gone better. I had my users test my game at the end. This left me with bugs still to fix with limited time left.

What I would do differently:

- Explore other asset stores, other than unity's, which provide free assets. This would lessen the chance of me not finding a suitable free asset that I need.
- Collect user feedback throughout development. This would help me fix bugs and adjust mechanics more frequently which would then provide a better end product.
- Explore and learn other methods of handling physics in a 3d game within unity.

8.2 Social, Sustainability, Commercial and Economic context

8.2.1 Impact on people

According to [14] individuals who play fps games like mine, tend to have better cognitive skills. This study concluded a test between two groups to test multitasking skills between the individuals. The group full of fps gamers came out with better multitasking abilities. If my project did happen to release on a wider scale with more development, it has a chance of impacting people in this way. Another positive impact of my game as well as other similar solutions, is to simply provide a source of entertainment and stress relief to people. Additionally, the interactive engagement of games could provide fun mental exercises such as puzzle solving. However, it is important to also consider potential negatives from my game or other similar products. Games can be addicting to certain people and cause bad habits and damage to people's well-being [15]. Another potential negative impact games like mine could have on people is exposure to violent content, which is mostly concerning for younger audiences as they are more impressionable than the average person. Some mitigating strategies to counter these negatives could include offering in game advice on long screen time to users and making it clear to users the nature of the game.

8.2.2 Impact on businesses

Bigger and more complex video games created by much larger teams tend to be created by large businesses for the purpose of commercial success. However, projects like mine, with more time, could be sold on independent marketplaces like steam [16] which is a platform which allows individuals like me to list games for sale and split the revenue made by it.

8.2.3 Risk to society

I don't see any possible risks presented by my project to society. This conclusion is based off the way I have created it as it has caused no direct impact on the Earth or society, since it was digitally created on a computer and is not being made specifically for commercial use.

8.3 Personal development

By completing this project, it has allowed me to expand my knowledge in several aspects of computer science including entertainment technologies, C# programming and agile development of projects.

Before starting this project, I had a fundamental understanding of unity and had experience in creating 2d games from the previous year's entertainment technologies module. However, this was not enough so I had to do some personal learning and figuring out on creating a 3d game. Since after completing this project, I am confident that I could produce an even better result with all the things I learned throughout this project.

Another area of personal development that came with this project is C# programming. Although I had experience with this language, it never went far enough for me to consider myself confident at programming using C# until after completing all my requirements.

This project also provided the opportunity to learn more about the development tools within unity. Previously I was unfamiliar implementing animations within unity but after some research on the animator tool I was able to implement multiple animations for each of my enemies and learnt how to program those animations to play via conditions which take place in the classes of those enemies. Another tool with unity I was unfamiliar with were the terrain creation tools. These were fundamental for creating a 3d environment. I also learnt about implementing different audio and control options within unity which I previously had minimal understanding of.

Although I was already familiar with the agile development methodology, this project provided useful hands-on experience with using it as it trained me to produce constant builds of my game to show off during the bi-weekly supervisor meetings.

9 Conclusion

To conclude, I have developed a first-person shooter game within the unity game engine and using C# to program my game. I deployed an agile approach to solve my problems and delivered frequent builds of my project which showed the progressive implementation of most of my requirements with the addition of extra requirements which were necessary to respond to evolving project needs. Considering all these factors, I believe that the aims and objectives of my project have all been met meaning that overall, my project was a success. Although I believe my project is a success when compared to my original aims, the system still has room for improvement and expansion. If my game were to continue development, I could implement the two minor requirements which I missed with the addition of starting work on all the suggestions and bugs pointed out by my survey participants.

Bibliography

[1] L. O. Mendes, L. R. Cunha and R. S. Mendes, “Popularity of Video Games and Collective Memory,” *Entropy,* vol. 24, no. 7, p. 860, 2022.

[2] I. Buyuksalih, S. Bayburt, G. Buyuksalih, A. Baskaraca, H. Karim and A. A. Rahman, “3D MODELLING AND VISUALIZATION BASED ON THE UNITY GAME ENGINE – ADVANTAGES AND CHALLENGES,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences,* vol. 4, no. 4, p. 161–166, 2017.

[3] J. D. González, J. H. Escobar, H. Sanchez, J. D. L. Hoz and J. R. Beltran, “2D and 3D virtual interactive laboratories of physics on Unity platform,” *Journal of Physics: Conference Series,* vol. 935, no. 1, pp. 1-7, 2017.

[4] A. BARCZAK and H. WOŹNIAK, “Comparative study on game engines,” *STUDIA INFORMATICA Systems and information technology,* vol. 1, no. 2, pp. 5-24, 2019.

[5] Unity, “unity-plans-student-and-hobbyist,” [Online]. Available: https://unity.com/pricing#plans-student-and-hobbyist. [Accessed 10 April 2024].

[6] unity3d, “Unity Documentation,” [Online]. Available: https://docs.unity3d.com/Manual/system-requirements.html. [Accessed 10 April 2024].

[7] W. Kramer, “What makes a game good,” *Game & Puzzle Design,* vol. 1, no. 2, pp. 84-86, 2015.

[8] G. Rivera, K. Hullett and J. Whitehead, “Enemy NPC Design Patterns in Shooter Games,” in *Proceedings of the First Workshop on Design Patterns in Games*, 2012.

[9] Unity, “Unity Asset Store,” [Online]. Available: https://assetstore.unity.com. [Accessed 10th April 2024].

[10] Adobe, “Mixamo,” [Online]. Available: https://www.mixamo.com/#/. [Accessed 10th April 2024].

[11] S. Prescott, “Games: Breaking the game: The beauty of a gameworld's outer limits,” *The Lifted Brow,* no. 23, p. 30, 2014.

[12] Unity, “Unity Manual,” [Online]. Available: https://docs.unity3d.com/Packages/com.unity.ai.navigation@1.1/manual/Glossary.html. [Accessed 20 April 2024].

[13] Unity, “Unity Documentation,” [Online]. Available: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html. [Accessed 23 April 2024].

[14] P. Kearney, “Cognitive Callisthenics: Do FPS computer games enhance the player's cognitive abilities?,” in *Proceedings of DiGRA 2005 Conference: Changing Views – Worlds in Play.*, Auckland, New Zealand, 2005.

[15] I. Mylona, E. Deres, G. D. Dere, I. Tsinopoulos and M. Glynatsis, “The Impact of Internet and Videogaming Addiction on Adolescent Vision: A Review of the Literature,” *Froniers,* vol. 8, no. 63, pp. 2-6, 2020.

[16] Valve, “Steam,” [Online]. Available: https://store.steampowered.com. [Accessed 30 April 2024].