Final Year Project Report

**Full Unit – Final Report**

# CS3821 Building an FPS Game: Technical background, tools, and culture.

Harshdeep Chopra

A report submitted in part fulfilment of the degree of

**BSc (Hons) in Computer Science**

**Supervisor:** Dave Cohen

Department of Computer Science

Royal Holloway, University of London

March 26, 2021

# Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count:15208

Student Name: Harshdeep Chopra

Date of Submission: 26/03/2021

Signature: Harshdeep Chopra

# Table of Contents

# Abstract

Throughout this project and report, I have been able to discover key aspects of the Unity framework alongside the fundamentals of game design. The proof of concepts that have been developed has provided me with a structure for the final game that I want to build. Through these proof of concepts, I have explored the use of assets, materials, logic trees and C# Scripts for my gameObjects to act on. By doing this I have achieved competency in game design and confidence in the way my game will play.

This report underlines the proof of concepts developed and how they help the 3D FPS Game in certain aspects. It also covers the type of code that should be written, in terms of whether specific game design patterns should be implemented and the benefit they provide to the project.

Over the course of the project another focus has to become an expert in the algorithms and mechanics in games programming. Gaining competency in Collison, rigidbody assets, nav mesh systems and other mechanics helped me develop my scripts and as a consequence my knowledge of the Unity framework and C# has grown.

Finally, the report highlights the background of game development and the way gamers, developers and publishers think and the impact this causes on wider society. As an aspiring developer and a fan of gaming, it is important to know the full lifecycle of a published game and the ideologies of developers separate to coding and what they believe makes a good successful game.

# Project Specification

Firstly, the aim of the chosen project was to build a game which consisted of independently selecting a framework and coding language. From this, I decided to build a 3D first-person shooter game using Unity. The Unity GUI is very user friendly and allowed me to learn and build on the framework with ease. Secondly, the project also asked for proof of concepts which are developed through Unity and pushed to the GitHub repository that stores all relative files and information of the developing game. Finally, the research section of the report is a study on the psychology of gamers and the impact of games in everyday society.

# Chapter 1: **Introduction**

Building a 3D first-person shooter game requires the use of multiple different gameObjects, assets and scripts linking them together It also involves research into materials, design patterns and environments to use. This report explains these topics mentioned and covers relevant concepts and how they are implemented, developed, and tested. In addition to this, the report also includes a deep dive into gaming culture and the psychology of gamers. By exploring these aspects, I have been able to research the full impact of a game lifecycle and all the stakeholders that are involved with their direct impact to the game and its community.

## 1.1 Aims & Objectives

The core aims of this project were to familiarise myself with the tools and fundamentals of C# and the Unity framework, enabling me to build a game from an empty scene. The game that I chose to develop is an endless hoard zombie survival game for which multiple concepts were implemented. This report breaks down the concepts needed for the project to be built by explaining the importance of specific implemented features. Key pieces of code that make gameplay fluid through the concepts are discussed.

Tools and fundamentals that I used in my game were researched through Unity documentation, reports, and videos. It is important to have knowledge on the tools being used in Unity as many gameObjects rely on one another and the memory space being allocated. In the background theory section of the report, I cover the tools that I have or could have used in my project and their use, requirements, actions, and cost on game performance to use.

The main objectives of this project were to create (script) a fully controlled player that has free to roam movement which works alongside a well-designed first-person view. Another objective was to develop gun mechanism scripts that allow different weapons to be implemented into the game. These weapons then have their own damage, ammunition and can be cycled through. In addition to this, elegant AI enemies that all have their own routes to swarm user and deal different damage from different ranges were also created. Finally, a score counter, which is dependent on the number of enemies killed and the time the player has been able to stay alive. This is all discussed in more detail through the report as well as a chapter on the psychology of gamers and the damage that publishers and game developers may cause to everyday society.

## 1.2 Motivation

The reasoning behind my choice of this project and report is to provide insight into game development and how a developer would think to structure a game. From this project I expect to achieve a competent level of understanding of C# and Unity which will in turn allow me to adapt my software engineering skills to become a better overall computer scientist. The report allows me to express the code I have written alongside covering key ethical issues of gaming and place myself in the shoes of every individual team that is involved in the cycle of a game from its development to its release.

# Chapter 2: **Technology and tools for supporting Games Development**

The two main pieces of software that are used concurrently throughout the project are the Unity3D framework and the GitHub version control system. Without this type of software, game development would still be hard to manage, maintain and develop. Due to leaps in technological advancement, developers are now able to use software like Unity to code and construct games using a graphical user interface and store their progress and files into cloud repositories such as GitHub to pick up and continue work on any device at any location.

## 2.1 The Unity framework

The game engine is the epicentre of development. This software is where scripts are coded, assets are initialised, and customisability takes place. Frameworks differ from one another based on the language of code and graphics capabilities. Frameworks like PyGame take python as the main language and can be coded from an embedded IDE over the internet. Unreal engine uses C++ which takes longer to code with however gives the ability for raytracing, providing greater graphical quality and therefore AAA titles use this framework. Unity is a great framework for simple projects, C# is an easier to learn coding language and setup is very simple to understand, as explained below.

The setup of Unity is the key to the development of the game and also contributes to creating a game build ready for publishing. It is very straightforward to set up and allows users to begin building specific scenes in a matter of minutes. The process starts by downloading the Unity Hub [1] from the Unity website and running this on the chosen operating system by the user. Once run, the Hub allows the user to name their project, providing an option to either start from a template or an empty scene (see Figure 1).

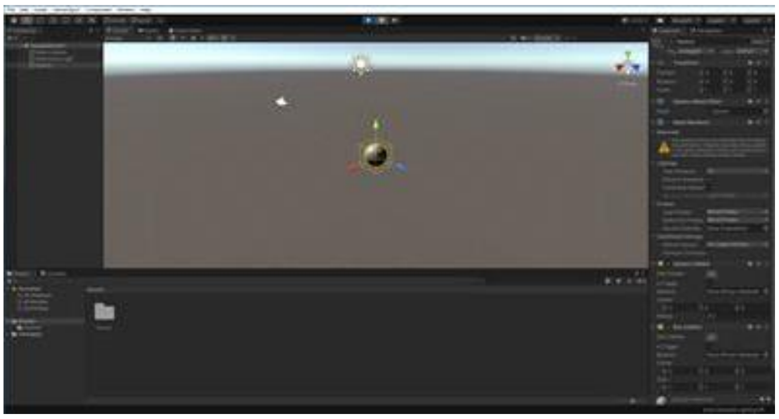


Figure 1. *The first empty scene on Unity 3D load-up*

Figure 1 demonstrates the Default Unity layout, from which users are able to select different views of the game build. The default Unity structure displays the scene, inspector, hierarchy, and project tabs which are customisable to preference.

### 2.1.1 Hierarchy

The hierarchy that is visible in the top left, is where users can see the assets in the current scene. Scenes will typically load with the main camera and lighting assets. By right-clicking, users can add more gameObjects into the scene. These can be configured through the inspector panel. Unity uses real-time views so whenever something is added, or deleted users can see it in the scene.

#### 2.1.2 Inspector

The inspector panel displays the settings and configurations of a specific gameObject. This provides focus on the object and goes into detail with all the components you can add as well as the scripts to write.

#### 2.1.3 Project

The project panel is the organisation of the game, its objects. scripts, materials, and packages. Here you can simply drag and drop assets onto a gameObject for customisability and allows users to set prefabs (prefabrications) for objects that may appear multiple times in the game. Prefabs that are regularly used in the project include: guns, trees, enemies, and animations.

#### 2.1.4 Asset store

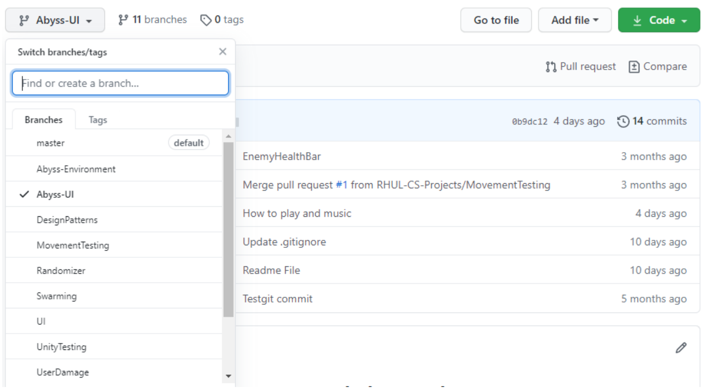
The unity asset store is fundamental in the project. Due to time constraints and limitations of the situation it is difficult to build your own 3D assets and sprites for the game. The Unity asset store allows users across the world to publish their assets to then be able to use in my project. This is beneficial as it requires simple tweaking to use custom assets in the game as well as provide a depth in the game rather than using the standard shapes that Unity provides.

## 2.2 GitHub Version control system

Version control systems (VCS) are used to record changes to files over time so specific versions can be recalled for later use. A complete history of every file added to the repository is available, allowing users to revert to older files or see notes and work on bug fixes. VCS also allows the use of independent streams (branches) to then be merged together for version release. Version control can be either centralised, which has a central copy on a server for example SVN or distributed, which doesn't rely on a central server and can be distributed and accessed anywhere such as Git

The VCS of choice for this Project was GitHub. GitHub is a website that allows you to host and control your projects. These are stored in the open source VCS, Git. My reasoning for choosing GitHub was the simplicity of managing my project. Using the command prompt can be quite daunting at times and GUIs are much easier to understand and navigate across. The option to have both as well as have simple connection between Unity and GitHub made it a great choice.

The project was continuously split up into branches over its development. Branches allow you to separate your project into separate folders that you can work on at different times (see figure 2). Once completed with a branch you can merge back into the master folder which holds the fully developed version. I used branches on UI, Environmental build, Player/Camera control and Enemy scripts. Branches helped me develop code without intervening with other aspects of the game and created ease of managing updates in my project.



| | |
|---|---|
| master | default |
| Abyss-Environment | |
| ✓ Abyss-UI | |
| DesignPatterns | |
| MovementTesting | |
| Randomizer | |
| Swarming | |
| UI | |
| UnityTesting | |
| UserDamage | |

0b9dc12 4 days ago 14 commits

| | |
|---|---|
| EnemyHealthBar | 3 months ago |
| Merge pull request #1 from RHUL-CS-Projects/MovementTesting | 3 months ago |
| How to play and music | 4 days ago |
| Update .gitignore | 10 days ago |
| Readme File | 10 days ago |
| Testgit commit | 5 months ago |

Figure 2. *GitHub Repository with a view of the current branches and the files inside this branch*

VCS with unity was seamless. Both Unity and GitHub make it easy to locate the source folder for the project and the branch. All you need to do is copy and paste the folder into your GitHub branch. You are then able to push your branch to the origin, so it is on your Git repository and not just your local system. GitHub provide a list of Ignore Files using a command .gitignore. Premade lists for unity are available to make better use of your VCS and avoid clutter. Unity also aid version control by allowing all files to be .meta files. This keeps all current settings for any asset, scene, or code to be stored onto the VCS.

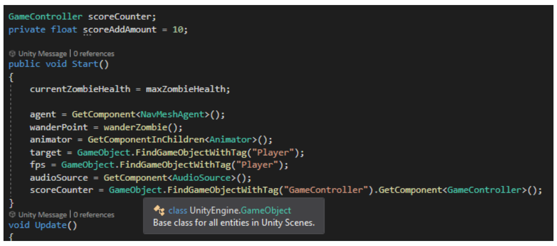
## 2.3 Integrated development environments

Coding is done inside of development environments, whether that is on a standard environment like notepad text editor or something more complex like Visual Studio (VS). IDE's provide a place to code and script in projects and is where the majority of a developer spends their time and therefore have adapted to increase programmer productivity through syntax highlighting, libraries, and debug consoles. [2]

Most current IDE's make use of tools to make a better coding experience. These are adapted differently between IDE's as each specialises in its own form of coding and languages it supports. For C# it is clear thar VS holds the title for the most popular IDE. This is because of its adaptability with extensions and full support of IntelliSense and C# debugging. C# is used by the Unity Framework due to its versatility and use in other projects aside from games development. Knowledge in C# can be adapted to other projects and similar language due to its ergonomic syntax. Although Unity allows the choice of scripting in any language and on third party IDE's, the pairing of C# with VS provides a seamless and easy to read coding experience. Some of the important properties are listed below.

Firstly, VS is owned by Microsoft [3] and is used for large scale projects around the world. Due to its affiliation with Microsoft, VS provides an ever-growing library of plugins and extensions that can be added to code to help it run better, add more depth, or adapt to different environments. The integration of IntelliSense is seen to be vital in most IDE's in the current age of development as it provides simple code completion using AI to find what function you may be looking for. Auto completion is used in everything from search engines to messenger applications and on average can reduce typing from up to 25%. Evidently this allows developers to produce more code in a given timeframe, aid coders with functions they may not know can be used and debug. [4]

Secondly, the use of syntax highlighting can provide visual cues of what specific keywords are and what they may directly mean to your code. Having this syntax highlighted allows users to quickly pick up larger files of code as it breaks up the elements in the files. Clicking on specific keywords will also highlight them to allow developers to see where they are used in the rest of the code. This is vital in Unity as gameObjects are constantly used and repeatedly called on. The integration of VS with unity allows for little to no setup. Empty scripts can be attached to any gameObject and code can begin from there. This is also possible the other way around. Saving scripts automatically update the Unity framework with the new developments and there is no need to update the code through a compiler. VS also has extensions available for direct GitHub use. With this plugin, the GitHub repository can directly be accessed from VS and branches, pull and push requests and commits can be handled directly from the IDE.



```
GameController scoreCounter;
private float scoreAddAmount = 10;

Unity Message | 0 references
public void Start()
{
    currentZombieHealth = maxZombieHealth;

    agent = GetComponent<NavMeshAgent>();
    wanderPoint = wanderZombie();
    animator = GetComponentInChildren<Animator>();
    target = GameObject.FindGameObjectWithTag("Player");
    fps = GameObject.FindGameObjectWithTag("Player");
    audioSource = GetComponent<AudioSource>();
    scoreCounter = GameObject.FindGameObjectWithTag("GameController").GetComponent<GameController>();
}
Unity Message | 0 references
void Update()
{
```

class UnityEngine.GameObject
Base class for all entities in Unity Scenes.

Figure 3. *The beginning of the AIController Script in the project. Showing VS syntax highlighting and IntelliScene referencing for efficient coding*

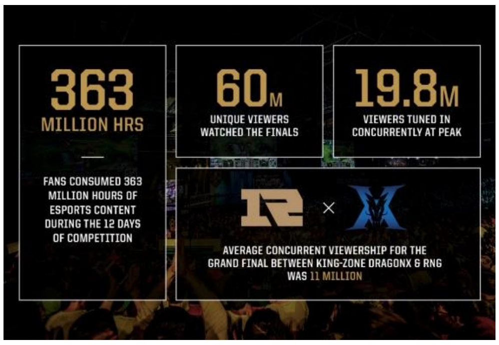
# Chapter 3: **Background: Understanding FP games**

The gaming industry is now larger than the cinema and music industries and is still developing. [5] With the number of games increasing day by day, it is important for the hardware that runs these games to increase their potential whenever possible. Gaming devices are consistently changing and with the new generation of console gaming in the PlayStation 5 and Xbox series X, hardware has adapted to become powerful enough to handle graphical powerhouse games. This background theory was important in my research for my project as it allowed me to familiarise myself with the lifecycle of a game, professional issues that can be found in games development as well as key features on how to develop and refactor games for better memory use.

## 3.1 Game culture

The culture of gaming is an international phenomenon, whilst one player may not be able to afford a game or console, they still have the opportunity to play on nine different generations of devices that can allow gameplay [6]. Moreover, watching gaming has become its own form of entertainment. Two of the top five channels on YouTube are video gaming related and 73% of gamers agree that they find entertainment in watching games. [7]

Games have had to adapt to allow content creators to want to play their games and drive more advertisement to their product, leading to more overall players. Games like call of duty have adapted to their typical friendly multiplayer gaming experience and developed an entire league and eSports tournament. It is suggested that by the end of 2021, eSports will have more concurrent viewers than the NFL in the United States [8]



363
MILLION HRS

FANS CONSUMED 363 MILLION HOURS OF ESPORTS CONTENT DURING THE 12 DAYS OF COMPETITION

60M
UNIQUE VIEWERS WATCHED THE FINALS

19.8M
VIEWERS TUNED IN CONCURRENTLY AT PEAK

AVERAGE CONCURRENT VIEWERSHIP FOR THE GRAND FINAL BETWEEN KING-ZONE DRAGONX & RNG WAS 11 MILLION

Figure 4. *League of legends 2018 eSports tournament, the largest viewership of any eSports event with over 60 million unique viewers at one point.*

It is important to find the balance of this in games development as there should be an aspect of casual play for beginners and those who do not find pleasure in a competitive scenario. When developing games, publishers and devs must forecast the outcome of the games lifecycle and whether it is a long-lasting competitive game or something that is expected to be replaced in a years' time.

## 3.2 The future of gaming

With the emergence of the 9th generation of gaming, graphical processing power and gaming mechanics have drastically changed, and a bright future is ahead. Games consoles have now adapted to use Solid State Drives (SSD) rather than hard drives. This change has allowed better handling of game save data and file sizes as well as loading assets. A direct example of this is Grand Theft Auto V, which takes 1:15s to load on 8th generation consoles and only 5 seconds on 9th generation consoles. [9]

Frameworks are consistently adapting to hardware changes too as it allows games developers more freedom and creativity in their games as the consoles can now manage the capacity. With the announcement of Unreal Engine 5, frameworks have more freedom than ever before to create amazing detail in games. Unreal now introduces Nanite virtualized micropolygon geometry that frees artists to create as much geometric detail as the eye can see. As well as this, Lumen has been introduced that can detect real time Sun raytracing to bounce reflections, shadows and lens glares whenever needed. This can all be edited in real time and supports development for gaming up to 8K HDR.

Animations and graphics have become easier to develop and recreate virtually thanks to the help of motion capture software. Motion capturing works by placing sensors all over an actor in a specialised room. The sensors are detected by capture software in real time and passed over to a virtual skeleton. These animations can then be passed over to graphic designers to create the body that will be attached to the skeleton. Motion capturing is used in movies with CGI as well as gaming for cut scenes and player triggered animations, they have allowed a breakthrough of realism including mobile applications that allow you to take a picture of your face and transfer it into a game as motion and face recognition has that much customisability now. (See figure 5)

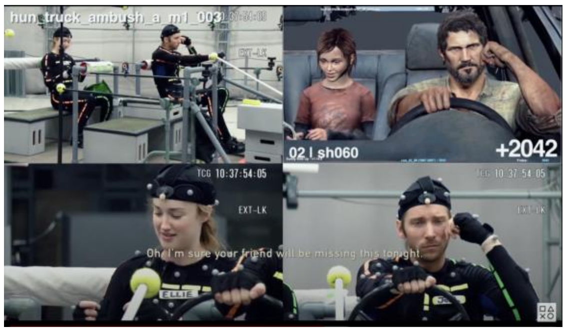

hun_truck_ambush_a_m1_003
02 l sh060
+2042
TCG 10:37:54:05
EXT-LK
Oh, I'm sure your friend will be missing this tonight.
ELLIE

Figure 5. *Motion tracking in PlayStation title, the last of us, with a scene to scene reference [10].*

## 3.3 Games Development

There are aspects to games development that need to be imported by developers themselves and other tools and features that are provided by the development framework. Unity has an extensive library of easy to use tools that create better depth and logic in games. Below are a list of tools and strategies that I researched for my game and the cost of including them into my game. The way they are implemented can be found in chapter 5 of the report.

### 3.3.1 Scene Management & Settings Manipulation

Games will have multiple different scenes that need to be navigable and work together to create a seamless experience. When a project has multiple scenes, they are held in the build settings of the Unity framework. Each scene has its own unique identifier and can be called by script and placed on multiple different game objects. Unity uses its own library called UnityEngine.SceneManagement, which allows scenes to be called on when specific action takes place. [11] There is little to no cost of using scene management in your project as it is recognised as a key feature to include for most games. Placing everything that a game needs on one scene can lead to compiling errors and memory not being allocated correctly. Instead, when a scene is loaded, the other scenes are destroyed.

Flash and loading screens are used in many games to give time for the scene to render all its assets. The stronger the hardware the game is running on, the faster the assets load. By having a flash screen, developers are able to advertise their game and company and concurrently let the game build all the assets it needs. Unity provide a default flash screen on build that you can add your own logos to.

Many games have ways to control settings options in game. To do this, Unity allows manipulation of settings in game using the default library and other scriptable objects. Settings scripts can be placed onto gameObjects in a scene to use as controllers. Dropdowns can be used to pick a graphic option and sliders for volume settings. To access these correctly, the scripts must adhere to the same constraints that the settings have in the framework editor. For example, Unity has a sound range from -80 to 0 and therefore the slider must have the same range. By exposing settings parameters, they can be used in script. Examples of how they are used in my project can be seen in chapter 5. [12]

### 3.3.2 gameObject properties

There is endless possibilities on what can be added to gameObjects. Depending on the use of the object it can be for handling logic in the game or can be a holder for logic that the player will directly interact with. Logic that is concurrent across a scene and doesn't require a trigger is typically placed in empty gameObjects. HUD and dynamic variables can be placed into canvases that act as an extra layer on top of the main camera of the game.

Useable assets is where customisability can reach its full potential. Assets can be acquired from the Unity framework, asset store and 3rd parties. Logic can be applied to any asset by connecting it to a C# script, animation controller or unity tools. 3D gameObjects can have custom materials that are created by graphics designers applied to them to create a more realistic feel to the object.

Any gameObject that may be repeated in the scene as a clone or may be used in multiple scenes can be packaged as a prefab. Prefabs are a Unity tool that take all the gameObjects components, properties and values and make them reusable. Common examples are environmental assets, AI, and Non-player characters. [13] gameObjects can be converted into a prefab by being dragged into the project files. From here the gameObject can be deleted from the scene and can come back whenever the prefab is either dragged into the scene or called using script. Prefabs can be instantiated with its own properties if that is what the developer is after. The cost of using prefabs is that they will not understand real time changes to objects and their positions unless this is scripted to the prefab.

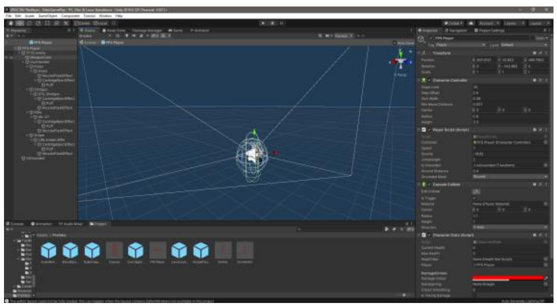


Figure 6. *FPS Player prefab that is called on scene load and removed when player dies.*

### 3.3.3 Animations

Animations are a straightforward way to add life to a game. Animations can be added to any gameObject, character or property and is handled through an animation controller. Animating AI and players can simply be done by setting key frames at timed intervals and changing the scene to match the animation, or complex animations for characters with skeletons which use the same logic but allows control per limb, leading to extensive animation possibilities. These can then be passed into the animation controller that creates links between transitions and provides a parameter and variable (int, bool, trigger) that can be called in script when the animation is required to take place. [14] The cost of animating can lead to needing multiple changes as the animation may not be suitable for a specific environment leading to objects sinking or floating on terrains.

### 3.3.4 Raycasts, Colliders and Navigation

Raycasts are a Unity physics tool that simply shoots out an invisible ray from the player camera (which is the origin point) and takes the forward direction. The ray will continue to travel until it has reached an impact point which in our case is either the zombie AI or environmental areas. Physics.Raycast is a library in Unity that provides endless possibilities of outputting raycasts (see figure 7). Layer masks can be placed on gameObjects to make sure only the objects that are wanted to be found in raycast scripting are seen. [15]

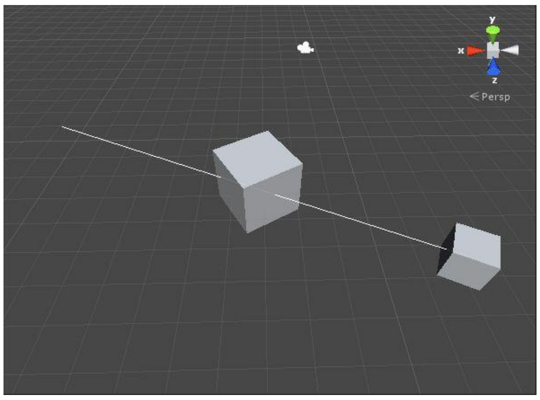


Figure 7. *An example of a raycast, from the cube on the right which is the origin and the impact area which is the cube in the middle.*

Colliders are a Unity tool that can be applied to gameObjects to create realistic collision theory. When two colliders interact with one another, an OnCollision event takes place. This only occurs so long that one of the gameObjects have a rigidbody attached to them. Colliders can be placed on objects in a Box, cylinder, or sphere shape. Mesh colliders are also available however, they take more system memory and can cause jittering as the detail of the mesh may be too accurate to recognise. Instead, developers apply primitive colliders to sections of gameObjects for better performance. When the collision takes place, script can determine the outcome of what you would like in the scene, for example health deletion or impact force being applied. [16]

Navigation Mesh is simply baking a path for AI that is not being controlled to walk across. By applying a Nav Mesh onto the terrain, developers can define what surfaces are walkable for the gameObjects as well as finding the shortest path to a specific goal. By adding layers onto all the gameObjects, you can easily control what the AI should deem as an obstacle and avoid. By baking onto the terrain, NavMesh allows developers to set variables that gameObjects should adhere too. Agents can take a humanoid characteristic which can limit their height off the NavMesh (keep agents stuck to the floor if wanted) as well as the level of slope they can walk across. [17]

### 3.3.5 Camera and Lighting

In a default scene, the only two things that are present are a Main camera and directional lighting. These can both be taken advantage of to create more realism in the game as well as correct camera placement for what the player will see. Unity now uses progressive lightmapping which can simulate real life lighting using direct and indirect illumination. gameObjects also have their own properties of emissive lighting and these are reflected amongst the main camera of the game, wherever the camera moves, light will gradually bounce off objects to reach the camera and simulate real lighting. [18] Many large production games using baking and shader raytracing to simulate real world reflections and refractions however this is very cost effective on the games overall performance as storing multiple lightmaps for the camera to recognise light changes in the terrain is very large. Many games provide the option for better lighting properties and lower frame rates or vice versa to let the player adapt to what they prefer. [19]

The camera is the main component that displays the items in the scene to the player when they are in the game. A camera must be present to render scene objects, else there is nothing to show in the game. The camera acts just like any other gameObject and can be exposed to manipulation, being a child of another game object or react in some way to a script. Positioning the camera can simply be done by dragging it around in the scene. To begin with the main camera is static but if the camera was a child of an object that moves in the scene, it will follow it around. Multiple cameras can be added which are appended to specific gameObjects however this can also lead to more CPU cost as the game is rendering objects twice from two different cameras, although this is a small cost it is something that must be taken into consideration if there are multiple objects and cameras.

# Chapter 4: **Topics and Simple examples of Games Programming.**

Software engineering is vital aspect of game design. Whether that is done through the Unity UI or developing scripts, the key fundamentals of games and anything beyond that will involve some sort of engineering. During the course of the academic year I developed multiple proofs of concepts that showcased my knowledge in gameObject scripting and linked directly into a prototype that demonstrated these skills. These proof of concepts were then restructured with new assets, UI management and better code for the final product. Mentioned below are some of the key concepts I have created highlighting their importance in my game during prototyping and final development, alongside gaming as a whole. Interesting code is also detailed to show how it made my game unique and improve the quality of its development.

## 4.1 Movement & Gravity

Movement can be seen in a majority of games, where characters in a game will always need to move to a specific point, whether that's through a keyboard, mouse click or other user input depending on device. In Unity3D, movement can be implemented through C# scripts that involve vectors with components and gravity can be simulated directly with rigid body components assets.

### 4.1.1 Player movement

WASD keys are the main movement input for common of PC games. There is a prefab in Unity that allows WASD keys to activate as the movement controller. As a result, a short method needs to be created to trigger this to be used and linked to the Player gameObject.

By using the horizontal and vertical axes given by Unity, a Vector3 function can be created that moves the player to a newPosition in x time (velocity). The new position is then passed onto the character controller component (Player object) which sends the player to a new position. By making the variable speed a public variable (see figure 8), we can edit the speed of movement directly from the inspector. The code below instantiates how fast we get to a given goal when moving the player.

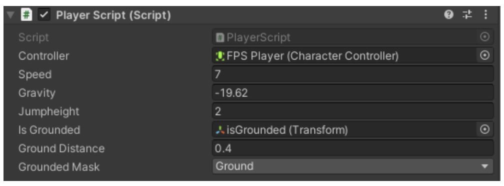
```
float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");

        Vector3 move = transform.right * x + transform.forward * z;

        controller.Move(move * speed * Time.deltaTime);

        if (Input.GetButtonDown("Jump") && grounded)
        {
            velocity.y = Mathf.Sqrt(jumpheight * -2f * gravity);
            FindObjectOfType<AudioManager>().Play("Jump");
        }

        velocity.y += gravity * Time.deltaTime;
                controller.Move(velocity * Time.deltaTime);
```


Player Script (Script)

| | |
|---|---|
| Script | PlayerScript |
| Controller | FPS Player (Character Controller) |
| Speed | 7 |
| Gravity | -19.62 |
| Jumpheight | 2 |
| Is Grounded | isGrounded (Transform) |
| Ground Distance | 0.4 |
| Grounded Mask | Ground |

Figure 8. *Public variable from the code above. Can be altered from the inspector for user preference rather than hard coded.*

### 4.1.2 Gravity

In games gravity is used to keep objects grounded on platforms rather than floating, this is easily done through the Unity rigidbody component which has a prebuilt gravity function. We can then manipulate rigidbody to our advantage by writing a C# jump script that allows the player gameObject to perform a jump action and provide more depth of movement in the game. Having a jumping function is important in the project as there will be multiple obstacles and enemies scattered around the environment that may block off the user, this option allows the player to find different ways around the level and escape the enemy hoard. [20]

The code below is a simple jump method that requires the user to input the spacebar key. It then uses a sphere gameObject linked to the player to check if the player is currently grounded, if not it allows x amount of force to be given to the gameObject which then creates a jump mechanism. The sphere also sets a limiter stating that if the player is not currently grounded, they are blocked from jumping again. This is to deter the use of spacebar to gain height without dropping back to the floor, which would break the realism of the game.

```
grounded    =    Physics.CheckSphere(isGrounded.position,    groundDistance,
groundedMask;

if (Input.GetButtonDown("Jump") && grounded)
        {
            velocity.y = Mathf.Sqrt(jumpheight * -2f * gravity);
            FindObjectOfType<AudioManager>().Play("Jump");
        }

        velocity.y += gravity * Time.deltaTime;
```

### 4.1.3 Mouse Movement

It is vital that in a first-person shooter game the player is allowed to move around and have a field of view of 360 degrees. This key feature of a first-person as not only does it direct the player camera to where the player itself wants to move, but also is the position of where the weapon will be facing. My game follows the same rule. At the beginning of the game, the camera centred was and locked to the scene in a first-person view. This was done by moving the camera component to become a child of the Player gameObject and then subsequently the main camera view for the player throughout the game. There is another camera for the weapons as this will avoid weapon clipping into objects and make for better looking gameplay (see figure 9)

To allow camera rotation through the mouse, a similar method to the player movement is used in the camera controller script. A vector that takes raw x and y-axis from the mouse and passes these onto another vector which creates a sensitivity variable was created. As the code below states, by transforming the local rotation of the player the camera is rotated instead of the player. Smoothing and sensitivity are both public attributes that can be altered in the Unity inspector and in the future of this game can also be implemented into the settings scene to change in real time whilst playing.

A simple cursor lock function was also placed on the camera controller to state then when the user is in play mode, the cursor is locked to the centre of the screen and is not visible. This is important in first person shooters as the crosshair is meant to be the focal point of the screen and the aid for the user to shoot. Having a cursor would cause confusion and distraction. The cursor is then only visible when the user is navigating around the project through menus or when the game has been paused during the play scene. Code for the cursor lock and camera movement can be seen bellow.

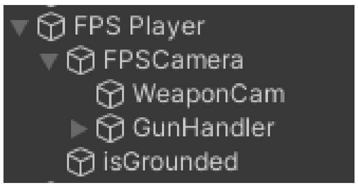


Figure 9. *Unity 3D hierarchy showing the link between the camera and player as a child and therefore wherever the player is in the terrain the camera will follow.*

```
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;

    void Update()
    {
        CursorState();

        float  mouseX  =  Input.GetAxis("Mouse  X")  *  mouseSensitivity  *
Time.deltaTime;
        float  mouseY  =  Input.GetAxis("Mouse  Y")  *  mouseSensitivity  *
Time.deltaTime;

        xRotation -= mouseY;
        xRotation = Mathf.Clamp(xRotation, -90f, 90f);

        transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
        player.Rotate(Vector3.up * mouseX);
    }
```

## 4.2 Enemy AI Behaviour

The project relies on enemies to be present and performing some form of action for there to be interaction with the player. Enemies are the key gameObject for the player to interact with and have multiple classes that bind them with unique characteristics. Below are the classes that the AI enemies either have a dependency on or directly interact with, as well as their purpose and some code displaying the importance of the script.

### 4.2.1 AI Spawning and movement

As the game state starts, the environment is expected to begin spawning enemies. Due to the endless survival-based nature of the game, it is important to form a unique experience for the player and create replay ability rather than repetition. This is done by creating a script that spawns enemies randomly across the spawn points in the environment, players are unable to guess AI spawn location and times, adding a level of difficulty to the game. The code below shows a simple script that takes an array of spawn points that are simple gameObjects placed around the map, when triggered, prefabs of the zombie gameObject will be spawned into the terrain and will begin to either wander or make their way to the player to attack.[21]

```
    public GameObject[] spawners;
    public GameObject zombie;

    private void Start()
    {
        spawners = new GameObject[5];

        for (int i = 0; i < spawners.Length; i++)
        {
            spawners[i] = transform.GetChild(i).gameObject;
        }
    }

private void SpawnEnemy()
    {
        int spawnerID = Random.Range(0, spawners.Length);
        Instantiate(zombie,          spawners[spawnerID].transform.position,
spawners[spawnerID].transform.rotation);
    }
```

A simple wave system was added to the game that determines when more zombies should be spawned into the terrain. This was created by developing a NextWave method that is called whenever the zombies that are currently in the scene have all been eliminated. When this has occurred, the next wave will begin and will spawn an extra two zombies. This increment of two will continue as the waves go on and therefore the longer the player survives in the game the harder it will get. In the future of the project, the wave system can be adapted to add more enemies when the score reaches a certain number or have a link to zombie or player health. A snippet of the script below shows how zombies are added. The zombies killed variable is accessed by the AI script which can pass the value of present zombies back to the spawner.[22]

```
private void Update()
    {
        if (zombiesKilled >= spawnAmount)
        {
            NextWave();
        }
    }

public void NextWave()
    {
        waveNumber++;
        spawnAmount += 2;
        zombiesKilled = 0;

        for (int i = 0; i < spawnAmount; i++)
        {
            SpawnEnemy();
        }

  }
```

Enemies should also use scripting to make their way towards the player, this is because I want the project to have a hoard feel to it as zombies are theorised to act in. By setting the player as the goal for the AI to reach, the zombies can have different states they act in. Wander, chase and attack. When spawned in the zombies are in a wander state. Here they move across waypoints that are directed to them using a Nav Mesh agent (more on this in chapter 4) until they are in a radius of the player that changes their state. When in the correct radius they will consistently chase the player without loosing him and when they are in an even smaller radius they will attempt to attack. This was completed using multiple Vector 3 distance and angle functions. All of the functionality of AI is handled under one script. Below are the functions that changes the state of the enemy.[23]

The method for the zombies wandering a terrain:

```
public Vector3 wanderZombie ()
    {
        Vector3 randomPoint = (Random.insideUnitSphere * wanderRadius) +
transform.position;
        NavMeshHit navHit;
        NavMesh.SamplePosition(randomPoint, out navHit, wanderRadius, -1);
        return    new    Vector3(navHit.position.x,    navHit.position.y,
navHit.position.z);
    }
```

Switching between finding a player and chasing a player is held in the update function as we want to check for this on every frame of the game as the player and zombies are consistently moving. The code bellow states that if the zombie is aware of the player (in its radius) then set its destination to the player, increase the speed into a chase and attempt to attack. Otherwise continue to try and find the player using the findPlayer function and keep the speed and logic in wander state.

```
        if (zombieAware)
        {
            agent.SetDestination(fps.transform.position);
            agent.speed = chaseSpeed;
            animator.SetBool("Chase", true);
            Attack();
        }
        else
        {
            findPlayer();
            Wander();
            animator.SetBool("Chase", false);
            animator.SetBool("Attack", false);
            agent.speed = wanderSpeed;
        }
    }

    public void findPlayer()
    {
        if                                (Vector3.Angle(Vector3.forward,
transform.InverseTransformPoint(fps.transform.position)) < fov /2f)
        {
            if(Vector3.Distance(fps.transform.position,  transform.position)
< fovDistance)
            {
                OnAware();
            }
        }
        else          if          (Vector3.Distance(fps.transform.position,
transform.position) < 20)
        {
            OnAware();
        }
    }
```

### 4.2.2 AI Enemy attack

AI attacks can be split up into two distinct formats, the player attacking AI and the AI attacking the player. Both directly link to the AI and player gameObject and use either raycasts or collisions to create the attack.

The zombie attack is from an enemy to the player. This is scripted very similarly to the player attacking the zombie, however instead of using raycasts the project uses collisions [24]. As the enemy is moving towards us and eventually colliding with the player, we can script this collision as the player taking damage. This uses the logic that if a zombie was to hit or swipe at the player then they would lose health. In the prototype, it was hard to show this as the enemies were a simple cylinder however when developing the final build using proper assets, I was able create the collision between the zombie hand and the player rather than the entire enemy object. This gives more time for the player to understand when an enemy is due to attack so they can evade, this also adds to the key objective of realism in this project. There is also a delay in how fast the same zombie can hit you as they must reset their attack stance. A two second delay is added to the attack however this is per zombie, so whilst one zombie is on an attack cooldown the other may still attempt to attack. The code and figure 10 bellow show how the zombie was animated as well as how an attack takes place.[25]

```
public void Attack()
    {
        float     distance     =     Vector3.Distance(fps.transform.position,
transform.position);
        if (distance < 2f)
        {
            agent.speed = 0;
            animator.SetBool("Attack", true);

            if (Time.time - lastZombieAttack >= attackCooldown)
            {
                lastZombieAttack = Time.time;
                target.GetComponent<PlayerStats>().takeDamage(damage);
                target.GetComponent<PlayerStats>().realtimeHealth();
                audioSource.PlayOneShot(zombieHit);
            }
        }
        else
        {
            animator.SetBool("Attack", false);
        }
    }
```

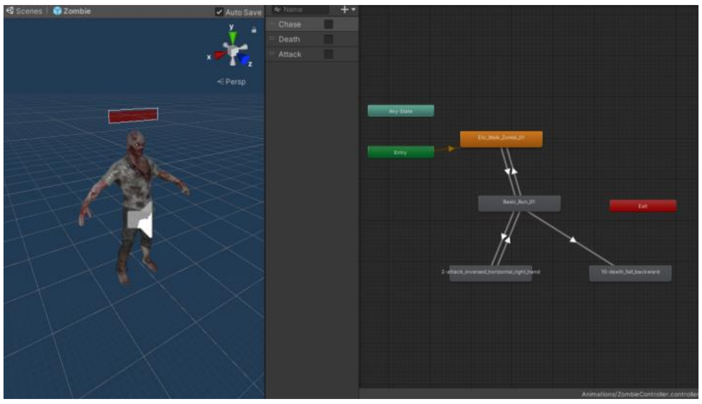


Figure 10. *The zombie prefab with its attached animation controller. The parameters bools are passed into script to switch state.*

### 4.2.3 Attacking the enemy

When the player intends to attack the enemy, raycasts are used. This is due to the distance between the player and the zombie. Players will aim to attack the zombie as if they were shooting from a distance and therefore raycasts are used. When a raycast hits the enemy, which can be recognised by the script using Unity tags, the zombie will lose an amount of its current health. The damage the enemy takes is handled by the same gun script that holds the logic of shooting the enemy. This script also holds the reload of each gun and as each gun has its own properties, variables are all public so damage, shot range, reload time and the magazine size can all be defined.

All the guns that are available in the game are a child of the gunHandler. As there is more than one weapon and we want the ability to switch between them, the handler provides seamless switching between the weapons through either a scroll or number input. The code below is repeated with < 0f to allow scrolling both ways as well as other if statements for numeric key inputs. [26]

```
if (Input.GetAxis("Mouse ScrollWheel") > 0f)
        {
            if (currentWeapon >= transform.childCount - 1)
                currentWeapon = 0;
            else
                currentWeapon++;
```

The gun properties scripts are not appended to the gunHandler as each gun has its own properties and therefore must be able to exclusively access the gun properties script. Two scripts were made for this, one that allows automatic shooting by taking Input.getButton and single fire weapons take Input.getButtonDown. Both scripts hold the same method for reloading and shooting. Players are simply only able to shoot if they have ammunition and as they shoot, their ammunition depletes. When ammuntuion hits 0 the player must reload by pressing "R" to allow the shoot function again.

The shoot function itself takes a raycast and looks for the target gameObject. If the target is visible in the raycast then when shooting, blood animations will appear as well as the zombie healthbar deplete. Shooting will also trigger different particle systems to make the game look more realistic as well as a camera shake as a realistic recoil graphic. These will be spoken more on in the next chapter. Bellow you can see the code for the shooting function as well as the change of the guns and the hierarchy with the handler. [15]

```
void Shoot()
    {
        audioSource.PlayOneShot(gunSound);
        muzzleFlash.Play();
        bulletEject.Play();
        currentAmmo--;

        RaycastHit hit;
        if                    (Physics.Raycast(fpsCamera.transform.position,
fpsCamera.transform.forward, out hit, range))
        {
            Debug.Log(hit.transform.name);

            AIBehaviour target = hit.transform.GetComponent<AIBehaviour>();
            if (target != null)
            {
                target.shootZombie(damage);
                GameObject bloodShot = Instantiate(bloodEffect, hit.point,
Quaternion.LookRotation(hit.normal));
                Destroy(bloodShot, 2f);
            }

            GameObject   impact   =   Instantiate(impactEffect,   hit.point,
Quaternion.LookRotation(hit.normal));
            Destroy(impact, 2f);
        }

        if (currentAmmo <= 0)
        {
            audioSource.PlayOneShot(dryClip);
```

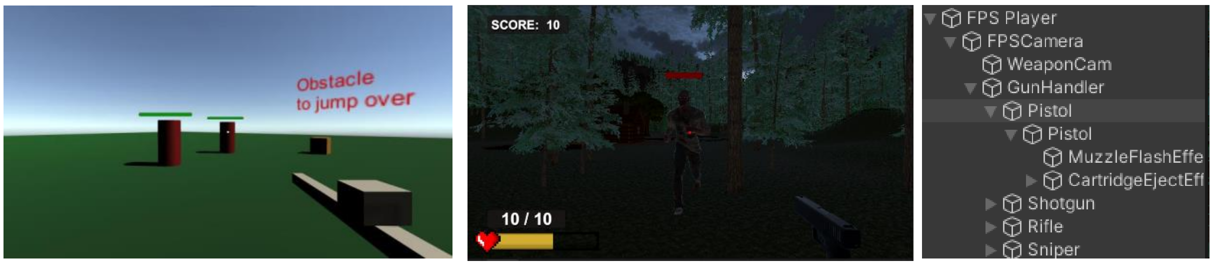


Figure 11. *In game view of the player camera facing an enemy and their health. A side by side view of the prototype and the final project. An image of the hierarchy is also present to show how guns are a child of the gun handler which is a child of the main camera which is a child of the player.*

## 4.3 Environment Building

There are multiple different ways to create an environment with Unity. At the start, an empty world space is the only thing present however there is unlimited possibilities on adding to this. The simplest options are to use 3d gameObjects from unity like cubes and morph them into platforms, another is to use the unity asset store and build on what someone else has previously made. Finally, by using the Unity terrain building system, a vast environment can be created that holds player and AI navigation. As I wanted to add more depth and display my knowledge of Unity, I built my own terrain space using simple assets.

### 4.3.1 Terrain structuring

To begin, Unity provides an empty terrain canvas that can be accessed as a gameObject. From here, I was able to manipulate the layout of the terrain using the implemented terrain tools that Unity offer. [27] Terrain tools that I used were raising and lowering the height of the floor for a hill effect for my forest. Secondly, I also used paint feature. This allowed me to paint the floor of the terrain with different material assets for a grass, snow, and dirt effect. This was layered with the tree painter and detail painter tool. By implementing assets into both of these too, I was able to create a well-structured forest environment for my game. All assets were acquired from the unity asset store which are linked in the appendix.

Directional lighting is a vital section of terrain design. It can make a game from looking basic to a better developed project. The skybox in the game was replaced with a night-time image from the asset store. This gave a more realistic representation of a dark eerie forest, which is what I was attempting to recreate. In addition to the skybox, I changed the positioning and colour concentration of the directional lighting to make the best possible shadow effects in the dark. Using Unity post processing tools, I was able to add layers of fog effects and saturation for the best possible look.

Finally, adding more gameObjects from the asset store like cabins, boulders and crates, makes more obstacles for the player to traverse across and brings more life into the terrain. Having a bland terrain can lead to players finding the game more repetitive as they see the same layout every time they play. When adding these objects, it is vital to pass information onto the AI that these are obstacles. This is simply done by including a Nav mesh agent on both the environment and the AI. By applying a navigation bake on the terrain, the nav mesh agent finds all the possible obstacles for the AI and passes this information onto the gameObject so it can find other paths. Bellow you can see the bake of the terrain (see figure 12)

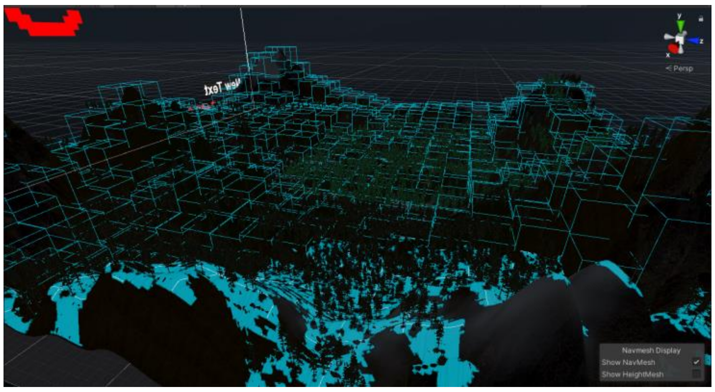


Figure 12. *The terrain the game takes place on. When scrolling into the environment, the blue flooring is where gameObjects with a nav mesh agent can traverse across, creating paths for the AI.*

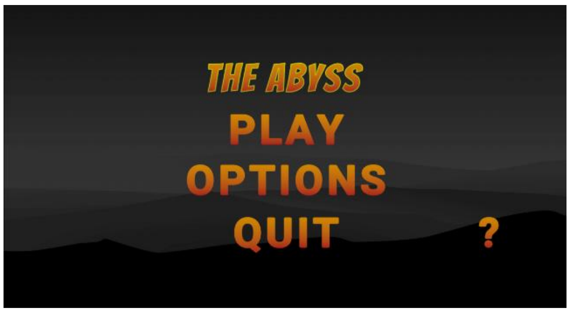
## 4.4 Designing a functional UI

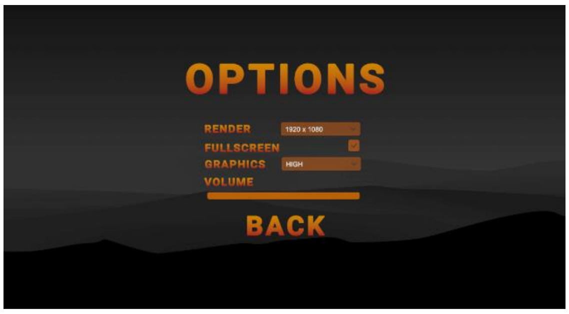
The importance of a UI to navigate through a game as well as display information is a key concept. Through UI & GUI concepts, users can enter the game scene as well as quit the game, enter options menus to adjust controls, and see vital stats through a HUD (Heads up display) when in the game scene. Below, these specific UI models are described in further detail.

### 4.4.1 Graphical User Interface (GUI – Menus)

The Main menu was created to navigate across the games scenes. I began to use some assets such as textMeshPro [27] text and buttons for a nicer appearance to the standard Unity text. Ideas from the prototype were directly included into the main project. All scenes that require text and buttons use textMeshPro keeping consistency across menus. Navigation is also handled by using scene management [28] and the main structure can create a build structure seen below. To traverse across menus, scripts were written and initialised to onClick functions for buttons to create an action. Audio listeners are also placed across all scenes to allow sound to play, this can be managed from the options menu and controlled to the users preference.

The options menu is activated from onClick as a Boolean function that states when options is clicked, main menu is disabled. The same is used for the options menu whilst in the game. The same themes and fonts are used across the menus to follow Nielsen's heuristics and keeping consistency across the UI.[29] A simple timer was set that states if the user is on the main menu and has been inactive for 20 seconds, then transition to the Demo scene. The demo scene plays a video automatically on loop until any key is pressed. This is a pleasant addition into the game as it allows a sample of the game to be seen in the background showcasing the gameplay. Smooth fading transitions are managed automatically through Unity keeping an elegancy across scene switching. Below is a screenshot of the main menu as well as the options menu, displaying consistent theming and practical structuring (see figure 13).





RENDER 1920 x 1080
FULLSCREEN
GRAPHICS HIGH
VOLUME

Figure 13. *Screenshots of the Main meu and the Options menu which are navigable using buttons.*

The final scene to be seen after playing the game should be the game over scene, this is triggered through a simple script that states when player health is down to zero, switch to the game over scene. This pauses the game state and takes the current score at zero health and showcases it on the game over screen. In addition to this, buttons to replay the game, return to the home screen and quit the game are present. This ease of navigation is vital as all human controlled interfaces (HCI's) should follow a user-friendly structure.

The pause screen is an in-game function that is triggered with the escape key. The pause menu hold the same information as the main menu, with access to the options menu, how to play and quitting the game. Instead of pressing play game again, this is switched with resuming the game again. More of the in-game user interface is mentioned bellow.

### 4.4.2 In-game User interfaces (HUD)

A display canvas or a Heads-Up Display (HUD), is a useful display that nearly every game has implemented. The HUD is where key information that needs to constantly be displayed to the user lies. This includes the player health, ammunition counter and score. This can very simply be created by adding an empty gameObject with static properties where we expect dynamic changes and script these properties.

Firstly, the player and zombie health bars are simple sliders that are taken from Unity. A gradient is added to both sliders that represent the health and whenever either the player or zombie is attacked, a damage value is subtracted from the slider and displayed on the HUD for the user to see. The health was represented by creating a public healthbar script that can be called by the AI script and the player script to display their maximum and current health values.

Secondly, a similar implementation is used for the score counter and ammunition. Instead of sliders, these are displayed with text gameObjects. When in editor mode the text is static however a simple method can be added that calls a public text variable that displays current ammunition or current score. As these update in the game they are also updating in the scripts and therefore can be displayed on the screen. The code below shows how ammunition can be seen in the game.

```
ammoCounter.text = currentAmmo.ToString() + " / " + maxAmmo.ToString();
```

The final key feature of the HUD is the countdown. A simple countdown animation was created that takes a three second countdown. This animation was then placed on the centre of the HUD and a simple script implemented the functions. The script states that when the scene is loaded stop time for x seconds, show the animation and then start the game timer. It is important to add this break as instantaneously entering the game from the main menu gives no time for the player to prepare and assets may not be completely loaded in. This can be seen bellow.

```
void Start()
    {
        StartCoroutine("Counter");
    }
IEnumerator Counter()
    {
        Time.timeScale = 0;
        float pauseTime = Time.realtimeSinceStartup + 5f;
        while (Time.realtimeSinceStartup < pauseTime)
            yield return 0;
        countDown.gameObject.SetActive(false);
        Time.timeScale = 1;
    }
```

The pause screen is a simple canvas that holds buttons and properties that can only be accessed when triggered. The pause screen is simply triggered when the user presses the escape key. This subsequently pauses the game timer, displays the pause canvas and the mouse cursor for the user to be able to navigate. It uses the same logic as the Main Menu, using onClick features of the button to navigate. With a simple if statement in the update method checking if the game is currently paused or not.

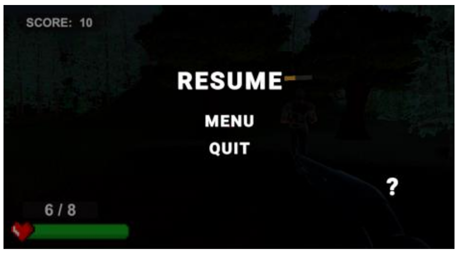


Figure 14. *The pause screen, still displaying the HUD however overlapping the contents and pausing the game.*

### 4.4.3 Score Management

Games have their own individual objectives whether that be getting the high score or to finish the storyline. There is typically an end to a game and a way to review your performance amongst others. This form of score management is a great way to add more interactivity, competition and objectives to a game and is an important aspect of my project. Below is the reasoning of why there is a score system in my game and how it is handled.

As the game is an endless survival and there is no finish to the game other than when player dies, it is important to provide some feedback to the player to give them a reason to retry. In my project the aim is to have users play more to get a better score. The score is held by killing zombies and surviving as long as possible. The game only allows one life for the player which has no health regeneration, this encourages users to play smarter, use their resources and strategize. 10 points are added per zombie kill and this is simply held in a Game Controller which makes a score counter holding the current score. This controller is called to the AI script and placed under the zombie die function to add score.

```
public void AddScore(float ammount)
    {
        currentScore += ammount;
        UpdateScoreUI();
    }
```

In future development of the project, I will attempt to host player scores on a database leader board online, this will add significant depth to the game as it will create a community who all play the game. Scores as of now are updated after every game but there is a possibility of adding a local device high score which can be adapted too in the near future.

# Chapter 5: **The Software Engineering process – explaining an example game.**

## 5.1 Use of Unity tools

The research from the background theory came in extremely helpful whilst developing the game. All of the research was used into the project to provide a high-quality piece of work. This section refers back to the background theory of Unity tools and how they were used in my game.

### 5.1.1 Prefabs

Prefabs were used whenever a clone of a gameObject was needed or when the gameObject had to be present in different scenes. A prefab for an Audio Manager was created that stores multiple sound files in an array that can simply be called in scripts when needed to play. Prefabs were also used for the zombie AI as these are called on countlessly during the gameplay. Particle systems and graphics for guns were also stored as prefabs as they were used for each gun in the game.

### 5.1.2 Animations

Animations were used to provide realism and depth in the game. The clearest use of animations and an animation controller was the zombie AI. The AI uses the controller to animate between states depending on the distance between itself and the player. Animations were also used for the countdown timer at the start of each playthrough and finally as a simple reload technique. These animations paired with fluid movement and audio led to a much more dynamic game rather than static cylinders which is what was in the first prototype builds of the game.

### 5.1.3 Raycasts, Colliders and NavMesh

Raycasts were used in the game whenever the player was shooting in the main game scene. Layer masks were used to provide logic to the raycast for the ray to know what it is shooting at. Particle systems were implemented with the raycast to display blood leaving enemies when shot or dust rising when shooting at the floor. It is the main physics object that allows the enemy to take damage and the guns to work.

Colliders are used all across the terrain. Tree colliders are placed to deter the player and AI to be able to walk through trees. This was also placed on all other static objects in the terrain. Colliders were also placed on the zombie hand as this is what is attacking the player and a collider was also fixed on the FPS player itself as a away of recognising the player taking damage from the enemy.

A NavMesh was simply used to bake the terrain and give logic to the AI to find the shortest possible path to the player to inflict damage. This simple yet powerful technique allowed the zombie to avoid any obstacles and still return to the player.

### 5.1.4 Camera and lighting

Lighting was used to create a dark moonlight effect across the terrain and the skybox was altered using Unity assets to support this. Post processing tools allowed lighting and shadows to be linear, creating more depth and fog was also implemented. Lighting was also used on the muzzle flash to replicate a quick fire-flash just like guns experience in real life.

The main camera was placed as a child of the FPS controller at eye height. By appending it to the player, the camera was able to dynamically move depending on user input (WASD Character placement and mouse movement). Another camera was placed for the weapons. This was to avoid clipping of weapon prefabs into environmental objects by using Layer masks and tags.

## 5.2 Version control structure

The project files were held in the GitHub repository and split up into branches for better file management. Whenever I was working on a new subtopic of code in the game, I created a new branch whilst bringing previous files from other branches across. This form of phased iteration allowed me to back up my work if any issues arise and helped me focus on each topic by scripting and developing functions for testing in the test suite scene and compatibility in the project as a whole.

New branches were made when there was certainty that a previous version of the project is functional and can begin to implement new features. These branches held the meta data for all of the gameObjects as well as the scripting and Unity tools linked to the objects. When development of a branch was nearing completion, I made sure that I was able to upload all the changes of my files into the branch and push it onto the GitHub repository. New branches took files from the current working branch as a form of phased development and merged into the master when a stable working version was ready to commit.

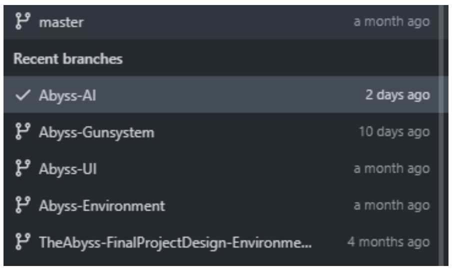
The branches that were made in my repository were as followed: (see figure 15)

- Environment – Creating the terrain with its gameObjects present as well as the FPS Player taking FP Camera controller properties to allow fluid movement across the terrain

- UI – A properly structured UI that allows navigation between scenes and triggers to occur with button presses. This includes the pause menu as well as the main and options menus.

- Gun system – The complete development of the weapon handler which allows the switching between weapons as well as giving each gun its own properties and appending it to graphics and other animations.

- AI – The final branch that holds all the required knowledge for the zombie AI, using Nav mesh tools, animation, and scripting features. This branch also holds the scripting for the HUD therefore logic for scores, player health and spawning is also present.



master a month ago
Recent branches
✓ Abyss-AI 2 days ago
Abyss-Gunsystem 10 days ago
Abyss-UI a month ago
Abyss-Environment a month ago
TheAbyss-FinalProjectDesign-Environme... 4 months ago

Figure 15. *The branches that are valuable to the final project. Other branches are made for prototype testing.*

## 5.3 UML Class Diagram

The diagram shown presents a UML (Unified Modelling Language) class design of the way the game is intended to play out. It implements the scenes, scripts and assets in the game and show how they independently or are directly linked to one another in the game through the Unity framework. (see Figure 16)

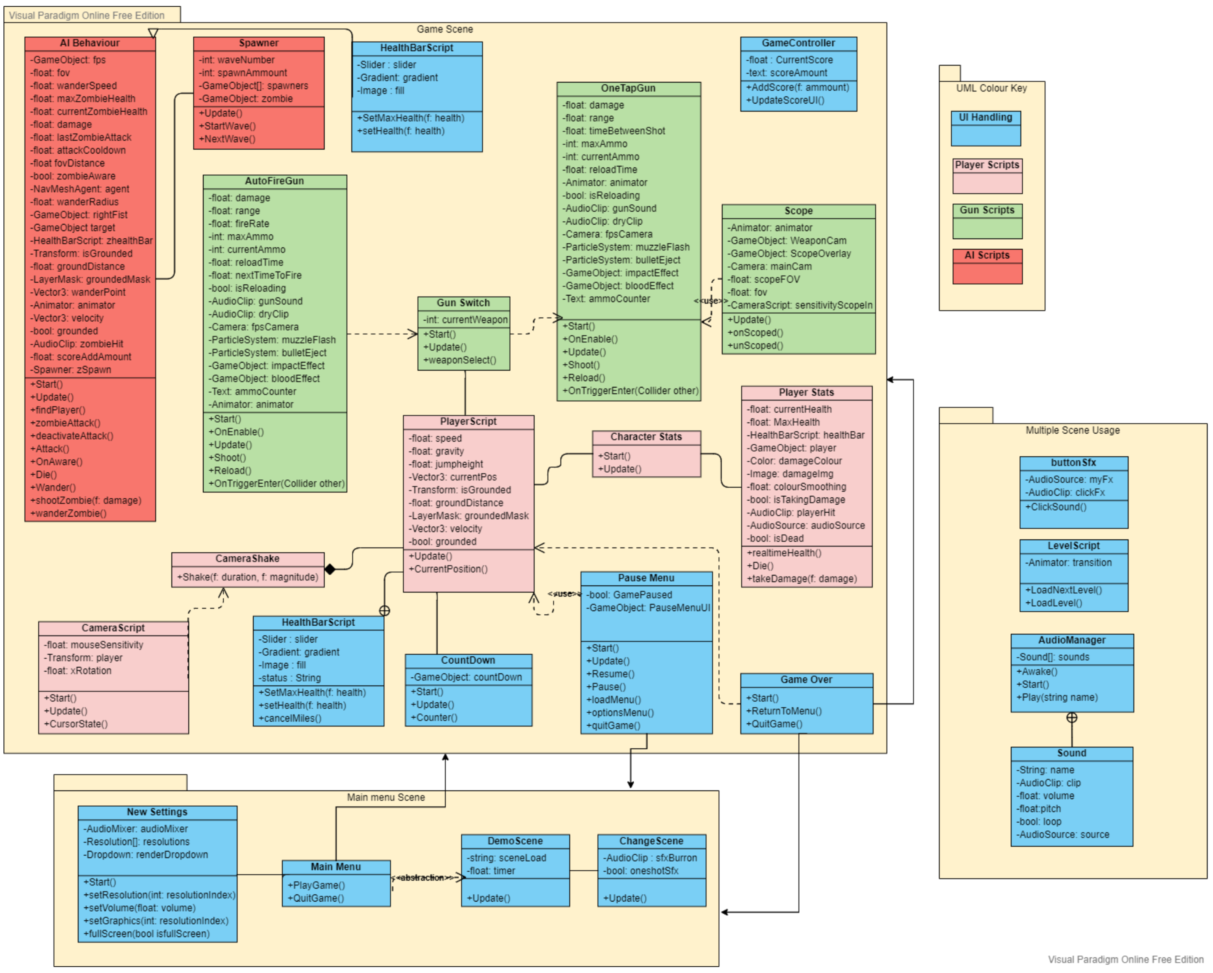


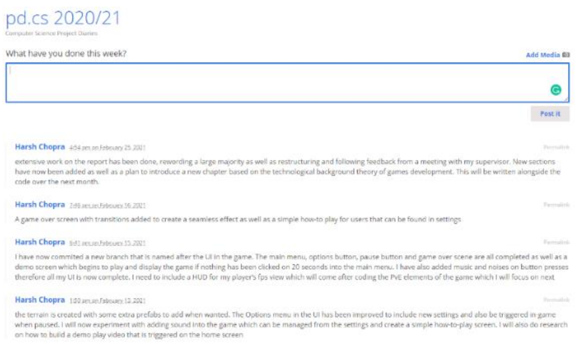
Figure 16. Unified Modelling Language *Class Diagram of the prototype First-person shooter Structure. Blue: UI class, Pink: Player classes, Green: Gun classes, Red: Enemy classes*

## 5.4 Project Communication – keeping track with a diary

A project diary is a written record of significant activities, events or processes that occur during the life of a project. The inclusion of a project diary in my project allowed me to keep my progress and insights noted, this was important as it allows a platform to reflect upon, see progress and plan for the future. As there are multiple branches and aspects in games development, holding notes in a diary can help developers keep track on what their thought processes were at the time and where to continue with work.[30]

The first instalment to my diary was after the first supervisor meeting, this helped me keep note on things discussed in the meeting and where I want to pick up from the next time I work on my project. Since this first instalment, I have added a post to my diary whenever there is a change in my game design, report or a significant event has taken place blocking me from progressing. The diary was beneficial to my project as I used it to keep myself updated with what branches I have been working on and the level of progress on a specific proof of concept. Subsequently, this aided me in saving time and knowing exactly where to continue my work as well as keeping my supervisor informed of my progress for discussion in future meetings.

An overall of the dairy and some key posts include, finding information on the psychology of gamers and making note of ideas in the diary to then develop on. Secondly, keeping note on player shooting not working and therefore working on another branch. Adding this to my diary let me and my supervisor stay informed as to why I stopped coding on that branch and began work on another. To summarise, the diary broke down report sections and proof of concepts during the prototype and actual development of the game. It allowed me to express any difficulties I have been going through with research or a lack of progress due to obstacles with life or other studies. Personally, I feel like a diary is vital in the life cycle of a project as it offers an insight into what works and what is causing obstacles, providing valuable information. On the other hand, as a diary can be quite informal, it can be difficult for other users to read and it may be difficult to keep up to date with diary posts.


pd.cs 2020/21
Computer Science Project Diaries
What have you done this week?
Add Media
Post it

**Harsh Chopra** 4:54 pm on February 25, 2021

extensive work on the report has been done, rewording a large majority as well as restructuring and following feedback from a meeting with my supervisor. New sections have now been added as well as a plan to introduce a new chapter based on the technological background theory of games development. This will be written alongside the code over the next month.

**Harsh Chopra** 2:46 am on February 16, 2021

A game over screen with transitions added to create a seamless effect as well as a simple how-to play for users that can be found in settings

**Harsh Chopra** 6:41 am on February 15, 2021

I have now commited a new branch that is named after the UI in the game. The main menu, options button, pause button and game over scene are all completed as well as a demo screen which begins to play and display the game if nothing has been clicked on 20 seconds into the main menu. I have also added music and noises on button presses therefore all my UI is now complete. I need to include a HUD for my player's fps view which will come after coding the PvE elements of the game which I will focus on next

**Harsh Chopra** 1:00 am on February 13, 2021

the terrain is created with some extra prefabs to add when wanted. The Options menu in the UI has been improved to include new settings and also be triggered in game when paused. I will now experiment with adding sound into the game which can be managed from the settings and create a simple how-to-play screen. I will also do research on how to build a demo play video that is triggered on the home screen

Figure 17. *Screenshot of project diary and how posts are added, the full diary can be seen in appendix of the report.*

## 5.5 Testing the example game

The importance of testing a game is as crucial as the code itself. If you develop anything that has not been well tested, it can lead to multiple bugs being found when publishing that can cause crashes, runtime failures and an unwanted experience for the User. I used an empty scene during development for user testing as well as allowing friends and family to test the game and provide feedback.

### 5.5.1 Testing Suite

A separate scene that is not included in the build properties was used as a tester scene. Regression testing was used as each gameObject was individually placed into the test scene to see its functionality without scripts or other gameObjects relying on it. This was successful as it helped find any compiling errors or Unity tool errors in early stages of development, making it easier to merge scenes and objects in the later stages. (see figure 18)

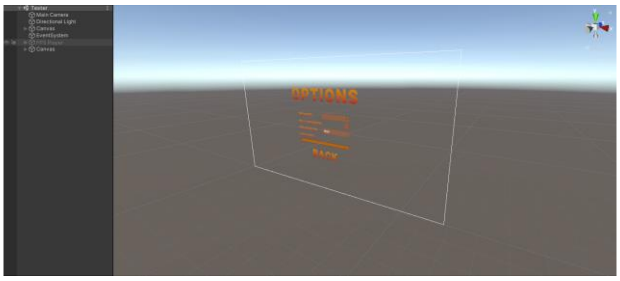


Figure 18. *The tester scene currently only holding the Main menu canvas to test on.*

### 5.5.2 Feedback testing

It was important for my game to be seen and tested by other users. The aim here was to allow selected users to play the game in early builds for feedback on the gameplay, UI, and ease of downloading. This covered Performance, compatibility, and functionality testing. Users were chosen from people with computer science knowledge to actively hunt for bugs and errors in the games like QA testers would. Compatibility was tested by providing build files to testers with different operating systems, CPU and RAM storage as this indicates that the game is able to run on a variety of devices. Finally, functionality was tested by asking users to play the game in real time with me as the developer taking notes. I watched the way the game was played differently and noted features that may be obvious to one player may not be clear to another. [31]

This led to further development of my game as feedback that was heard multiple times and good ideas that make the game more enjoyable was a big factor to the development of the game. Some users stated that the game over should not be a separate scene but more a canvas overlaying on the main game for quicker and smoother transitions. Furthermore, testers suggested that the terrain was extremely large for the size of the single player game, with this feedback I created walls around a specific area which is now seen as the main play area.

# Chapter 6: **Improving Game Design with Patterns**

Patterns are used to express gameplay in different methods. The way a game plays is through its design patterns leads to cleaner code and utilisation of memory. Design patterns for Unity3D are written in C# and are directly involved in gameObject assets. The three key features of design patterns of Unity belong in the fundamental design of the framework, the structure and mvc and the development processes. Unity encourages the use of design patterns in games development as it automatically includes it multiple times itself in assets such as prefabs. Below is a list of design patterns my final project uses alongside other patterns that can be found in the Unity framework.

## 6.1 Common Patterns

### 6.1.1 Command Pattern

The command pattern is a simple yet effective pattern where an object and its information are encapsulated into a trigger event for a later time. It allows users to rebind keys and undo previous movements. A queue of commands that takes mouse input and waits for new commands to add to the queue for the game object to move to can be set up using the command pattern. It is commonly used in prefabs that hold all the variables and are triggered on call, for example spawning zombies. Developers use command to trigger animations as well as calling prefabs onto the scene. The code below takes commands and adds them to a queue so all commands can be completed. [32]

```
private void ListenForCommands ()
{
if (Input.GetMouseButtonDown(0))
{
var ray = Camera.main.ScreenPointToRay(Input.mousePosition);
if (Physics.Raycast(ray, out var hitInfo))
{
MoveCommand moveCommand =     new
MoveCommand(hitInfo.point,_commands.Enqueue(moveCommand);
            }
      }
private void ProcessCommands()
}
```

### 6.1.2 Flyweight pattern

Often games use the same gameObject multiple times. The flyweight pattern allows the game to run smoother by utilising these repeat gameObjects. The code below shows a prototype where flyweight uses the same block object that is taking multiple colours using GetProperty and SetProperty to save memory in Unity. This is also used for environment building to save game memory. Thousands of trees across the terrain would take a large capacity of working memory however flyweight utilises these objects to create sustainable memory for other aspects of the game. This is also be used for enemies, as AI all perform similar actions and can be allocated to the same memory cache, flyweight can once again utilise the performance of the game. Developers can implement this into Unity for better memory allocation.

```
private void Awake()
{
renderer = GetComponent<Renderer>();
propBlock = new MaterialPropertyBlock();
}

private MaterialPropertyBlock propBlock;

void Update()
{
renderer.GetPropertyBlock(propBlock);
propBlock.SetColor("Color",GetRandomColor());
renderer.SetPropertyBlock(propBlock);
```

## 6.2 Behavioural Patterns

### 6.2.1 Type object

Type object is a clean and easy way to store multiple pieces of information in just two object classes instead of a cluster of multiple classes. In this project, it can be assumed that there will be a variety of enemies and guns used and instead of making a class for each enemy e.g. (zombie, crawler, far range enemy) type object can be used. Here I would have one class for an enemy and one for breed, in the breed all the types of enemies will be stored rather than having separate subclasses for them all. The same pattern would work for guns as each gun would store its own information (ammo, damage, range) however, all guns can be stored in a single gun controller class.

### 6.2.2 State pattern

The State pattern holds information on how a specific gameObject should act in its state. For example, how an enemy will act depending on if it was hit or stood too close to [33]. This is important as the object can have multiple states e.g. roaming enemy and attacking enemy so there must be a trigger in the object for it to switch. (see Figure 19)

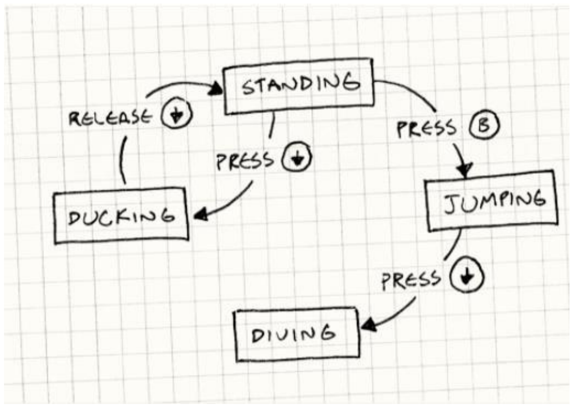


Figure 19. *A common use of the state design pattern, taking input from the user keyboard that changes the state of a specific gameObject*

## 6.3 Decoupling patterns

### 6.3.1 Component

The component is embedded into the Unity framework itself. The pattern builds the structure of objects and allow you to add additional functionality and features to those specific objects. This is a decoupling pattern as even though all of these components reside in the same object, they are all unique to each other and will therefore act separately. Changing a component should not change the actions of the other functions of the object. [34] By using the required input.update class shown below, functions can be separated and have a clear breakdown of what the objects need to do.

```
input_.update(*this);
```

### 6.3.2 Publish Subscribe

The publish subscribe pattern is where messages from publishers get sent to a channel, that then pushes these messages to the subscriber. In simple terms, when the user of the game publishes a message e.g. press the mouse button to shoot the enemy, this message is then sent to any subscribers of that message. Subscribers for shooting an enemy includes: the movement mechanics for the mouse button, ray cast features for shooting and the enemy class for shooting the enemy, as they are all members.

### 6.3.3 Game loop

Prebuilt into the Unity engine, the game loop plays a key factor in most games. The idea behind this pattern is that a user processes an input, this then updates the game in some situation. The update is then rendered and is looped back to the user's input. This needs to be used and tweaked carefully, as frames can be skipped or loaded too quickly which would lead to runtime errors in the game.

### 6.3.4 Update method

The update method works very well combined with the game loop. As the game is creating multiple AI objects at once that need to run simultaneously, it will use Unity's Update() method and each frame will update every object to see if there needs to be a change. This is also prebuilt into unity and can be seen in every script on first load. Whenever there is a potential change in the scene, it should be relayed into the update method so the scene can display the action in its next frame.

To conclude, there are many accommodating design patterns that I can and have used throughout the project. A lot of key patterns are implemented in the Unity framework, having these patterns leads to less coding, smart implementation and an overall cleaner look of the game making the user experience much friendlier.

# Chapter 7: **Professional issues - The psychology of gamers**

With a staggering 2.7 billion gamers worldwide as of 2020, the video game industry has made its mark with the effect it has on society. [35] The industry is a dynamic, rapidly developing ecosystem that consists of gamers, developers, publishers, marketing and many more businesses. Research has shown that although gaming is typically seen as a stress reliever, there is an equal number of negatives that can be associated too. In this chapter, we deep dive into the three key stakeholders of gaming ; (gamers, developers, publishers), and how specific genres of games can be wrongly mistaken.

## 7.1 Gamers

In my research on gaming [36], there was a clear trend that showed the majority of gamers are in the 21-29 age group and typically play games every day or at least a few time a week. This information is vital to see who is being most affected by the games, and whether this has a positive or negative impact. (see figure 20)

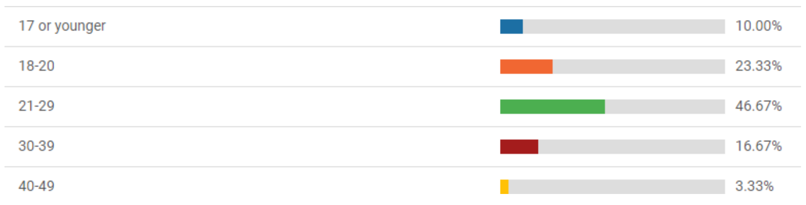


Figure 20. *Breakdown of gamers in age groups based on my survey*

From the results it is evident that played regularly were more into single-player games and aimed to achieve 100% completion as it provided a sense of satisfaction. Gamers were also split into those who were inclined to buy a game due to its popularity pre-release, and those who are influenced by their peers to purchase a game. On one hand, some genres like adventure games have proven to enlarge the hippocampus in the brain and trends of better memory can be seen in gamers compared to non-gamers. [37] Many other industries have also noticed this as aspects of games are placed into multiple different scenarios such as jobs. These show agility, muscle memory, reaction time and most importantly hand-eye co-ordination, which can all be beneficial to the gamer as well as employers and other everyday life skills.

On the other hand, there are of course downsides to gaming. Studies of the brain show that children and teenagers that have games readily available to them tend to have less self-control and increased emotional responses. [44] This is typically why publishers select age ratings for games to categorise the correct audience that should be playing a specific title, something explored later in the report. Role play games such as IMVU and VR chat allow users to enter a virtual reality and cases have been reported that users replace face to face social interaction with in-game chats. Some organisations that aim to help treat gaming addiction as this is a very common occurrence amongst gamers. Playing games releases dopamine in the brain and therefore attracts players to return and play more. Although this can be seen as a positive, there are multiple side effects such as fatigue, eye strain, spinal misalignment, and even social replacement. Research suggests that more children are replacing real-life activities with virtual ones and this has also created spikes in cyberbullying and predatorial attacks. [38]

## 7.2 Developers

Developers are just as important as gamers. They attempt to see from the player's perspective and think about every aspect of how a game may be played and what will be encountered. Developers speak directly to writers, actors, and publishers to develop a game and therefore have a key impact on the direction of the game's narrative and the way it is portrayed.

There is clear evidence that in some games, developers actively glorify topics that should not be advocated for. Game franchises such as Grand theft auto is an obvious example of this. [39] Rockstar Games have been heavily criticized for glorifying gang culture, sexual assault, and gun/drug misuse. Although their clarity and warnings from the developers that these games are fictional and should not represent real life, there is an undeniable scene of realism. This is the reason so many players are attracted to games like grand theft auto, which provide that sense of escapism users want. There have been countless examples of crime in the real world that have outstanding similarity to game missions as well as some criminals actively blaming the game for influencing their actions. [40] Developers are known to add controversial topics in their games as it leads to a conversation in today's society, this then directly leads to more revenue for the franchise itself. (see figure 21)

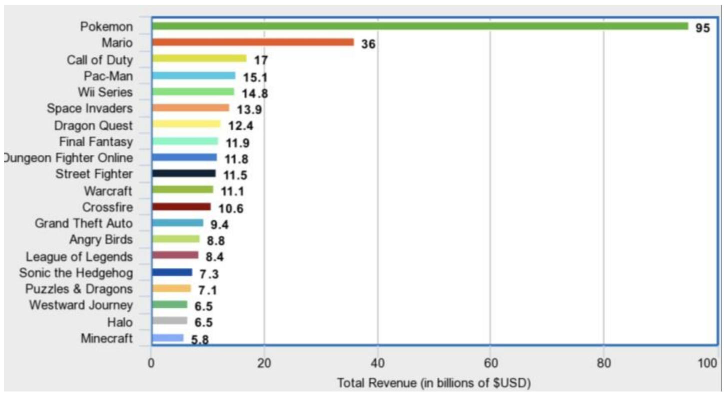


Figure 21. *A breakdown of the top 20 gaming franchises revenue. [41]*

Despite the negatives, developers must also be praised for the leap of gaming in the past 50 years. To think that the very first games such as pong were 32-bit with 28 lines of code and 50 years later we have AAA titles that are critically acclaimed and globally available is outstanding. These new games are also more accessible than ever before. Colour-blind settings, audio descriptions and peripherals for people with disabilities can bring everyone together for a gaming experience.

## 7.3 Publishers

The final part of the gaming ecosystem and the "big three" are publishers. They have the responsibility to release the game to the public as well as maintain the game after launch with the developers.

Studies show that more families are openly allowing their kids to play 18+ games. The reasoning behind this is the availability and ease of purchasing these games. A domino effect takes place as one child may purchase an 18+ title and then friends want to be playing the same games, putting pressure on parents to allow children to play the games their friends are playing. Moreover, children are able to purchase games themselves without the need for an identification or age check. Marketplaces such as PlayStation Plus and Steam do not require you to authenticate your age and proactively give free titles of all age ratings. Making it easier for children to download games themselves. It is easy to place the blame on parents for not checking the games their children are playing, but it is important to hold publishers accountable for the lack of restrictions on children accessing these games. (see figure 22)

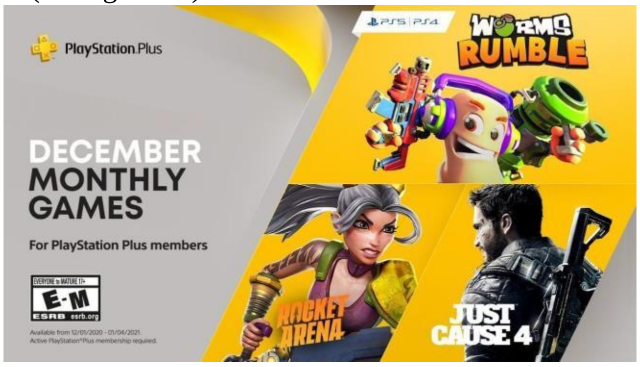


Figure 22. *PlayStation plus free games for December 2020, including 18+ titles*

Publishers and developers have also sparked a conversation about the effects of in-game purchases. Games such as Fortnite, which is a free-to-play game, makes its revenue directly from advertising and its in-game store. In 2019 alone Fortnite, was recorded to make over 1.8 billion dollars across their 125 million players [42]. This can be problematic as many see in game purchases as a form of gambling and addiction which can ultimately spark interest in real gambling in the future, as many see gambling as a form of gaming too.

Advertising plays an important role for publishers too. When the development phase is over the specific game needs to reach an audience on a global platform. With the growth of social media and the influx of children on websites such as YouTube, lucrative advertisement deals can be made to lure the audience into downloading their game. An example everybody sees is TV and billboard advertising. Trailers of games are played between children's TV programmes as a form of marketing. This allows the domino effect spoken about previously to take place.

Another form of advertising is through sponsorships with YouTubers. YouTube currently has two billion users worldwide with one billion hours of play time daily. [43] Content creators are offered sponsorships from publishers to promote games to their audience, which monetises their videos and causes a growth of users on the games being promoted. A clear example of this is again on the game Fortnite. Developers Epic Games partnered with creators to encourage YouTube audience to buy in-game items and whilst doing so they are supporting their favourite YouTube creators. Children of young ages are easily influenced by YouTubers and many see them as role models, whilst they think they are supporting their favourite content creators, in reality a fraction of revenue is given to them whilst the majority is taken by the publishers.

# Chapter 8: **Conclusion**

To Conclude, the project as a whole can be seen as a clear success. My knowledge of games development, C# and the Unity framework is at a competent level where I have confidence to understand C# code if presented to me as well as navigate my way around the Unity framework. Playing games in my free time has now opened my eyes into a developers mindset and not just a casual player. Being able to understand how logic in the games I play everyday and notice minute details that others wouldn't notice is clear evidence of the change of my knowledge in games development.

## 8.1 Project Overview

The gameplay of the project was broken up into separate sections which were handled by Git branches. From here, I carried out segmented development whilst testing out each gameObject individually in a tester scene before merging it in with other build assets. This allowed me to make sure there are no compiler errors as well as pinpoint any issues I did come across for a fix. Testing the game in a casual environment allowed feedback to be a conversation and not criticism, I wanted to hear the positives and negatives of testers playing my game in real time so bugs and unwanted features could be fixed. The Unity asset store allowed me to import and play around with many different assets which subsequently changed the forecast of my game as new ideas were thought about.

The use of my project plan, diary and meetings led to well-structured development, research, and cohesiveness. The layout of the report follows a timeline of my knowledge of games development prior to my background research and prototypes to the end where I have confidence in my skills with Unity and C# as well as a clear understanding into the business and psychological aspect of development.

## 8.2 What's next

### 8.2.1 What have I learnt

Throughout this project, I have grown a great interest into Unity and the game development industry. Whilst planning, developing, and editing my code I have seen the intuitive design elements of C# and the Unity framework. Learning about the importance of scene management has allowed me to better structure my work and find different ways to access resources that are needed. Furthermore, working first-hand with other people's assets and code has given me insight into how others may code and the positives and negatives that come with it. I have learnt to adapt code and assets from multiple resources to make use of them in my own project and the same goes for the sources I have found for this report.

The project overall has also shown me the importance of time management. Allocating tasks and working towards deadlines is something that is vital in everyday life and this project has helped me better myself for those moments.

### 8.2.2 What went well or could have gone better

There are many positives to take out of my game and the project as a whole. Many are mentioned above in what I have learnt. In addition to those, I believe that my skills in adapting to new software and coding languages have drastically improved. Being able to familiarise myself with C# in a small timeframe is something I am proud of. Creating a large terrain with depth for the player to navigate around seamlessly and coding the player vs. enemy for it to add a sense of survival, thrill and overall excitement was something I was glad to achieve. Regularly scheduled meetings were also a positive out of this project, due to conversation and feedback, an overwhelming change was seen in the scope of my game. Constructive criticism is crucial for me in large projects and therefore working off feedback to better myself and my project was something that also went well.

On the other hand, there are factors that could have been improved during the project. Firstly, if time was better resources across all studies rather than taking breaks to focus on one module, I could have added more depth to the game. Finding ways to make the game less repetitive and giving more replay time for the user is very important in a game and this could have been introduced across more time. In addition to this, information could have been displayed in other ways on the heads up display rather than a standard format. Furthermore, the lack of interaction due to the COVID-19 pandemic has led the development of the project to be static. Not being able to have consistent interaction with supervisors and peers for feedback and new ideas has made the project a harder obstacle to overcome.

### 8.2.3 What is next

This project aided my goals for the near future. Choosing a subject based on game design was important for me, as it was one topic that was lacking in my portfolio of computer science as a whole. Personally, I find it important to be competent and well-rounded in all forms of your studies and therefore game deign in a new language and framework was an excellent choice. I am to continue to challenge myself to broaden my knowledge of computer science after this project and university also.

# Chapter 9: **Bibliography & Appendix**

[1] UnityHub – Software of unity framework, Unity (2021) –
https://unity3d.com/get-unity/download

[2] Specifics of an IDE, Code academy (2021) –
https://www.codecademy.com/articles/what-is-an-ide

[3] Visual Studio Documentation, Microsoft (2019) –
https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019

[4] Auto-complete statistics – Danny Sullivan, Google (2018) –
https://blog.google/products/search/how-google-autocomplete-works-search/#:~:text=On%20average%2C%20it%20reduces%20typing,Yes%2C%20per%20day

[5] What is Gaming - Joseph Evans, SWGfl (2017) –
https://swgfl.org.uk/magazine/what-is-gaming/

[6] History of video game consoles, Wikipedia (2021) –
https://en.wikipedia.org/wiki/History_of_video_game_consoles

[7] Reasons for gaming viewership on YouTube – Ekaterina Petrova, Google (2017) -
https://www.thinkwithgoogle.com/data-collections/gamer-demographics-gaming-statistics/

[8] eSports overtaking traditional sports (analysis) – Alesia Krush, ObjectStyle (2020) -
https://www.objectstyle.com/agile/esports-may-soon-overtake-traditional-sports-analysis

[9] PS4 vs PS5: Load time comparisons – OriginZ, YouTube (2019) -
https://www.youtube.com/watch?v=XmLYh1ybLqs&ab_channel=OriginZ

[10] Active use of motion capturing in The Last of Us – ILLUMINAT3D, YouTube (2020) -
https://www.youtube.com/watch?v=C3OnriYLVPw&ab_channel=ILLUMINAT3D

[11] Unity SceneManager Documentation, Unity (2020) -
https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html

[12] Generating an array for render quality – UnityFourms, Unity (2018) -
https://answers.unity.com/questions/1463609/screenresolutions-returning-duplicates.html

[13] Unity Prefab Documentation, Unity (2020) -
https://docs.unity3d.com/Manual/Prefabs.html

[14] Unity Animation Documentation, Unity (2020) -
https://docs.unity3d.com/Manual/AnimationOverview.html

[15] Shooting with Raycasts – Brackeys, YouTube (2017) -
https://www.youtube.com/watch?v=THnivyG0Mvo&t=507s&ab_channel=Brackeys

[16] Collision Logic in Unity3D – Joseph Hocking, Unity in Action (2015) -
Unity in Action Multiplatform game development in C# with Unity 5 – Joseph Hocking

[17] Unity NavMesh Documentation, Unity (2020) -
https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html

[18] Lighting in Unity – Brackeys, YouTube (2018) - https://www.youtube.com/watch?v=VnG2gOKV9dw&t=4s&ab_channel=Brackeys

[19] Controlling Unity Camera Behaviour – Unity Technologies, Unity (2020) - https://learn.unity.com/tutorial/controlling-unity-camera-behaviour

[20] FPS Movement and Jumping – Dapper Dino, YouTube (2018) - https://www.youtube.com/watch?app=desktop&v=1o-Gawy3D48&t=7s&ab_channel=DapperDino

[21] Unity AI Tutorial – Adrian Chase, YouTube (2019) - https://www.youtube.com/playlist?list=PLc74DR4WWHumNpmv-FEfB8KxqQyx4xEG-

[22] FPS Game in Unity Tutorial Playlist – Single Sapling Games, YouTube (2019) - https://www.youtube.com/playlist?list=PLKklF7YNi0lOM0C8r_L3JN3oTC6AY9iFE

[23] Logic of AI Pathfinding and attacking – Ray Barrera (2015)
Unity AI Game Programming – Ray Barrera

[24] Unity Collision Documentation, Unity (2020) - https://docs.unity3d.com/ScriptReference/Collision.html

[25] Zombie Animations & Attack – Adrian Chase, YouTube (2018) - https://www.youtube.com/watch?v=u5rmP_kAJ2o&list=PLc74DR4WWHumNpmv-FEfB8KxqQyx4xEG-&index=5&t=6s&ab_channel=AdrianChase

[26] Weapon Handling and Gun switching – Brackeys, YouTube (2017) - https://www.youtube.com/watch?v=Dn_BUIVdAPg&t=63s&ab_channel=Brackeys

[27] TextMeshPro Tutorial – UnityTechnologies, Unity (2020) - https://learn.unity.com/tutorial/working-with-textmesh-pro

[28] Creating and navigating scenes – Brackeys, YouTube (2017) - https://www.youtube.com/watch?v=zc8ac_qUXQY&ab_channel=Brackeys

[29] Nielsen's 10 UI Heuristics – Atefe Rahdan, UX Collective (2020) - https://uxdesign.cc/jakob-nielsens-10-heuristics-for-user-interface-design-3fe09af5fd99

[30] Benefits of a Project Diary – Harry Hall, Project Risk Coach (2021) - https://projectriskcoach.com/7-benefits-of-keeping-a-project-journal/

[31] Unique game testing techniques – Niranjan Limbachiya, DZone (2020) - https://dzone.com/articles/7-different-types-of-game-testing-techniques

[32] Game Design patterns that developers should use – Jason Weimann, YouTube (2020) - https://www.youtube.com/watch?v=hQE8lQk9ikE&ab_channel=JasonWeimann

[33] Game Programming Patterns – Robert Nystrom (2011) - https://gameprogrammingpatterns.com/contents.html

[34] Unity Design Patterns in C# - Erik Nordeus, Harbour (2021) - https://www.habrador.com/tutorials/programming-patterns/

[35] Video Game statistics – Deyan G, TechJury (2021) - https://techjury.net/blog/video-games-industry-statistics/#gref

[36] Gaming Survey Dissertation – Harshdeep Chopra, SmartSurvey (2020) - https://www.smartsurvey.co.uk/s/E0AWI6/

[37] Short/Long term effects of video games – Clare Knight, News Medical (2019) - https://www.news-medical.net/health/Short-Term-and-Long-Term-Effects-of-Playing-Video-Games.aspx#1

[38] Pros & Cons of Video Games – Dr. Jonathen Bartholomew, UV Paediatrics (2019) - https://www.uvpediatrics.com/topics/video-games-pros-and-cons/

[39] Real life examples of GTA influence – Tara Jabra, Medium (2019) - https://medium.com/jsc-419-class-blog/gta-v-promotion-of-violence-through-glorified-deviance-to-young-adults-4e3d27e90a75

[40] Criminal acts blamed on video games – Chris Langton, The Gamer (2017) - https://www.thegamer.com/15-despicable-crimes-that-got-blamed-on-video-games/

[41] Highest Grossing gaming franchises – Wikipedia (2019) - https://www.reddit.com/r/gaming/comments/dxnt27/a_graph_of_the_highestgrossing_video_game/

[42] How does a free game (Fortnite) make money – Akhilesh Ganti, Investopedia (2020) - https://www.investopedia.com/tech/how-does-fortnite-make-money/#:~:text=Key%20Takeaways-,Fortnite%20is%20a%20free%2Dto%2Dplay%20video%20game%20set%20in,in%20revenues%2 0of%20%241.8%20billion.

[43] YouTube Statistical Analytics – YouTube (2021) - https://www.youtube.com/intl/en-GB/about/press/#:~:text=One%20billionhours%20watched%20daily,day%2C%20generating%20billions%20of%20views.

[44] Psychological Impact on gaming – Jamie Madigan, Getting Gamers (2015) - Getting Gamers: The Psychology of Video Games and Their Impact on the People who Play Them - Jamie Madigan

## List of Unity Assets in the project

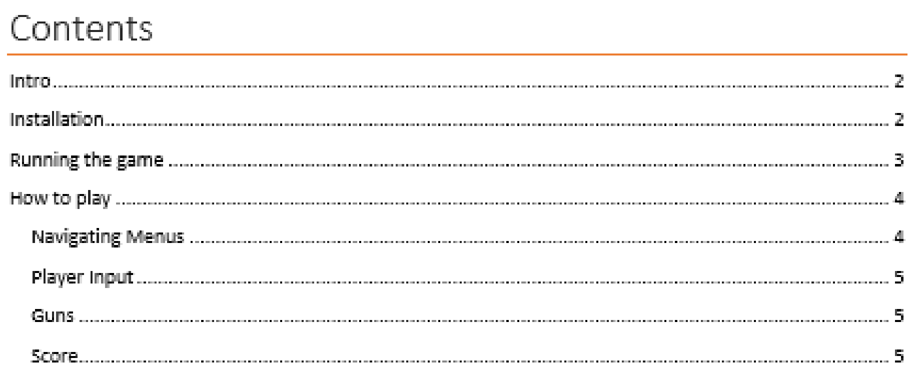
| Asset Name | Publisher |
|---|---|
| **AllSky Free – 10 Sky / Skybox Set** | rpgwhitelock |
| **Basic Motions (Free Pack)** | POLYGONCRAFT |
| **Conifers [BOTD]** | Forst |
| **Fantasy Forest Environment – Free Demo** | TriForge Assets |
| **Free Zombie Character Sounds** | Idia Software LLC |
| **Grass Flowers Pack Free** | ALP8310 |
| **Low Poly Crates** | PULSAR BYTES |
| **Low poly Forest environment package** | Free Dimension Forge |
| **Modern Zombie Free** | Artalasky |
| **Outdoor Ground Textures** | A dog's life software |
| **Terrain Tools Sample Asset Pack** | Unity Technologies Inc. |
| **Unity Particle Pack 5.x** | Unity Technologies Inc. |
| **Zombie Animation Pack Free** | Animus Digital |
| **Modern Weapons** | DevAssets – Matias Hollmann |

## 9.1 Project User Guide

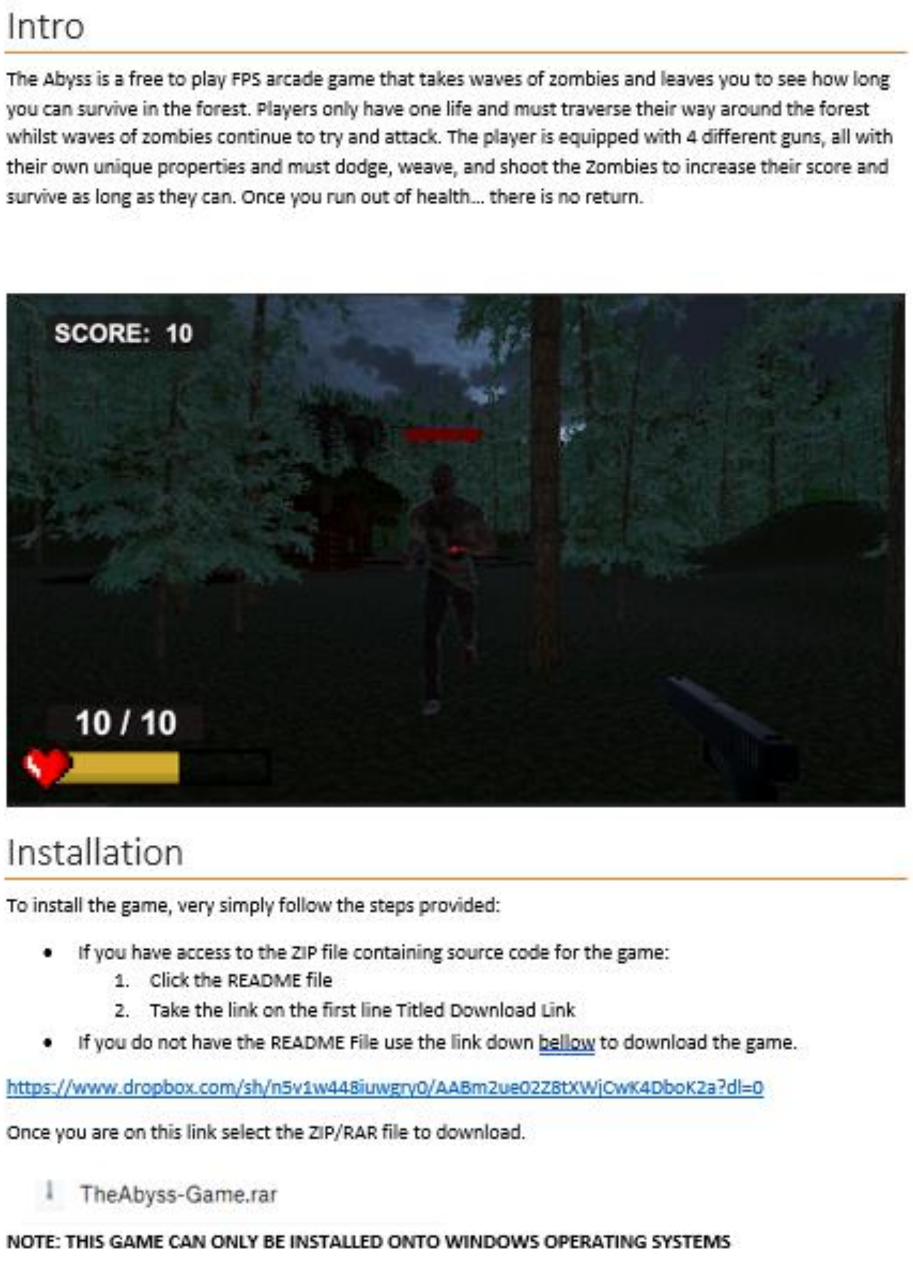
# The Abyss

USER GUIDE

Chopra, Harshdeep (2018)

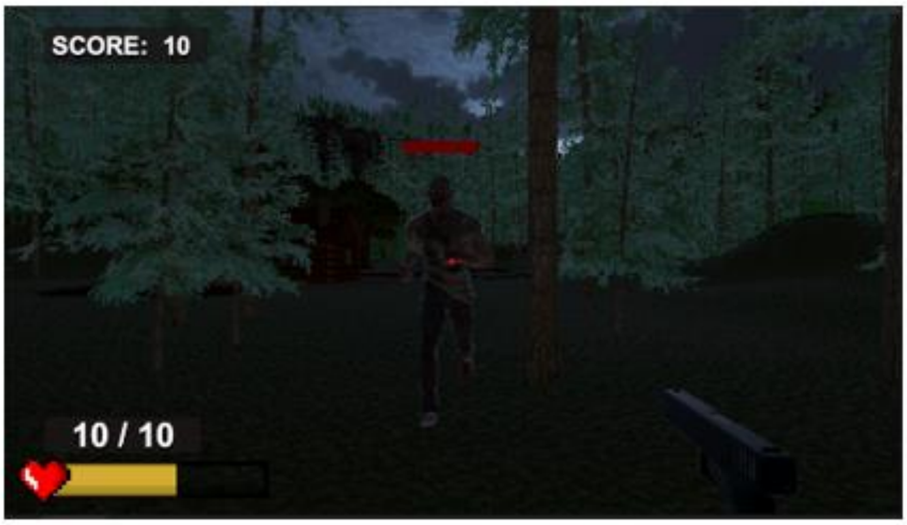


## Contents



## Intro

The Abyss is a free to play FPS arcade game that takes waves of zombies and leaves you to see how long you can survive in the forest. Players only have one life and must traverse their way around the forest whilst waves of zombies continue to try and attack. The player is equipped with 4 different guns, all with their own unique properties and must dodge, weave, and shoot the Zombies to increase their score and survive as long as they can. Once you run out of health… there is no return.



## Installation

To install the game, very simply follow the steps provided:

- If you have access to the ZIP file containing source code for the game:
  1. Click the README file
  2. Take the link on the first line Titled Download Link
- If you do not have the README File use the link down bellow to download the game.

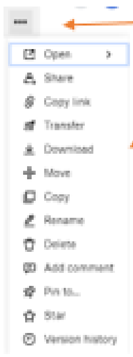
https://www.dropbox.com/sh/n5v1w448iuwgry0/AABm2ue02Z8tXWjCwK4DboK2a?dl=0

Once you are on this link select the ZIP/RAR file to download.

TheAbyss-Game.rar

**NOTE: THIS GAME CAN ONLY BE INSTALLED ONTO WINDOWS OPERATING SYSTEMS**

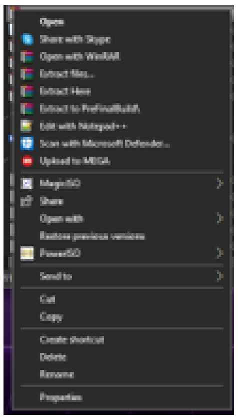
**To download the file:**



- Select the more (ellipses) option on the specific zip file and click on download.

Once downloaded (Extract):

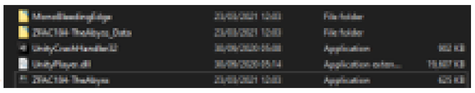
1. find the zip file on your device.
2. Right click and extract using the extraction application of your choice



## Running the game

Once extracted. Click on the folder and find the final Application to run the game. ZFAC184-TheAbyss.exe

This may take a few minutes as Unity is setting up the Build assets for the first time on your device. After this you are ready to play. Keep this application and all other files in this folder to allow the game to find its necessary resources.



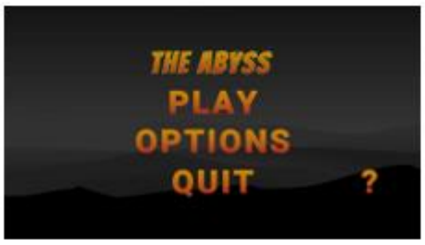
| | | | |
|---|---|---|---|
| MonoBleedingEdge | 23/03/2021 12:03 | File folder | |
| ZFAC184-TheAbyss_Data | 23/03/2021 12:03 | File folder | |
| UnityCrashHandler32 | 30/06/2020 05:08 | Application | [illegible] KB |
| UnityPlayer.dll | 30/06/2020 05:14 | Application exten... | [illegible] KB |
| ZFAC184-TheAbyss | 23/03/2021 12:03 | Application | 625 KB |

## How to play

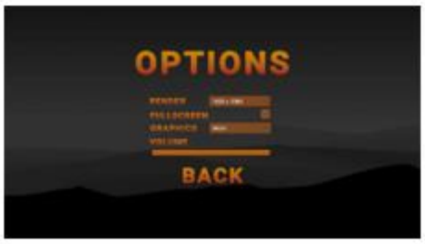
### Navigating Menus

Once loaded, The Main menu will start. You can start the game by pressing play, change settings in the options menu, Quit the game and return to the desktop or press ? for a how to play scene.

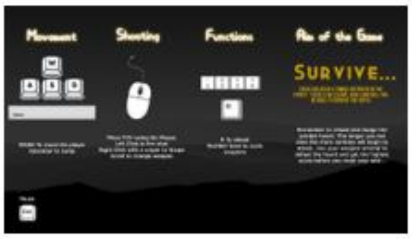
If you remain idle on the main menu, a demo video will play which demonstrates the gameplay.



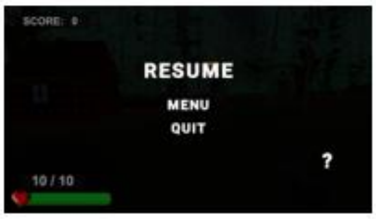
Change your volume, full screen toggle and render settings from the main menu



The how to play screen provides the key user input to play the game



You can pause the game by pressing the ESC (Escape key on your keyboard). From here you can resume game, quit, or return to the main menu.



## Player Input

| Keyboard Input | Functionality |
|---|---|
| WASD | Player movement around the forest.<br>W moving the player forward<br>A moving the player left<br>S moving the player backwards<br>D moving the player right |
| Spacebar | Player Jump |
| Mouse | Move the mouse to look and aim |
| Left mouse button | Shoot |
| Right mouse button | Only for Sniper rifles – AIM |
| Scroll Wheel / " 1 " , " 2 ", " 3 ", " 4 " | Cycle through weapons |
| ESC | Pause Game |
| R | Reload |

## Guns

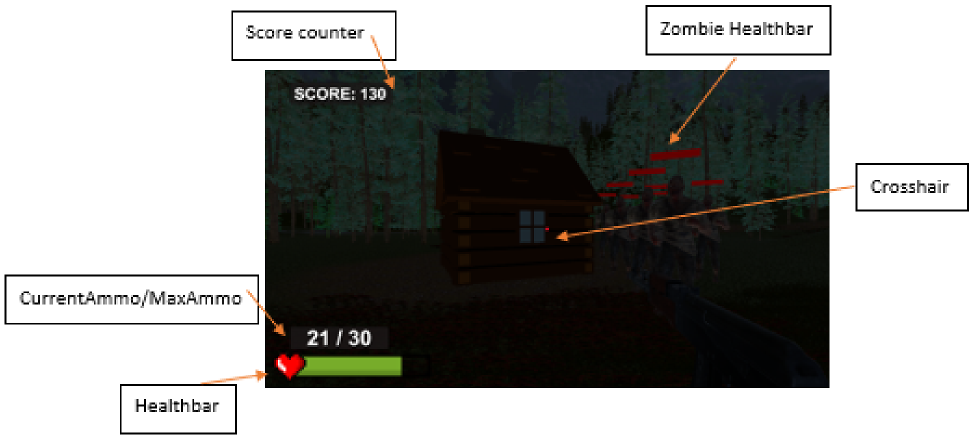
| Gun Name | Gun Damage | Gun Range | Gun Reload Time | Gun Clip size |
|---|---|---|---|---|
| **Pistol** | 15 health points | 40f | 1 Second | 10 bullets |
| **Shotgun** | 50 health points | 30f | 3 Seconds | 8 bullets |
| **Rifle** | 30 health points | 70f | 2 Seconds | 30 bullets |
| **Sniper** | 100 health points | 100f | 5 Seconds | 1 Bullet |

Gun ammunition must be handled by the player. The clip size and current ammunition in the magazine can be seen above the player health bar and once the clip has 0 bullets, the player must reload.

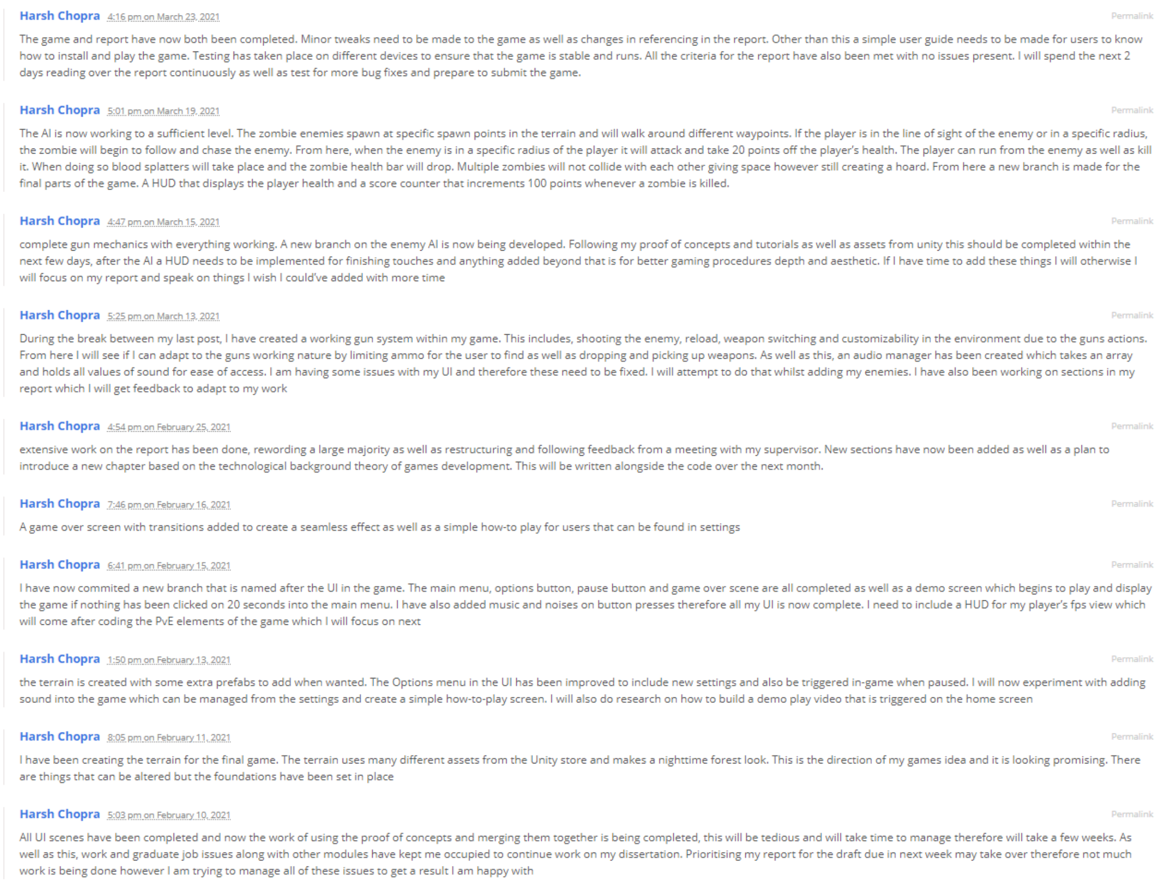
## Score

10 Points are given whenever a zombie is eliminated from the game.

Scores are held in the top left of the screen and as the game progresses more zombies will spawn, and more points are able to be gathered. Compete with yourself and friends to see who can get the highest score.



## 9.2 Project Diary



**Harsh Chopra** 4:16 pm on March 23, 2021 Permalink

The game and report have now both been completed. Minor tweaks need to be made to the game as well as changes in referencing in the report. Other than this a simple user guide needs to be made for users to know how to install and play the game. Testing has taken place on different devices to ensure that the game is stable and runs. All the criteria for the report have also been met with no issues present. I will spend the next 2 days reading over the report continuously as well as test for more bug fixes and prepare to submit the game.

**Harsh Chopra** 5:01 pm on March 19, 2021 Permalink

The AI is now working to a sufficient level. The zombie enemies spawn at specific spawn points in the terrain and will walk around different waypoints. If the player is in the line of sight of the enemy or in a specific radius, the zombie will begin to follow and chase the enemy. From here, when the enemy is in a specific radius of the player it will attack and take 20 points off the player's health. The player can run from the enemy as well as kill it. When doing so blood splatters will take place and the zombie health bar will drop. Multiple zombies will not collide with each other giving space however still creating a hoard. From here a new branch is made for the final parts of the game. A HUD that displays the player health and a score counter that increments 100 points whenever a zombie is killed.

**Harsh Chopra** 4:47 pm on March 15, 2021 Permalink

complete gun mechanics with everything working. A new branch on the enemy AI is now being developed. Following my proof of concepts and tutorials as well as assets from unity this should be completed within the next few days, after the AI a HUD needs to be implemented for finishing touches and anything added beyond that is for better gaming procedures depth and aesthetic. If I have time to add these things I will otherwise I will focus on my report and speak on things I wish I could've added with more time

**Harsh Chopra** 5:25 pm on March 13, 2021 Permalink

During the break between my last post, I have created a working gun system within my game. This includes, shooting the enemy, reload, weapon switching and customizability in the environment due to the guns actions. From here I will see if I can adapt to the guns working nature by limiting ammo for the user to find as well as dropping and picking up weapons. As well as this, an audio manager has been created which takes an array and holds all values of sound for ease of access. I am having some issues with my UI and therefore these need to be fixed. I will attempt to do that whilst adding my enemies. I have also been working on sections in my report which I will get feedback to adapt to my work

**Harsh Chopra** 4:54 pm on February 25, 2021 Permalink

extensive work on the report has been done, rewording a large majority as well as restructuring and following feedback from a meeting with my supervisor. New sections have now been added as well as a plan to introduce a new chapter based on the technological background theory of games development. This will be written alongside the code over the next month.

**Harsh Chopra** 7:46 pm on February 16, 2021 Permalink

A game over screen with transitions added to create a seamless effect as well as a simple how-to play for users that can be found in settings

**Harsh Chopra** 6:41 pm on February 15, 2021 Permalink

I have now commited a new branch that is named after the UI in the game. The main menu, options button, pause button and game over scene are all completed as well as a demo screen which begins to play and display the game if nothing has been clicked on 20 seconds into the main menu. I have also added music and noises on button presses therefore all my UI is now complete. I need to include a HUD for my player's fps view which will come after coding the PvE elements of the game which I will focus on next

**Harsh Chopra** 1:50 pm on February 13, 2021 Permalink

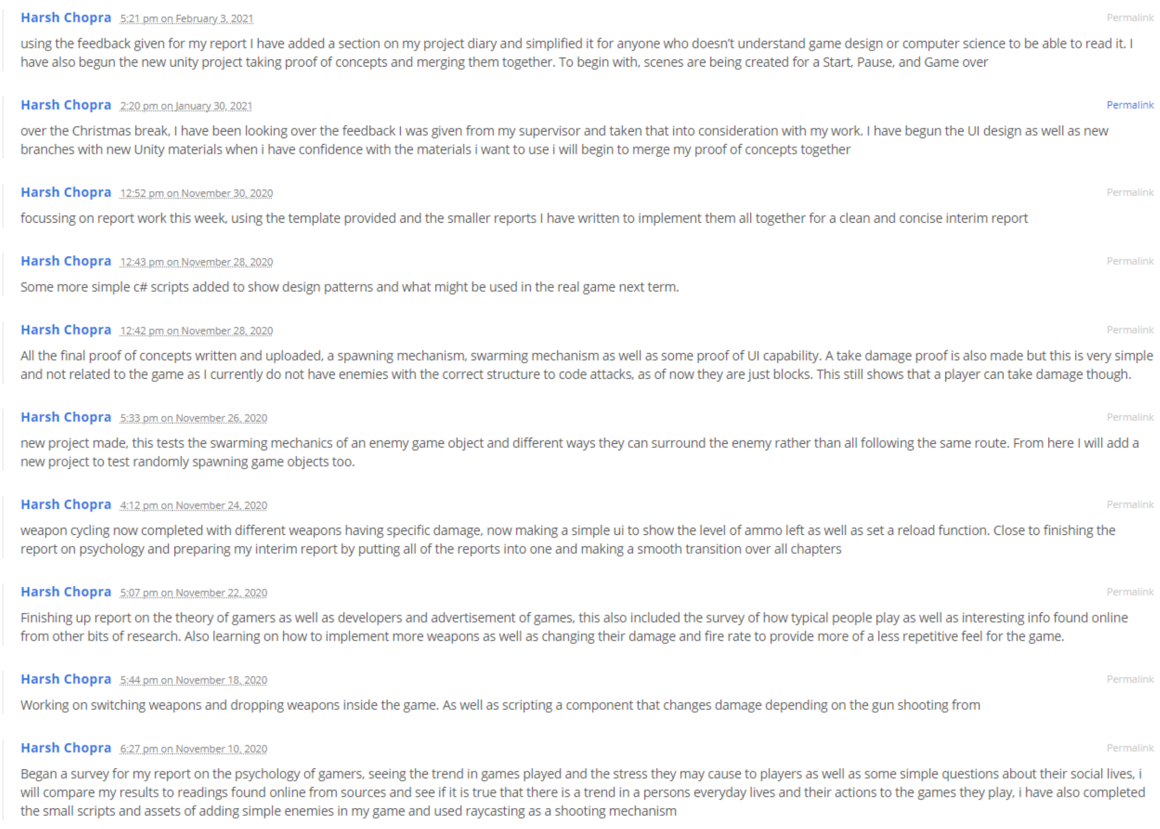
the terrain is created with some extra prefabs to add when wanted. The Options menu in the UI has been improved to include new settings and also be triggered in-game when paused. I will now experiment with adding sound into the game which can be managed from the settings and create a simple how-to-play screen. I will also do research on how to build a demo play video that is triggered on the home screen

**Harsh Chopra** 8:05 pm on February 11, 2021 Permalink

I have been creating the terrain for the final game. The terrain uses many different assets from the Unity store and makes a nighttime forest look. This is the direction of my games idea and it is looking promising. There are things that can be altered but the foundations have been set in place

**Harsh Chopra** 5:03 pm on February 10, 2021 Permalink

All UI scenes have been completed and now the work of using the proof of concepts and merging them together is being completed, this will be tedious and will take time to manage therefore will take a few weeks. As well as this, work and graduate job issues along with other modules have kept me occupied to continue work on my dissertation. Prioritising my report for the draft due in next week may take over therefore not much work is being done however I am trying to manage all of these issues to get a result I am happy with



**Harsh Chopra** 5:21 pm on February 3, 2021 Permalink

using the feedback given for my report I have added a section on my project diary and simplified it for anyone who doesn't understand game design or computer science to be able to read it. I have also begun the new unity project taking proof of concepts and merging them together. To begin with, scenes are being created for a Start, Pause, and Game over

**Harsh Chopra** 2:20 pm on January 30, 2021 Permalink

over the Christmas break, I have been looking over the feedback I was given from my supervisor and taken that into consideration with my work. I have begun the UI design as well as new branches with new Unity materials when i have confidence with the materials i want to use i will begin to merge my proof of concepts together

**Harsh Chopra** 12:52 pm on November 30, 2020 Permalink

focussing on report work this week, using the template provided and the smaller reports I have written to implement them all together for a clean and concise interim report

**Harsh Chopra** 12:43 pm on November 28, 2020 Permalink

Some more simple c# scripts added to show design patterns and what might be used in the real game next term.

**Harsh Chopra** 12:42 pm on November 28, 2020 Permalink

All the final proof of concepts written and uploaded, a spawning mechanism, swarming mechanism as well as some proof of UI capability. A take damage proof is also made but this is very simple and not related to the game as I currently do not have enemies with the correct structure to code attacks, as of now they are just blocks. This still shows that a player can take damage though.

**Harsh Chopra** 5:33 pm on November 26, 2020 Permalink

new project made, this tests the swarming mechanics of an enemy game object and different ways they can surround the enemy rather than all following the same route. From here I will add a new project to test randomly spawning game objects too.

**Harsh Chopra** 4:12 pm on November 24, 2020 Permalink

weapon cycling now completed with different weapons having specific damage, now making a simple ui to show the level of ammo left as well as set a reload function. Close to finishing the report on psychology and preparing my interim report by putting all of the reports into one and making a smooth transition over all chapters

**Harsh Chopra** 5:07 pm on November 22, 2020 Permalink

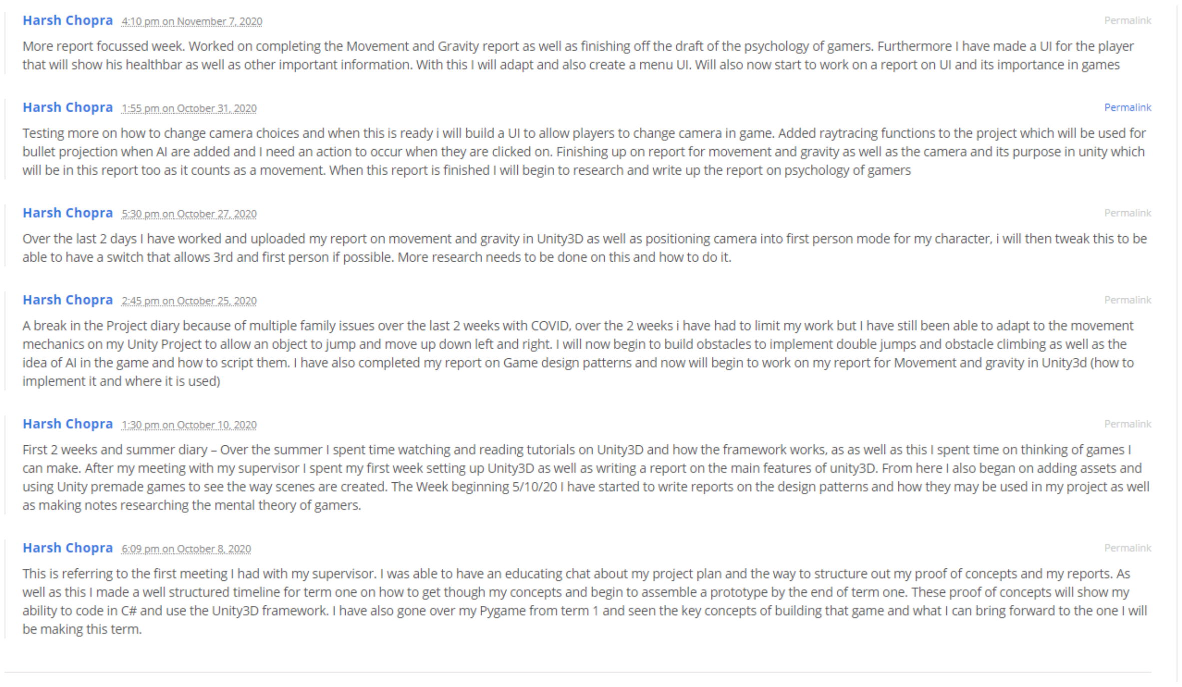
Finishing up report on the theory of gamers as well as developers and advertisement of games, this also included the survey of how typical people play as well as interesting info found online from other bits of research. Also learning on how to implement more weapons as well as changing their damage and fire rate to provide more of a less repetitive feel for the game.

**Harsh Chopra** 5:44 pm on November 18, 2020 Permalink

Working on switching weapons and dropping weapons inside the game. As well as scripting a component that changes damage depending on the gun shooting from

**Harsh Chopra** 6:27 pm on November 10, 2020 Permalink

Began a survey for my report on the psychology of gamers, seeing the trend in games played and the stress they may cause to players as well as some simple questions about their social lives, i will compare my results to readings found online from sources and see if it is true that there is a trend in a persons everyday lives and their actions to the games they play, i have also completed the small scripts and assets of adding simple enemies in my game and used raycasting as a shooting mechanism



**Harsh Chopra** 4:10 pm on November 7, 2020 Permalink

More report focussed week. Worked on completing the Movement and Gravity report as well as finishing off the draft of the psychology of gamers. Furthermore I have made a UI for the player that will show his healthbar as well as other important information. With this I will adapt and also create a menu UI. Will also now start to work on a report on UI and its importance in games

**Harsh Chopra** 1:55 pm on October 31, 2020 Permalink

Testing more on how to change camera choices and when this is ready i will build a UI to allow players to change camera in game. Added raytracing functions to the project which will be used for bullet projection when AI are added and I need an action to occur when they are clicked on. Finishing up on report for movement and gravity as well as the camera and its purpose in unity which will be in this report too as it counts as a movement. When this report is finished I will begin to research and write up the report on psychology of gamers

**Harsh Chopra** 5:30 pm on October 27, 2020 Permalink

Over the last 2 days I have worked and uploaded my report on movement and gravity in Unity3D as well as positioning camera into first person mode for my character, i will then tweak this to be able to have a switch that allows 3rd and first person if possible. More research needs to be done on this and how to do it.

**Harsh Chopra** 2:45 pm on October 25, 2020 Permalink

A break in the Project diary because of multiple family issues over the last 2 weeks with COVID, over the 2 weeks i have had to limit my work but I have still been able to adapt to the movement mechanics on my Unity Project to allow an object to jump and move up down left and right. I will now begin to build obstacles to implement double jumps and obstacle climbing as well as the idea of AI in the game and how to script them. I have also completed my report on Game design patterns and now will begin to work on my report for Movement and gravity in Unity3d (how to implement it and where it is used)

**Harsh Chopra** 1:30 pm on October 10, 2020 Permalink

First 2 weeks and summer diary – Over the summer I spent time watching and reading tutorials on Unity3D and how the framework works, as as well as this I spent time on thinking of games I can make. After my meeting with my supervisor I spent my first week setting up Unity3D as well as writing a report on the main features of unity3D. From here I also began on adding assets and using Unity premade games to see the way scenes are created. The Week beginning 5/10/20 I have started to write reports on the design patterns and how they may be used in my project as well as making notes researching the mental theory of gamers.

**Harsh Chopra** 6:09 pm on October 8, 2020 Permalink

This is referring to the first meeting I had with my supervisor. I was able to have an educating chat about my project plan and the way to structure out my proof of concepts and my reports. As well as this I made a well structured timeline for term one on how to get though my concepts and begin to assemble a prototype by the end of term one. These proof of concepts will show my ability to code in C# and use the Unity3D framework. I have also gone over my Pygame from term 1 and seen the key concepts of building that game and what I can bring forward to the one I will be making this term.