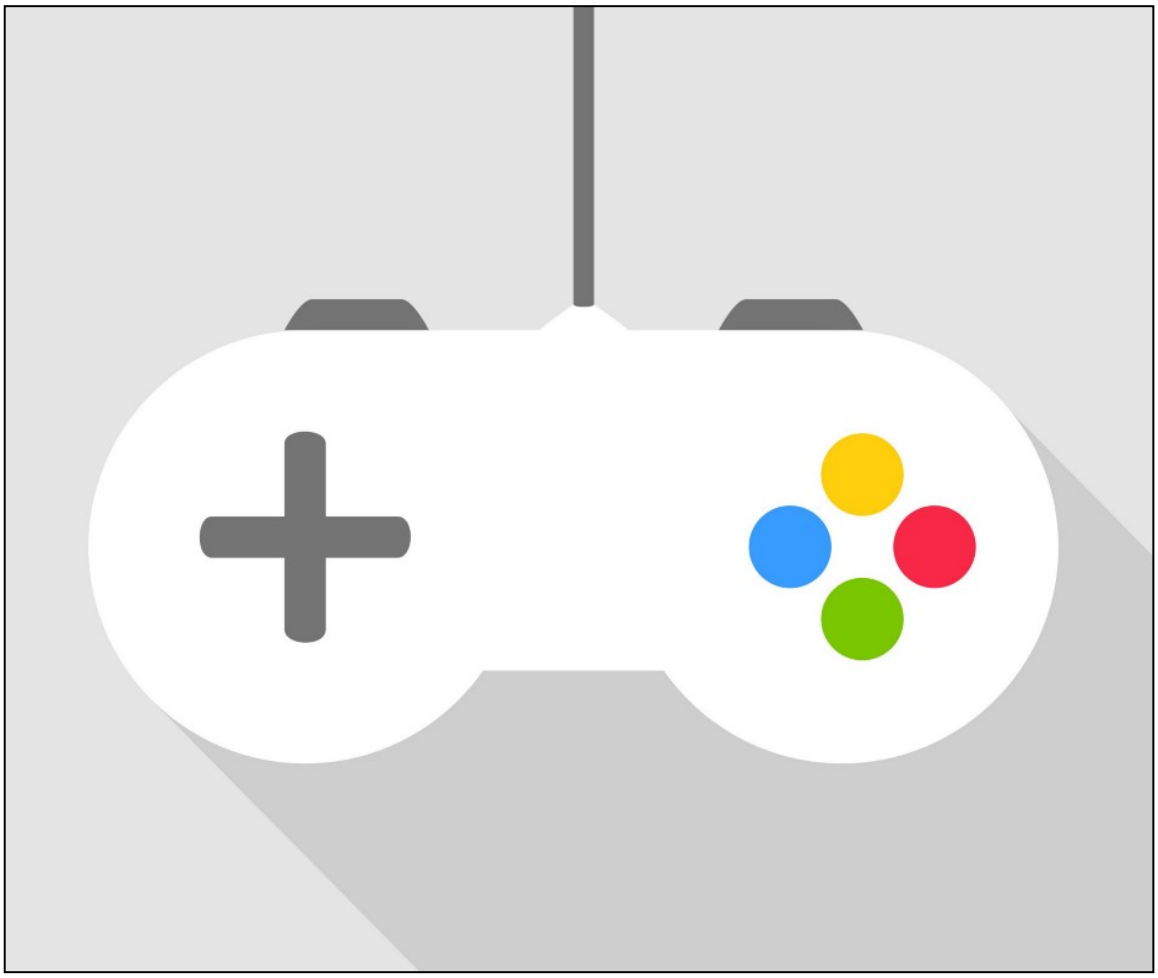
Making the ‘Perfect’ Video Game



Raj Mohammad

King's College London Mathematics School

Contents

Introduction

The gaming industry has seen rapid growth in the last decade. The improvement of hardware and technology in general has also seen video games improve in aspects such as graphics, artificial intelligence, and the progression from 2D to 3D worlds.

Gaming has become a popular form of entertainment, with platforms namely YouTube[2] and Twitch[3] allowing millions of people to watch professional gamers. It has come to a point where gaming has become its own sport and lets those gifted at games to represent teams and participate in competitions to win monetary rewards. For example, Kyle Giersdorf, also known by the nickname "Bugha", won a prize of $3 million in the Fortnite World Cup, the largest pool prize towards an individual in the history of gaming[4]!

Figure 1.1: E-Sports live event for League of Legends[5]

With the marketability for new-coming games, game studios have looked to produce the next sensation. But how hard really is this process? This article will aim to go through from the research process of attempting to find an answer for what is the 'perfect' game, to the production stage of creating the game itself alongside other aspects which improve the gaming experience including audio, art, and animation.

More specifically, I intend to explore the gameplay side and build a game with robust mechanics. Though visuals and audio have an impact in the overall experience, the most important component of a game arguably is the gameplay and how this psychologically intrigues the player.

Contrary to the simplicity of the matter, the term 'perfect' would be subjective to different viewpoints. Games are mostly based on personal preferences therefore the production of a game such that it satisfies everyone is not possible. However, I will venture to produce a game that would be best suited for the community of King's College London Mathematics School using opinions of students and teachers. This would simply involve surveying into order to get the most popular game genres, building an idea around the theme and finally producing and documenting the development.

Literature Review

The video game industry requires developers to have thorough understanding of the market and its target audience. I planned to focus my research on the specific game genres, having knowledge of different game mechanics while also realising which of these features and aspects are popular amongst certain genres.

Firstly, I decided to look at how to approach the question of what makes a perfect game. Rich Stanton's article titled 'Can there be such a thing as a perfect video game?' gives a good insight on this topic[6]. Stanton was able to clearly communicate that perfection in a game can only be achieved if no more features can be added to it. However, it was added that this cannot be practically attained due to the variety of preferences people have. Despite this, the author was able to give examples of game mechanics that have been implemented in exemplar games and its impact, as well as evaluating whether it was positive or negative. Stanton clearly outlined that due to different perspectives, a game will never have a positive rating of 100% in a large sample since there will always be criticism. This article was very useful because it was able to explain how the consideration of multiple factors like gameplay, storytelling, audio, and other aspects can help improve the experience of a video game. Hence, it has allowed me to think more broadly about the development stage of the game and features that I should take into account. Additionally, the reliability of the article can be reinforced by Stanton's experience in the gaming industry, specifically his critiquing ability as he is a games journalist. His experience working alongside IGN and Eurogamer shows he understands games well enough to give an unbiased review. One limitation is that there is nothing explicit that the article has contributed to towards the research such as how to deal with the difficulty of producing a game that achieves perfection, but rather just improved my understanding of the topic. Overall, I believe this article has helped me gain the required knowledge about designing a well-thought-out video game.

Next, I set out to look at the features that make a game up to standard. Daniel Super's response to 'What are the elements of a good game?' on Quora[7] tells us that the greatness of a game is not dependent on how the game is designed and produced, but rather the psychology of the player when playing the game. He expands on themes such as autonomy, competence, and relatedness and how these link to the player as well as how they are the key factors that makes a game stand-out. Similar to Rich Stanton's article, Super was able to conclude that games cannot be made perfect and also suggest successful mechanics that make a game better. Super's profile on Quora has received overwhelming positive remarks including but not limited to having 'an enormous wealth of experience and knowledge' and being able to 'provide very good answers to game development questions'. So superficially, there is no doubt that Super is an authentic source. On the other hand, one may argue that Super's answer is brief, concise and does not expand enough in each of the example games. Contrary to this, for the scope of the research, in-depth knowledge on a particular game is not needed. Contrary to Super using a forum website, the information can be intended towards any game developer, whether being hugely experienced or someone relatively new to the industry, making it versatile and useful. Daniel Super's contribution has allowed me to comprehend a crucial factor that I did not initially consider, player psychology. I hope to research more about the effects and how certain psychologies can be achieved. Ultimately, his response will allow me to take into consideration the psychology factor in improving the experience of the game.

Another problem I had to consider was which game engines or frameworks that can be potentially used. The video by Ask Gamedev on YouTube titled 'The Best Game Engines of 2021' gave the positives and negatives of each of the various options of game engines available[8]. The channel considers factors including licensing and the scripting languages used, which was very insightful and informative. Additionally, the history of all the engines is given alongside their respective creators and examples of games that had been produced with the engines, instantly suggesting that the video was intended for an audience with no experience with game engines. After giving a brief overview, the channel suggests possible uses for the different engines including whether they would be suitable for 2D or 3D game production, animation, non-programming possibilities and more. There was little to no limitations in the source since the information on each of the game engines was thorough and sufficient for the extent of the project. The video itself gained a considerably large likes to dislike ratio which implies that the video was useful for many of the viewers. In addition, the channel itself holds over 100K subscribers, meaning the information given in this video can be considered reliable due to its large viewing audience. In conclusion, after watching this video, I can now come to a reasonable judgement on which game engine to use for the production of the game.

Lastly, another key feature of video games development I had to explore was game mechanics. I used OK Beast's video on YouTube titled 'Breaking Down Video Game Mechanics' to help with this[9]. OK Beast gives an interpretation of mechanics as actions within a game that allow the user to interact with its environment to create gameplay. The video expands on how a simple mechanic like jumping in Super Mario Odyssey[10] or the axe in God of War[11] are the primary focus. From this starting point, there is the ability to build upon the feature and improve. For example, Super Mario: Odyssey allows the jumping mechanics to destroy enemies and hit blocks to spawn coins and powerups. However in God of War, a levelling system gives players the option to increase the power of the axe. Despite the low subscriber count (just less than 10K), the video includes good analysis of the use of game mechanics in games and is informative on the subject. Therefore, the sources will be very useful when making the game because my focus will shift more towards how the main mechanic will help improve gameplay and hopefully make the game more fun. Although other mechanics can be added, the source concludes that the main mechanic will be the most importance because it will be most significant, ergo the majority of the criticism will be based on this. Therefore, the focus should be on solidifying the mechanic the game is based on. Overall, I believe my understanding of game mechanics has improved after watching the video and will help with the development of the game.

To sum up, all the sources mentioned will be of great use, whether having a direct impact to the project or aiding to understanding a topic. For example, the YouTube video from Ask Gamedev has informed me of all the game engines I can use, including those I have not heard of. This means I can now make a rational decision on which engine I should use based on the features it offers, making the development stage of the project as efficient as possible. Additionally, I now have a strong foundation to start the designing stage of the project knowing the application of mechanics and which game engine I should use based on the results of the questionnaire (this will be discussed further in the Game Proposal section). Additionally with reference to Super's response Quora, I will now have given thought to the players' experience and perspective, something which I would not have initially done.

Planning

In this section, I will list all the objectives I plan on doing for this project. Also, I will explain how these tasks will be organised.

Objectives

The approach for this project will follow chronologically from the following list:

- Researching popular games in the industry and the reason for their popularity
- Researching game engines and frameworks for game development
- Write up the findings
- Enquire about favourite game genres within a community (King's College London Mathematics School)
- Analyse the data that has been collected
- Selection of game idea and basic planning
- More in-depth design for the game including character model research, GUI suggestions etc.
- Selecting and learning the most appropriate game engine/framework for the scope of the project
- Planning the execution of different parts of the game
- Complete the game to allow beta testing and enough time for fixing bugs and acting upon feedback.
- Evaluation and conclusion for the development of the game

Organisation

For the purpose of this task, I will be using an online task-management platform called Trello[12], as seen in Figure 3.1, due to its simplicity and my past experience using the software.

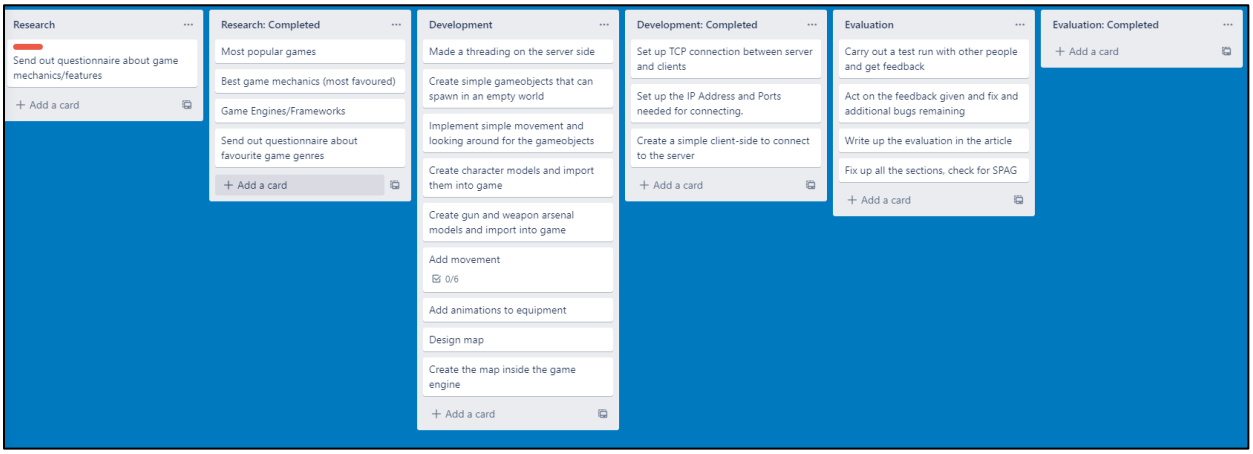


Figure 3.1: Trello board showing the tasks that will be done for the project

I have split tasks into three categories: Research, Development and Evaluation. In each of these categories, there is a tab with the raw name and the name followed by 'completed'. This allows me to keep track of the work I have completed and work that need to be done.

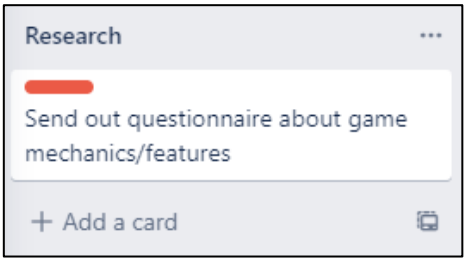


Figure 3.2: Research category of Trello Board

The reason I have specifically chosen Trello over other organising applications is due to the fact it offers features adding different label to each of the cards to give it a specified meaning. For example, in Figure 3.2, I have not completed the questionnaire for the game mechanics and features. However, since there is a red label on the card, I can identify that this task was originally planned to be done but was dropped.

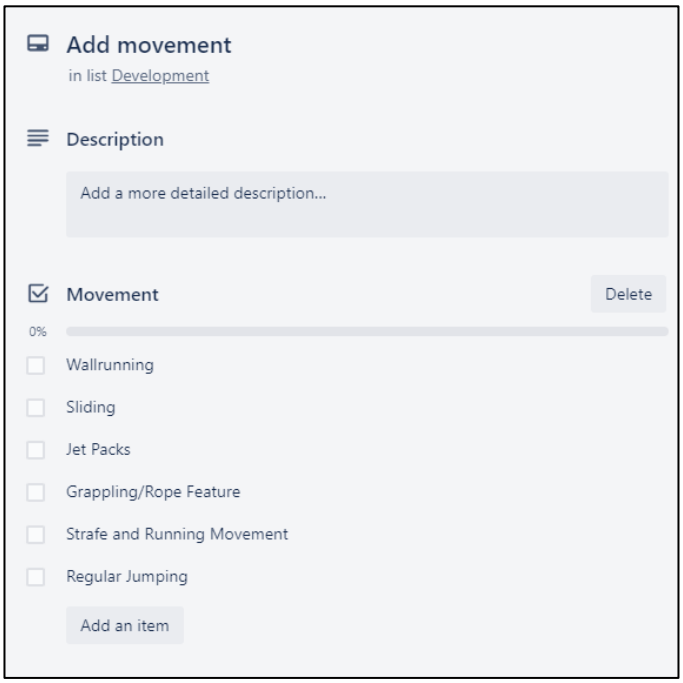


Figure 3.3: Expansion of the 'Add movement' card in the Trello Board

Additionally, with reference to Figure 3.3, Trello allows the feature to make a checklist within a task. This means main tasks can be split into smaller parts instead of making multiple cards and also helps with keeping track of which of the minor tasks has been completed along with those that need to be done. Additionally, subtasks can be grouped together within the same broader task for convenience and organisation.

Research

For the research section, I intend to research and talk about factors that make a perfect game. Additionally, I will look at popular games over the years, comparing and analysing any patterns or changes in the demand within the gaming industry, as well as perhaps looking into expert analysis on a particular game. Finally, I am going to research the different game engines and frameworks that are available to me to use for the development of the project and evaluate which of these are best suited for the designing stage (this decision will be made after looking at the data of the questionnaire which is in the Game Proposal section).

What makes a game 'Perfect'?

As mentioned by Stanton in his article 'Can there be such a thing as a perfect video game?', a game can only be perfect if no more features can be added to it[6]. This is because the game would already be considered complete and therefore does not need additional aspects to improve the game. But when we think of this in a practical sense, there is no game that can fit the criteria of being perfect because there will always be negative criticism. This is due to the variations of personal preferences and interests; some people may prefer first-person perspective games while others might enjoy top-down view games. However, one key factor which makes a game stand out is the uniqueness of the game. This could be through its mechanics, gameplay, or introducing a new game genre. Alternatively, little additions such as daily missions and levelling systems also help enhance the experience of the game[7]. These are all things game designers would need to consider.

It is important to consider how the player would feel and react to the game in front of them. A compelling storyline could be touching or relatable, but perhaps the pure gameplay experience might be enough to satisfy if it is well made with little or no bugs. If we consider a game where the mechanics are not robust, and the storyline is not grasping the attention of the player, then it would seem like there is no point in playing.

In the Game Proposal section, I will expand further on the criteria I will need to meet in order to achieve what can be deemed 'perfection' for the game.

Popular Games

What is considered the best game in the market? To approach this, I considered looking at the best-selling games in three different time scales: all time, in the last decade, and in the year 2021.

By viewing the best-selling games of all time[13], Tetris[14] stands at top with 495+ million downloads, followed by Minecraft[15] and Grand Theft Auto V[16] (GTA 5) as top three. However, by taking into account the release date, Tetris was released in 1984 while Minecraft and GTA 5 were released in 2011 and 2013 respectively. Therefore, perhaps it was expected for Tetris to be best-selling of all time with a very early release date when the video game market was minute and undeveloped.
Moving onto the last decade[17], Minecraft and GTA 5 remain in the top three with the addition of PlayerUnknown's Battlegrounds[18] (more commonly referred to as PUBG) in third place. The

reason for PUBG to become popular could have been the battle royale genre being relatively unexplored and bringing a new experience for players.
Lastly, we see a dramatic change in the rankings in best-selling games of 2021[19] as the top three positions are taken by Call of Duty: Black Ops: Cold War[20], MLB: The Show 21[21] and Resident Evil: Village[22].

By analysing the rankings, Grand Theft Auto V, PlayerUnknown's Battlegrounds, Call of Duty: Black Ops: Cold War, and finally Resident Evil: Village all share the shooting mechanic. This might suggest that shooters are a popular genre amongst the many people. However, we see the mechanic being used differently, based on the main genre of each of these games.

Grand Theft Auto V is considered an action-adventure game, where there is a wide range of weapons available. This could be to allow players to have more weaponry to play around with. Similarly, Call of Duty: Black Ops: Cold War also gives a variety of options with weapons. However, a levelling system where attachments can be unlocked and added to customise and improve the 'feel' of a gun is a primary feature in the game. In addition to attachments, weapon wrap variants (camouflages) let players choose pre-set colours, designs and patterns which can be added on guns and used during gameplay. The wrap variant has been a popular feature in many modern games including Valorant[23], Counter-Strike Global Offensive[24], DOOM Eternal[25] and many more. Ergo, it might be a sensible idea to include some of the mentioned features despite having minimal significance to the gameplay, due to the fact it may provide a sense of personalisation and hence improve the overall gameplay.

To get a better understanding on how games become popular, we can look at Kevin Webb's article on Insider[26] which talks about Fortnite: Battle Royale[27]. Webb identifies that one of the main reasons for Fortnite's popularity was its social factor, as it allows players regardless of their platform to play against and with each other. Moreover, he adds that the free-to-play factor made the accessibility to the game easier for everyone. Another reason for Fortnite's rise was due to their marketing strategies. By making business deals with other companies and franchises, many people quickly became aware of the game's existence. Lasty, Webb mentions the weekly updates, seasonal challenges and customisable gear which was attainable through in-game microtransactions were also big factors which attracted many of its fanbase. By looking at this article, I am able to examine that although it should be understood that a game without good mechanics cannot have become as popular as Fortnite, it should also be keep in mind that reasons outside of the gameplay such as the power of marketing can help branch out to a vast audience and become popular.

Game Engines / Frameworks for Game Development

Game engines are a software that use libraries and support programs to allow the development and design of games. There are many game engines online which are efficient for certain games. In this section, I will evaluate each game engine with their respective strengths and weaknesses along with my decision about which engine would be most suitable for the development of a game[8].

Construct 3

Construct is a web-based game engine that requires no code. This engine is most popular amongst indie game developers and over 50% of HTML5 games submitted to Kongregate have been made in Construct. Due to its no-code structure, it allows quick prototype creations and one of the best tools for non-programmers. Construct offers features such as timeline animation,

powerful engine, built-in CPU and GPU monitors, monetisation tools and many more. Exporting can be done to platforms including Windows, Mac, Linux, Android, iOS, and HTML5. However, Construct requires a monthly or yearly paid subscription to use.

GameMaker Studio 2

GameMaker is seen as a powerful engine for the creation of 2D games. With its visual scripting system, this engine is popular amongst beginners to games development since they need little or no knowledge at all of programming. However, there is an option to view and modify the code, which is written in GameMakers own custom language. The tool also allows support with animation, physics, and art. Despite being able to export to many platforms, there is a license fee to do so.

Unreal Engine

Unreal Engine is one of the most popular game engines that is used for modern AAA-rated games that uses breath-taking, realistic character models, animations, and texture detail. The engine primarily uses C++ as its supported programming language but allows a tool called 'Unreal Blueprints' that allows creators to see a visual representation of the logic. The engine is powerful yet complicated to understand and use, especially for those with little to no knowledge in game development and programming. However, Unreal Engine is free to download and use with no initial payment needed. Although if the game's lifetime gross exceeds $1 Million USD, then a 5% royalty is paid to Epic Games, the creators of Unreal Engine.

Unity

Unity is one of the most famous game engines in modern game design. Described by its developers as a robust ecosystem where imagination can run loose, the engine allows the creation of games in 2D, 2.5D, 3D, Augmented Reality, Virtual Reality, and mobile. The engine uses C# as its supported programming language and integrated-development tools for certain features like animation, art, audio, monetisation etc. Unity is available for free for the purpose of personal use but would require a subscription for industrial usage.

Godot

Godot (formerly recognised as Larvotor, Legacy, NG3D and Larvita) is an open-source software for 2D and 3D game development. The engine allows scripting languages including C++, C#, GDScript (a custom language similar to Python) and finally a visual scripting tool. With GDScript, Godot allows those with no programming knowledge to understand the logic as well as learn as they use the engine. With this engine, exporting to desktop platforms, mobile platforms, consoles, and web platform are all possible.

After reviewing the survey in the Game Proposal section, I will be able to come to a decision on which game engine to use based on the game I plan to make.

Game Proposal

In this section, I will go through the survey that I completed at King's Maths School, representing this using a graph and listing ideas and my thoughts for the potential build of the game.

Survey

In order to gain a good understanding of the type of games played within my chosen small community, I carried out a questionnaire regarding the most favoured game genre. The methodology of data collection involved asking a sample for their top three game genres from the list of: sandbox, shooters, role-play (RPG), simulation, puzzle/party, action/adventure, horror, platformer, multiplayer, and rhythmic. This sample consisted of close friends, students, and teachers, all of whom attend King's Maths School. A tally chart was used to keep track of the number of students/teachers that preferred each genre. The reason for allowing a selection of three genres is due to modern games having a mixture of genres to target a larger audience. Additionally, this allows the freedom of choosing multiple genres rather than having to evaluate the most favoured genre out of the list.

Game Genres	Tally
Sandbox	4
Shooters	13
Role-Play (RPG)	2
Simulation	4
Puzzle/Party	6
Action/Adventure	10
Horror	0
Platformer	2
Multiplayer	14
Rhythmic	2

Figure 5.1: Results of the data in a tally chart

After having a table of results, I decided to visually represent the data using a pie chart since it would be easier to analyse and compare the popularity of each game genre.

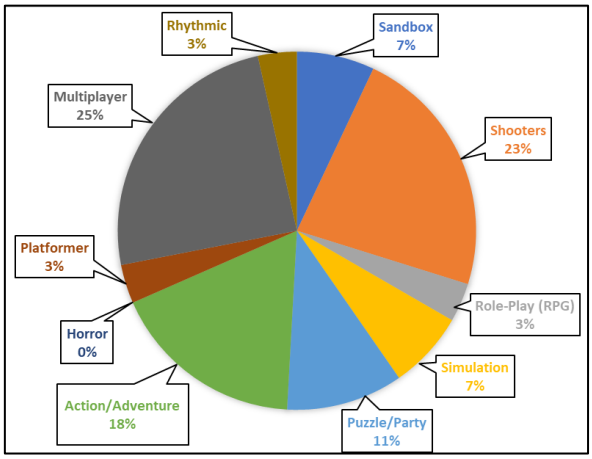


Figure 5.2: A Pie Chart representing the data from Figure 5.1

Observing Figure 5.2, multiplayer clearly has the biggest sector and therefore means it was the most popular genre at King's Maths School. Shooters is close in second place, which may suggest that a multiplayer-shooter game might be the best game as per the data. Although action/adventure has come third place, I personally feel like it doesn't need to be incorporated as much as the other two genres, although it might be something to consider when expanding on the game if there is time.

Ideas / Plans / Concepts

After the evaluation of the questionnaire, I concluded in making a first-person shooter (FPS) with a multiplayer aspect. For this, I had to consider many aspects to the game and plan.

Basic Concept

Firstly, I had to decide what type of FPS I was going to make. Was it going to be team-based? How many players on each team? What would the end goal?

I determined that my game would have the round-based factor. Adding to this, due to the overpopulation of team-based shooters, I wanted to make my game to test an individual's skill. Also, I wanted my game to be fast paced, keeping players on alert and their adrenaline rushing.

Keeping all these in mind, I finalised that the game would be a three-way free-for-all. The objective is to kill each player on the playing field gaining points for each kill as well as remaining as last one standing which would also give points. The inspiration of the point system came from Fortnite: Battle Royale[27] with their Arena game-mode. After each round, the teams would rotate in order to allow all players to compete for their respective teams. This is how I aim to achieve the individual skill factor. However, I must consider how the order of the team will be decided. Would the team decide upon themselves? Will there be an ordering system based on skill-level? Or could the order just be randomised?

At the end, the team with the most points would be deemed the winners of the game. Additionally, the scoring system will give bonus scores for each players levelling system if they complete challenges or from their performance from each of the rounds. For example, killing all the enemies, getting the most kills in your team, most kills in the game, most rounds won etc can all be achievements which test a player's skill and give additional experience points. This can be decided later on during the development of the game.

Player Movement

For the player movement, I have decided to take inspiration from Titanfall 2[28], Call of Duty: Black Ops 3[29] (BO3), and Apex Legends[30] with its unique movement of mantling and wall-running. Since each of these games have been given praise for its movement mechanics and my experience playing BO3 and Apex Legends has been enjoyable, I would like to implement these features.

An idea that I initially planned to incorporate was the leaning feature from Tom Clancy's Rainbow Six Siege[31] and double jumping from Hyper Scape[32]. However due to the complications with implementing these abilities, I decided that the features I was already thinking of incorporating were enough. And additionally, bombarding a game with too many movement features could confuse players and discourage new players from wanted to learn the game.

Map Designs

In terms of map designs, I am considering only making one map due to the time restrictions of the project. I have chosen to use the map Nuketown from the Call of Duty franchise (see Figure 5.3) as the building block for my own map design. This iconic map is very popular amongst the Call of Duty community. Metzger, in his review[33] on the maps, states 'it is [the map] tons of fun.' He expands by adding that the map allows enough gunplay to satisfy due to its relatively small layout. In addition, the buses placed in the middle of the maps reduces the number of angles that a player can be shot from, thus having a balancing element for those that like to roam around the playing-field.

Figure 5.3: Nuketown map from Call of Duty: Mobile[34]

Gunplay

Regarding the gunplay, I am considering implementing guns with no travel-time or projectile motion. This essentially means that from the moment the player shoots with their gun, the bullet has close to infinity speed such that it will reach an obstacle or player instantaneously. This is because it will make hit-detection less complicated. Additionally, I will split the arsenal into sections such as sniper, assault rifle, submachine gun, shotgun with the ability to mix and match between a Glock (a semi-automatic pistol with fast fire-rate) or a Desert Eagle (a semi-automatic pistol with slow fire-rate but high damage).

Depending on if there is spare time at the end, I might learn to create textures so that weapons have their own unique skin variant rather than a plain black gun model.

Character/Object Models

Due to my lack of knowledge using Blender (an open-source, 3D, graphics software), I am debating whether I should learn how to use the application to make character and object models or whether I should use resources on the internet.

Using Blender would allow me to have full customisability on the designs of each of the models, but at the same time it would take multiple hours. This may be a concern looking at the scope of the project and the relatively short period of time. Another argument against using Blender is the fact many game studios have their own team dedicated to producing character models since they require attention to detail and I am attempting to make the entire game myself.

By using character models found on the internet will mean efficiency in the making of the game as well as allowing more time to concentrate on the mechanics and gameplay. But since I am not making the models, there may be mismatches ruining the aesthetic of the game and may put certain objects/characters out of place.

Utilities / Abilities

A default in most first-person shooters are explosives and stun grenades in many forms. Therefore, it would be a good addition to add these into my game. Implementing this should be relatively easy since both explosives and stun grenades can be seen as projectiles but each one with its unique trajectories, parameters stating whether they attach to objects and the impact (damage) it causes to other players.

Game Engine Selection

Due to my past history in coding using languages including C++ and C#, I am not limited to the selection of game engine. However, the limitations of GameMaker Studio 2 and Construct 3 only allowing production of 2D games would mean I would prefer not selecting those engines since my game will require 3D graphics in order for a first-person shooter game. So I have concluded in Unity because I have experience using it before and I feel more comfortable using the interface for Unity rather than Unreal Engine.

Additional Factors

- Since I am making a multiplayer game which I have no knowledge or experience in, I will have to dedicate time into learning networking, server-side programming and setting-up a server to allow clients (i.e., the players) to play.
- In order to save time, I might look at already existing code for movement, guns and abilities and add this to the project.
- I will need to add a main menu to the game that will allow the player to connect to the server and ready themselves before a game starts.

Measuring the perfection of the game?

Once the game is finished (or midway through development depending on how long the designing will take), I will once again reach out to students and teachers from my chosen community and allow them to try out the game.
They will be asked to give their feedback on various factors (whether this is through Microsoft Form or just written criticism will be decided). I can then see by evaluating the reports whether I have achieved my target of making the perfect game.

Since perfection is an abstract idea, I will define it slightly differently for this project. It will be based on how positive the responses are and if it meets a certain threshold (for example an 85% satisfaction rate response from students and teachers), then it can be deemed as perfect.

At the current moment in time, I will look for responses regarding the following aspects:

- Gunplay – the shooting mechanics
- Movement – does it feel natural?
- Visuals – are the graphics aesthetically pleasing to look at?
- Experience – is the game nice to play?
- Specific features – there will be sub-questions that will talk about the different features implemented
- Map design

Development

In this part of this article, I will show the methodology and chronological progression of the game by using screenshots and explaining various features that are added each time. It will show development from start to the end of the design.

Multiplayer Networking

For a functioning multiplayer game, there needs to be a server to allow clients (which in this case are the players) to connect to, such that information in the form of data packets can be communicated between the different players via a central server.

Without much knowledge in networking, I firstly wanted to understand more about how data is transferred through the internet or local area networks. I read an online book called *Aspect of networking in multiplayer computer games*[35]. It brought up the fact the limiting factors in networking which includes bandwidth, latency, and the hosts' computational power. It also gave information on the different communication architectures that can be used including peer-to-peer, client/server, or server-network. In addition, the book talks about compensatory techniques to reduce the amount of networking involved by using methods like compression, interest management, and dead reckoning (prediction of data transfer). Lastly, the book was very informational on security in online games being a major concern by mentioning the existence of packet and traffic tampering (more commonly known as wall-hacking or aim-bot), information exposure (gaining access to hidden game data like player status) and design defects which allow exploitation and cheating to occur that was not intended.

To get started, I followed along to a YouTube video series by Tom Weiland in order to set up the client-server connection [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46].

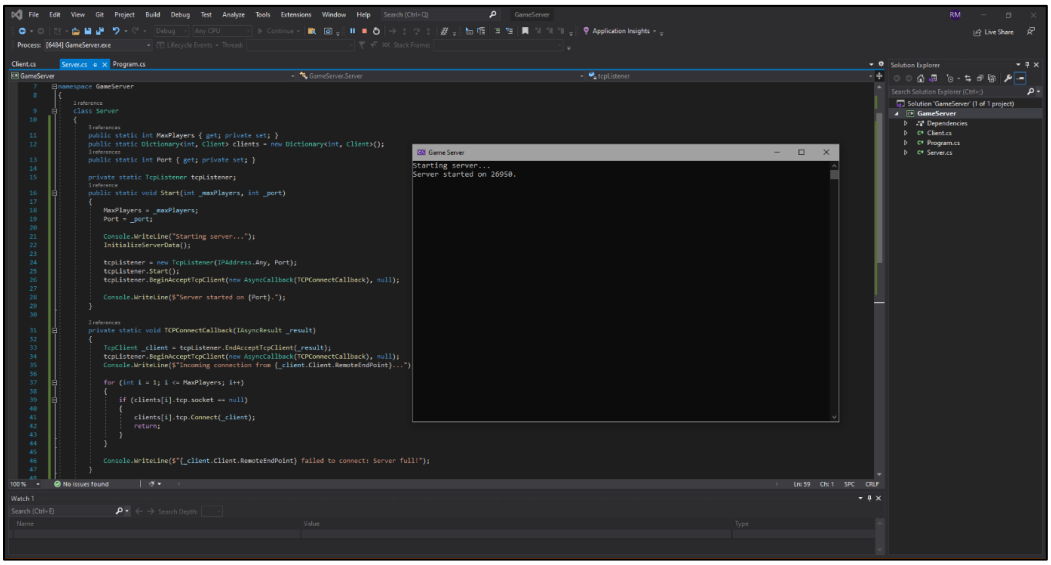


Figure 6.1.1: Initial set up of the server

The code in Figure 6.1.1 shows the initialisation of a TCP connection between an arbitrary client which can connect to this server by using the local IP and the Port number which in this case is 26950, although it could have been any number between 1 and 65536 inclusively. The communication is setup to be asynchronous since there will always be a delay with connections from a server to client and vice versa. In addition, there is a check to make sure in the case of a failed connection, a try-catch method will handle the error and prevent the program and server fatally crashing. With the server running, the next task was to get a client to connect to this server. In my case, the client-side can be simulated through Unity.

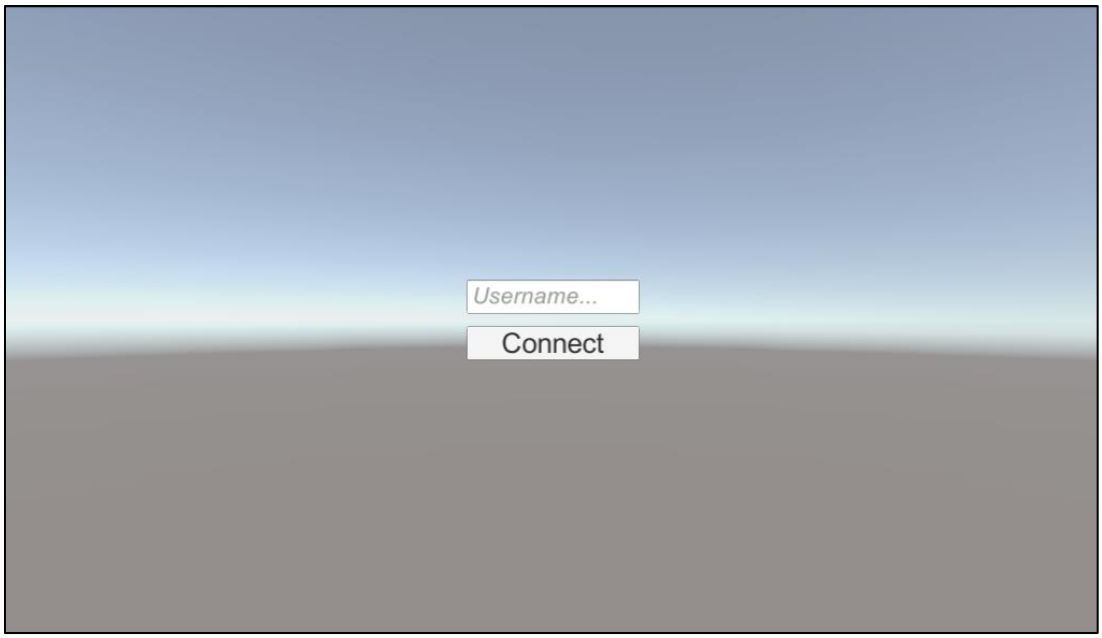


Figure 6.1.2: Client-side view just before connecting

By creating a menu UI, I was able to add an input field and a button as seen in Figure 6.1.2. At the current moment, the input field has no functionality. However, the connect button would automatically attempt to connect to the server I made earlier through the same Port.

```
    stream.BeginRead(receiveBuffer, 0, dataBufferSize, Rece

    ServerSend.Welcome(id, "Welcome to the server!");
}

3 references
```

Figure 6.1.3: A section of code written when there is a successful connection

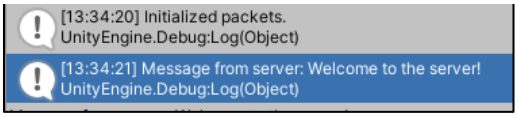


Figure 6.1.4: Unity log when a scene is running

In the 'ServerSend' class, the method called 'Welcome' simply sends a desired message through data packets to the client in which it can be displayed. In Figure 6.1.3, we can see that the message entered as the argument to the method was "Welcome to the server!". Consequently in Figure 6.1.4, I can confidently say the one-way communication from the server to the client is very much successful since the message has now been displayed in the Debug.Log section.

The current problem is that the server does not check whether the client has sent any data once it has started running. Therefore, there needs to be a way for the server to know when a client has sent data. The only way this can be possible is if the server has occasion checks to see any received data. A similar analogy is checking whether the mail has been successfully sent through the post. But then there is another issue. Since the framework being used on the server side is purely C#, it does not have a built-in update method like Unity.

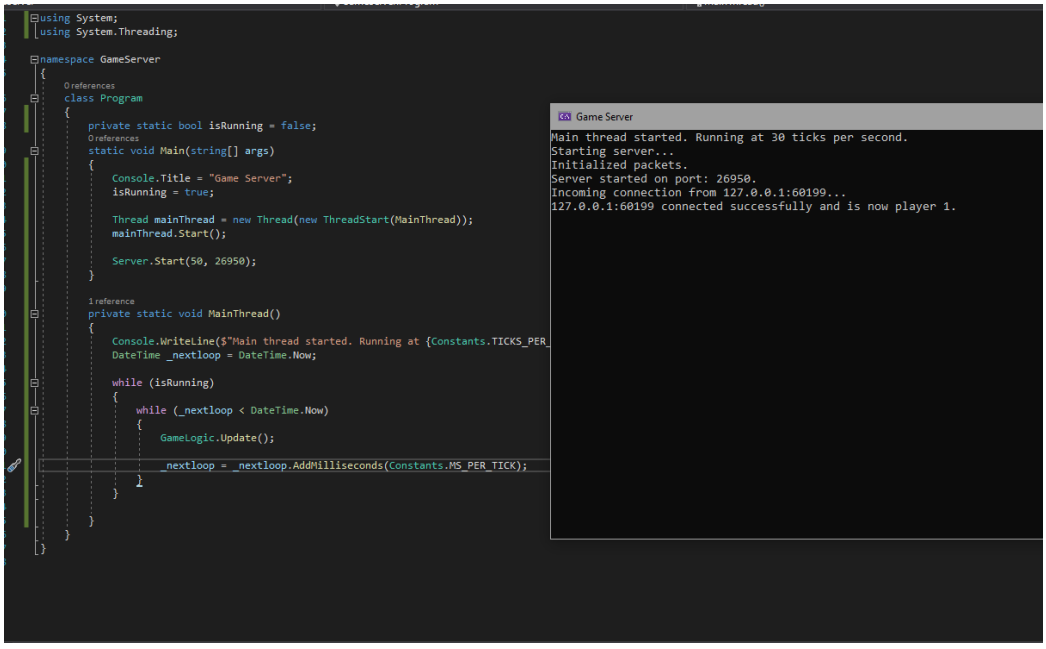


Figure 6.1.5: Section of code showing the threading and a console window showing the threading successfully working

A way to get around this issue is by using a process called 'threading'. This is essentially executing instructions by utilising the scheduler (which is part of the operating system).
The way threading has been used in the above code is by implementing a nested loop, in which the outer loop runs based on whether the server is running while the inner loop uses the timings of the computer to carry out game loop at regular intervals.
After this was completed, I added another method that displays when a user has connected as well as their identifier which is this IP address for now.

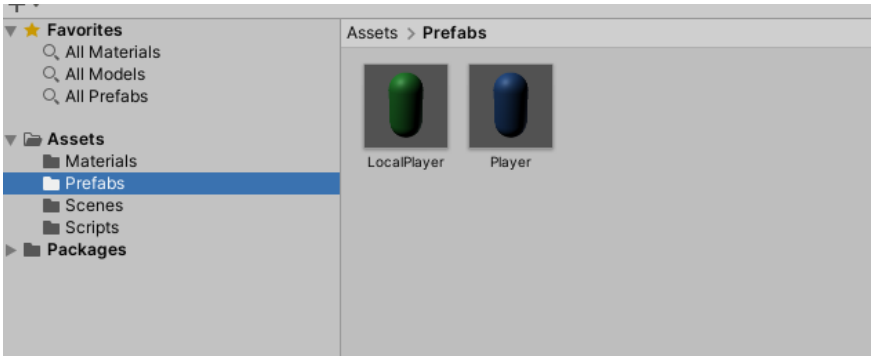


Figure 6.1.6: Prefabs folder in Unity

Now I wanted to get a simple scene built so that I could simulate simple character movement as well as the ability to look around. I firstly created two prefabs, one of which models the local player (i.e., the character model that is being controlled by the client) and a model for other players. For simplicity, the character models are 3D bean shapes, and the distinguishing factor between them is their colour.

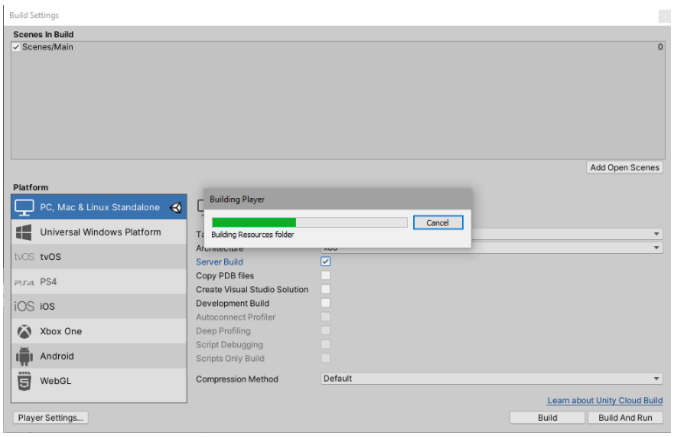


Figure 6.1.7: Building the current scene

For a game to have a multiplayer functionality, it must allow more than one player to connect and play at once. To simulate this, I compiled and built a test version of` the game as it is. This meant I could open it in another window as well have the engine's debugging version both running at the same time.

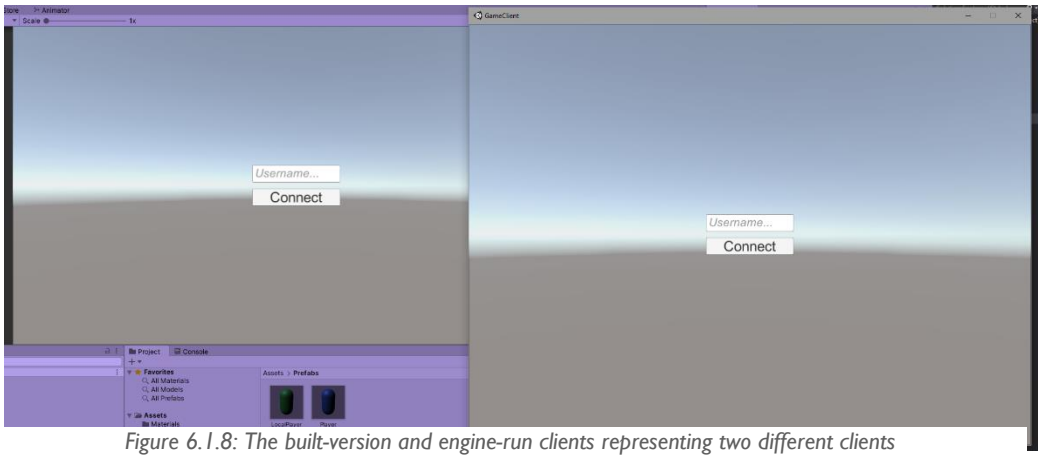


Figure 6.1.8: The built-version and engine-run clients representing two different clients

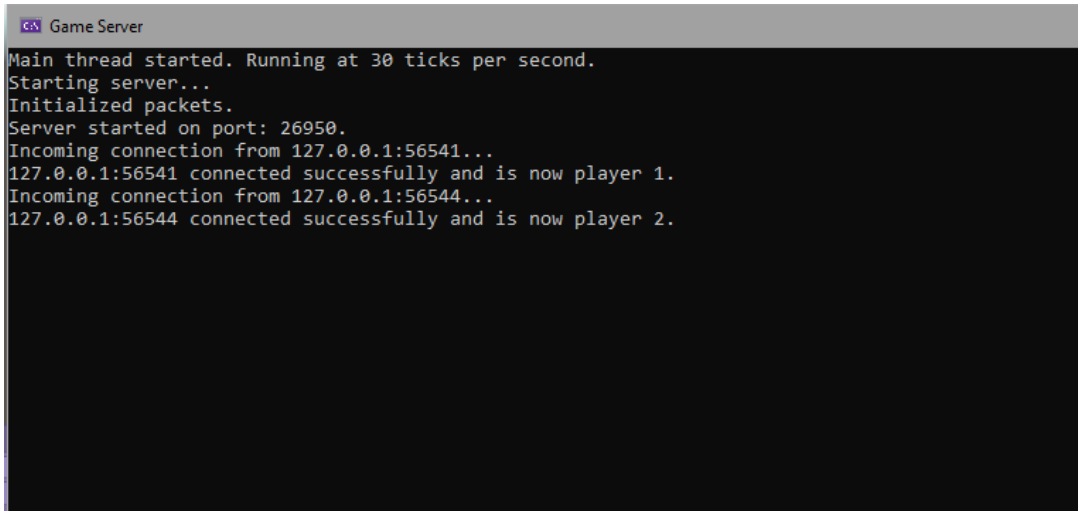


Figure 6.1.9: Console window showing the two clients connecting to the server at the same instance

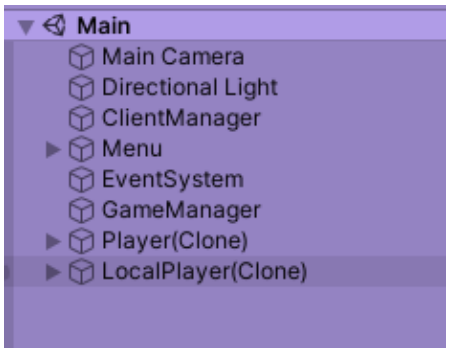


Figure 6.1.10: Hierarchy section of Unity

I was extremely happy to see that the two clients were able to connect successfully. As well as this, it was astounding to observe the server (in Figure 6.1.9) and the hierarchy section of the game engine (in Figure 6.1.10) recognising this connection. All that was left is configuring the necessary properties and information to send between the server and clients. However, this has to be done once the game is completed, due to the fact I have not yet thought about what sort of information will need to be sent.

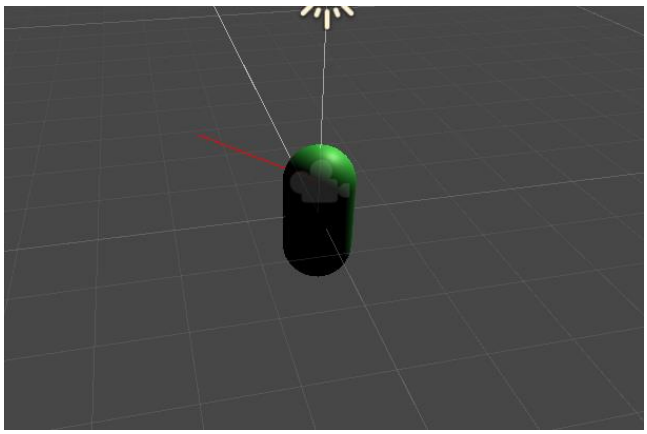


Figure 6.1.11: Local player in editing mode while engine is running

Penultimately, I added the ability to let the player to move and look around in an empty three-dimensional space. Looking at Figure 6.1.11, it can be seen that the character can move since they are no longer at the 3D space origin. Additionally, I put a red ray in the direction the player is looking at for designing purposes. It should also be noted that the movement and orientation of the player is sent to the server by converting its vector position and quaternion rotation into separate numerical values for x, y, and z co-ordinates.

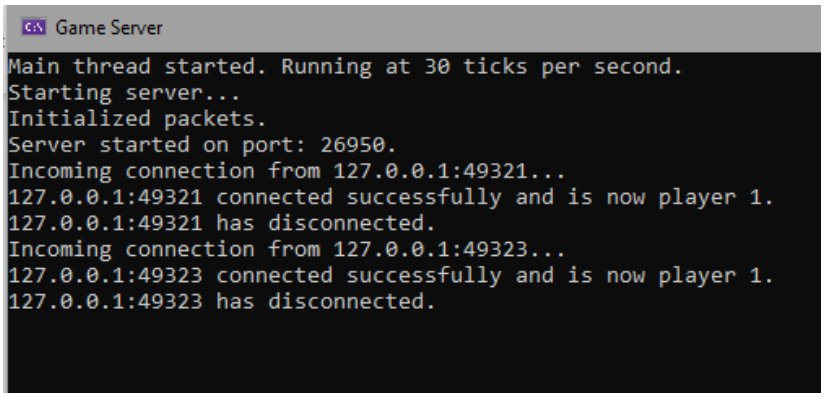


Figure 6.1.12: Console window after player 1 disconnected

Finally, the only problem left in terms of multiplayer networking was the disconnecting functionality. At the current moment, there is no option to allow a player to leave and disconnect from the server which means force-closing the game is the only option. Doing so disconnects the player on the client-side, and the server may recognise the disconnection, however there is no method to remove the socket which connects the two. So after adding relevant code, players now disconnect without an error message being displayed, as well as letting the user know that a player has disconnected by outputting the update in the console window, which can be seen in Figure 6.1.12.

At the moment, the server is being run on the local machine and hence the IP address that was connected to was "127.0.0.1". However, in a commercial multiplayer game, the server would be set up in a remote location in which players connect to the external IP address of that server. To replicate this, I decided to set up a server build within a Linux instance using Amazon Web Services[47] (AWS).

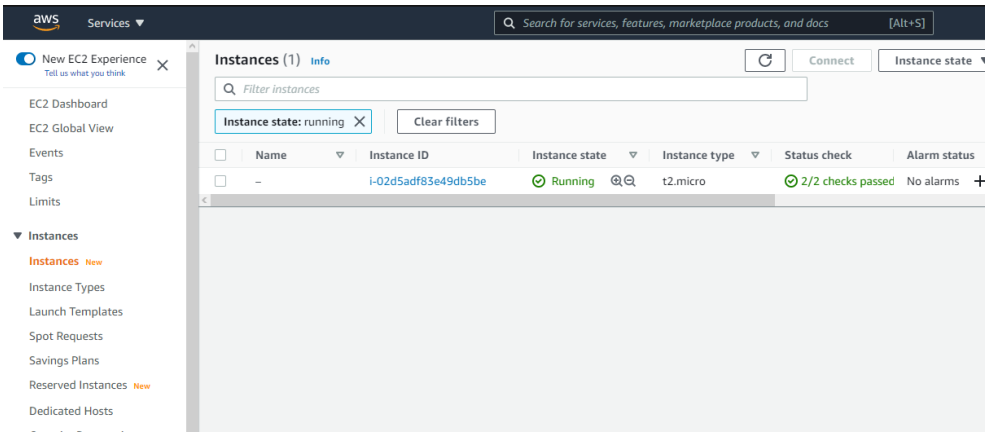


Figure 6.1.13: AWS dashboard showing the Linux Instance running

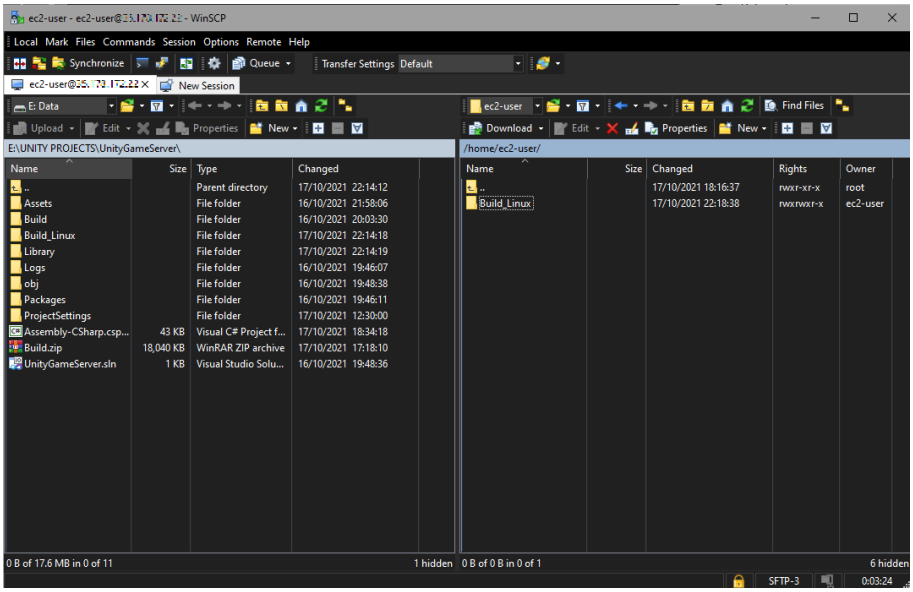
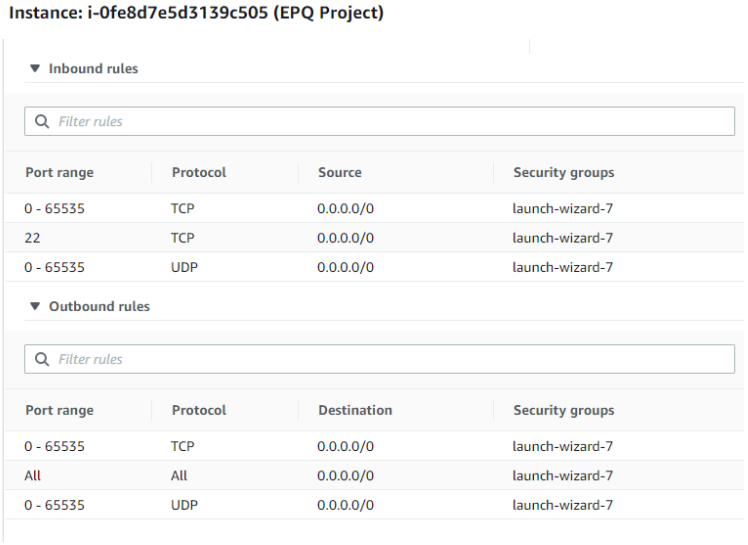


Figure 6.1.14: WinSCP Window

After creating a Linux instance as seen in Figure 6.1.13, I transferred the server build files to the instance using an application called WinSCP (Figure 6.1.14). The purpose of using WinSCP is to allow files to transfer through TCP via port 22 which is not possible without a generated PuTTY key, which carries out all these tasks in an efficient way rather than manually doing it myself.



Instance: i-0fe8d7e5d3139c505 (EPQ Project)

▼ Inbound rules

Filter rules

Port range	Protocol	Source	Security groups
0 - 65535	TCP	0.0.0.0/0	launch-wizard-7
22	TCP	0.0.0.0/0	launch-wizard-7
0 - 65535	UDP	0.0.0.0/0	launch-wizard-7

▼ Outbound rules

Filter rules

Port range	Protocol	Destination	Security groups
0 - 65535	TCP	0.0.0.0/0	launch-wizard-7
All	All	0.0.0.0/0	launch-wizard-7
0 - 65535	UDP	0.0.0.0/0	launch-wizard-7

Figure 6.1.15: Inbound and outbound rules for the Linux instance created

Although the game server was running successfully in the remote server, I was running into problems with clients connecting to the server. This was because the inbound and outbound rules did not allow connections through to the Linux instance whatsoever. To do so, I had to add the rules, seen in Figure 6.1.15, for both TCP and UDP protocol communication as well as allowing the source and destination to be 0.0.0.0/0, which allows all IP address to be part of the connection. This can be restricted to only certain IP addresses for security purposes however there is no need due to the scale of the project.
And with this completed, the networking aspect of the game is done which means I can move onto development of other sectors of the game.

Pick-ups and Projectiles

The pickup implementation was fairly simple. It included setting up the locations of the pickups on the server side. On the client side, the positions of the pickups would be known from the ClientHandle class and the rotation and bobbing animation was done through coding.

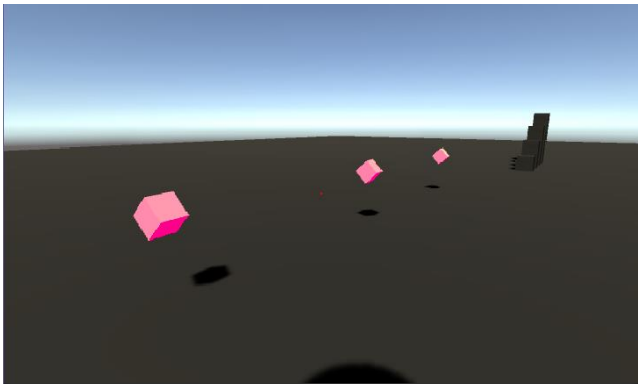


Figure 6.2.1: Template for pick up items

Later in the project, the texture/shape of the pickups seen in Figure 6.2.1 can be changed for various power-ups and the logic can be added subsequently.

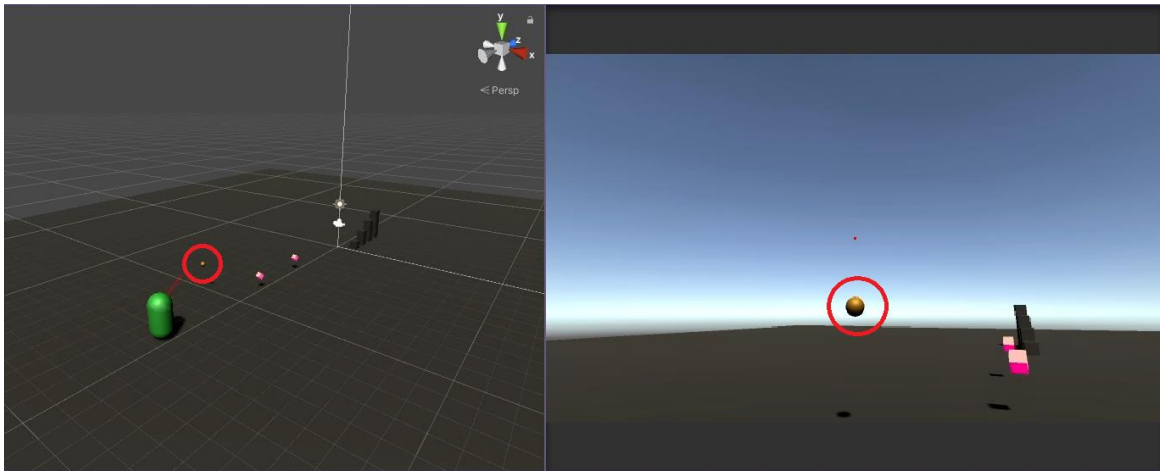


Figure 6.2.2: Template projectile being thrown

Secondly, I added the ability to throw projectiles which are circled in Figure 6.2.2. At the current moment, the projectile is an orange sphere, and the explosion animation is not refined to look like an accurate explosion. However, with the development of the game, I will alter the projectile to include flash grenades, frag grenades and Semtex. This will include logic for the time elapsed from the moment the grenades are cooked to when they will explode as well as whether they will stick to surfaces. Another problem with the current projectile mechanics is that the trajectory is lower than what I would have liked it to be. Luckily this alteration is very easy, all that needs to be done is change the initial force and angle the grenade is thrown.

Movement

Due to the time and difficulty of producing a fully functioning movement for a player, I instead opted to using a pre-existing project created Will-S-Dev on GitHub[48](seen in Figure 6.3.1), which includes movement, jumping, wall-running, mantling, and wall-jumping. However, I was not completely happy with the existing project and decided to add my own additions to simplify the movement and changing the wall-running feature so that the player stays on the wall rather than have an arc-shaped path on the wall and falling off after moments later by just removing the gravity component while wall-running.

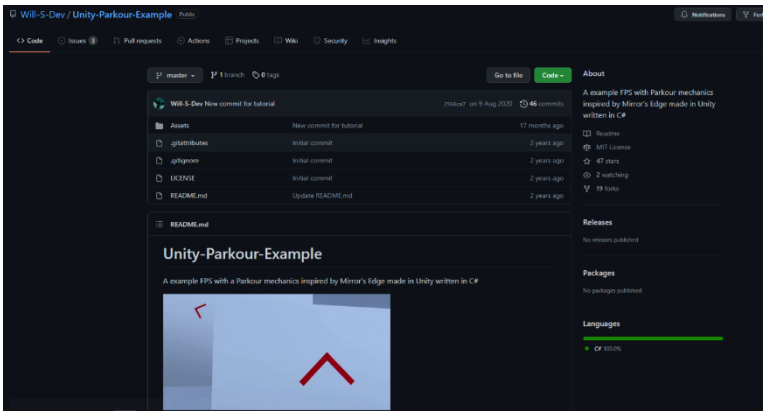


Figure 6.3.1: Parkour project on GitHub from Will-S-Dev[48]

However, one problem I came across was jittering while moving and looking around. This means that the rendering of the scene was not smooth and the times between the render was not consistent and therefore resulted in stuttering.

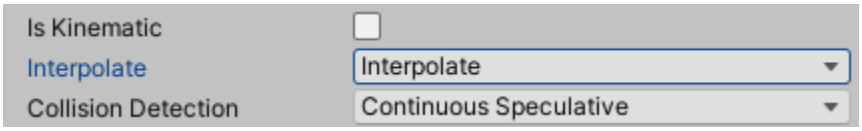


Figure 6.3.2: Interpolate setting for player RigidBody

However, this issue was easily fixed by changing the interpolating setting of the RigidBody to interpolate. And therefore, the game engine compensates for the jittering.
The way the Player-Controller works in Will-S-Dev's project is by using invisible colliders around the player to detect when there is a wall overlapping the circumjacent area of the player. This is for the wall-running feature to function well. Additionally, the looking around is controlled by obtaining the raw input of the mouse, which gives feedback on the direction the player wishes to look towards and converts this into a quaternion angle for horizontal rotation on the player and vertical rotation on the camera angle.

Gunplay

There are two aspects of gunplay within gaming: the gun model and programming the firing properties. Gun models are simply made using a 3D modelling software such as Blender. Unfortunately, due to my lack of experience and length of time required to get used to the software, it was an unrealistic expectation to make the models myself. Instead, I used existing models found on Unity Asset Store. However, there were limitations with the file I chose. Since there was no expense of using the models, the guns are 'low-poly', meaning a polygon mesh with a low number of polygons in 3D graphics, ergo the shape of the model is not entirely smooth.

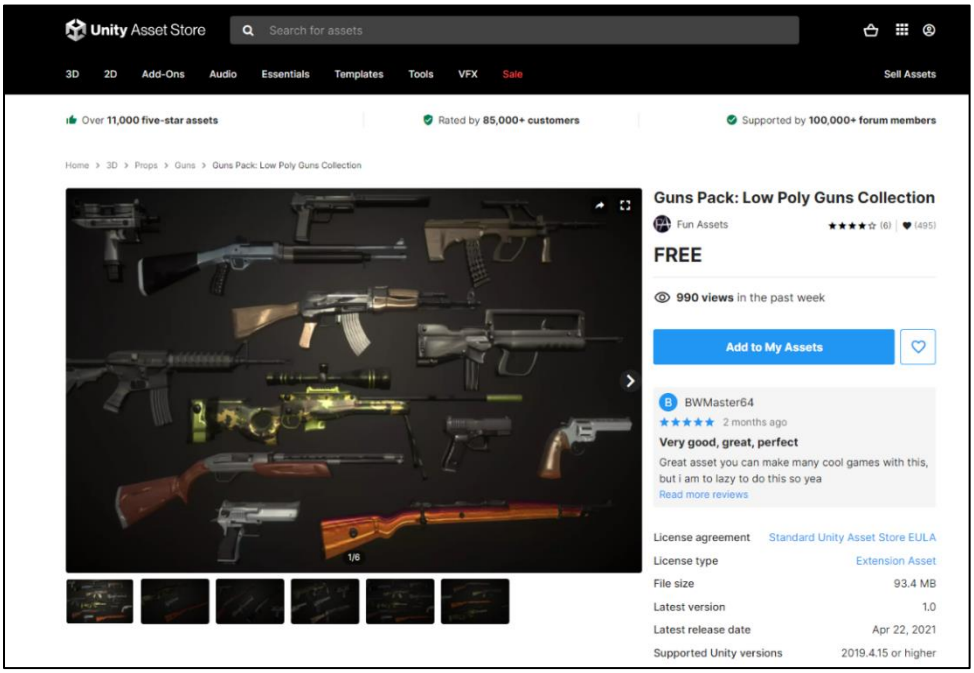


Figure 6.4.1: Unity Asset Store displaying gun pack files ready to add to project.

The next task was programming the guns to shoot at the user's input and also positioning it such that it follows the player's viewpoint. In Unity, adding a game object as the child of another object in the hierarchy allows the child game object to move relative in space to the parent object. So as seen in Figure 6.4.2, the gun model stays within the view of the player giving the authentic first-person feel.

Figure 6.4.2: First person perspective for player

Now the last thing to do was add a script to the guns to allow them to shoot. I made the script so that it was versatile with all the possible gun properties, for example including the option of choosing whether the gun was a burst rifle, the size of the magazine, shoot speed etc. This can be seen in Figure 6.4.3.

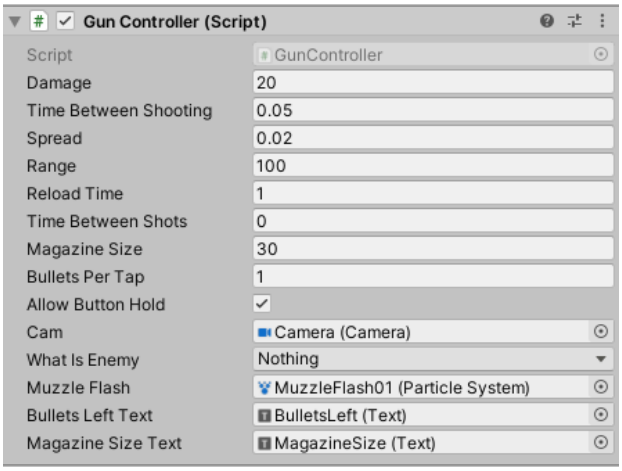


Figure 6.4.3: Gun Controller properties for an assault rifle

Map Design

Before starting to develop the map using an in-built feature called ProBuilder, I firstly wanted to plan out how the map would look. Since the game was going to be three-way, I decided on a triangle-based shape, with the addition of corners removed such that the angle between each wall was not acute. A sketch of top-down view of the map can be seen in Figure 6.5.1.

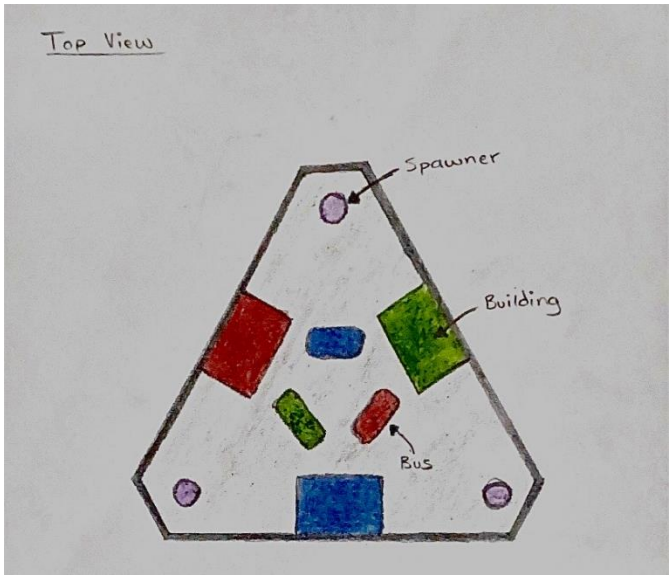


Figure 6.5.1: Sketch of a map

Since mentioning about using Nuketown as an inspiration, I have placed buildings between the players spawn positions and also buses in the middle of the map recreating the classic layout of the popular map. These locations were specifically chosen to allow players to move around a lot making the most of the movement mechanics but also giving players cover. The colours I have chosen are the primary colours for computer screens (red, green, and blue). So next was recreating the same layout of the map in ProBuilder and the result is seen in Figure 6.5.2.

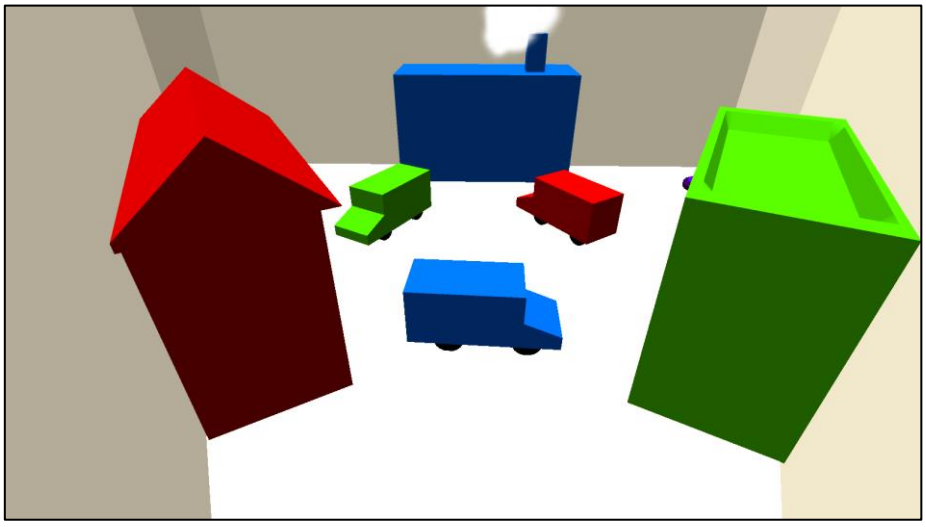


Figure 6.5.2: Scene showing the modelled map

Evaluation

Due to the time constraints, there was no time to build upon certain features such as the leader-board display, the heads-up display, more detailed 3D-models, environment audio and incorporating pick-ups and projectiles. However, the game was playable after loading and wiring the map, gun-controller scripts, and movement controller scripts to the multiplayer scripts.

By distributing the game via a downloadable link, I was able to give voluntary beta testers a chance to play the game. The connection to the server was already setup through the code so all that was needed was to press the connect button as seen in Figure 6.1.2. After playing through the game for a couple of hours, I proceeded to ask each of the beta testers to give their feedback on various aspects in the game through Microsoft Forms, giving a rating from 1 to 5 on each component with 5 being most satisfactory.

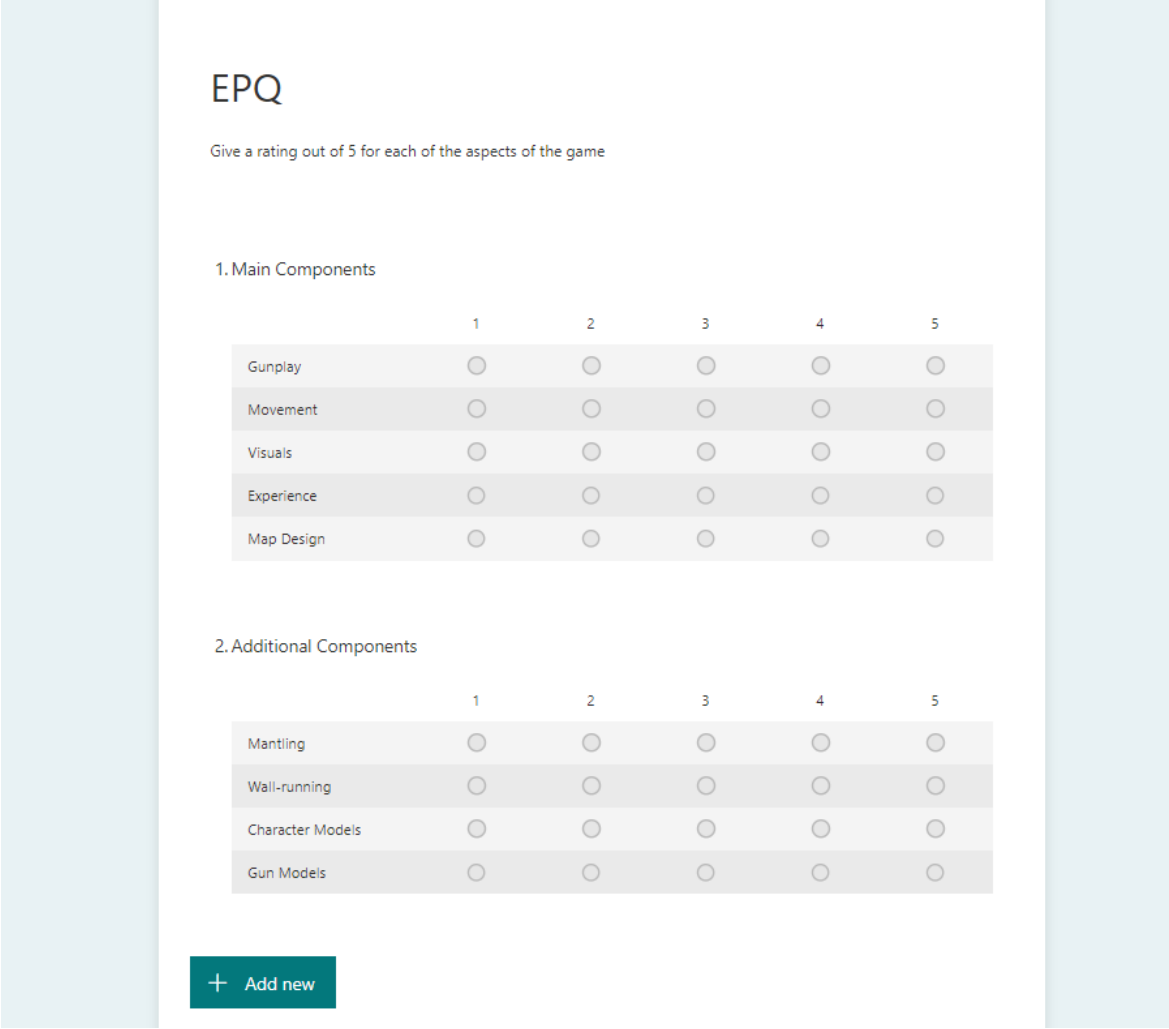


Figure 7.1: Microsoft forms showing the questionnaire or post-testing feedback

After collecting the data, I exported the results to an excel spreadsheet (see Figure 7.2) and made calculations for the average of each component, the average overall, and then a conversion to a percentage representation of the overall feedback on all the aspects combined.

	A	B	C	D	E	F	
1		Gunplay	Movement	Visuals	Experience	Map Design	Ma
2		5	5	4	5	4	
3		4	5	4	5	5	
4		5	4	4	5	4	
5		4	5	4	5	4	
6		4	5	5	4	5	
7		5	4	4	4	5	
8		4	4	5	5	5	
9		5	4	3	5	5	
10		5	5	5	4	5	
11		5	4	4	5	5	
12		5	5	5	4	5	
13		4	4	5	5	5	
14							
15	Average (each component):	4.58	4.50	4.33	4.67	4.75	
16							
17							
18	Average:	4.63					
19	Percentage:	92.593	(3 d.p)				
20							

Figure 7.2: Microsoft Excel spreadsheet showing the result data

Looking at the spreadsheet, gun models was given the best average rating. In retrospect, this makes sense because the gun models were created by an external source who has experience making 3D models. Additionally, looking at the percentage of positive feedback 92.593% which is greater than 85% I had originally suggested to be a boundary line to determine this game as perfect, meaning I have statistically accomplished what I had planned to do.

But despite the triumph in results, I personally would say the game is not yet perfect. This is because by recalling Stanton's article[6], a game is deemed to be perfect if no additional features can be added to it to make it better. However, that is not the case since there were many features that I was unable to incorporate due to time limitations.

Despite personal unsatisfaction, I believe the completed version of the game has potential so moving forward, I am thinking of adding several changes including:

- Going from three-way to four-way or even five-way to increase the amount of action
- More maps and more details on 3D models
- Grappling guns and jetpacks to add a flying aspect
- Leader board and heads up display functionality
- Allowing users to create accounts which stores their levels and setting preferences
- Expanding to different platforms to make the game cross-platform
- More sounds

With these additions and more testing, I may potentially release the game on the popular video game distribution service Steam[49] and see the views of people outside my chosen community.

Personal Evaluation

Moving away from the game itself, completion of the project has positively impacted my approach to long-term goals in general. My organisation skills are much better since I have more familiarity with using Trello[12] and all its provided features compared to before when I would use the basic ones. Additionally, by building upon my existing knowledge of game development, I am now more confident in the development of making games and can use my experiences to make the designing stage quickly and more efficiently.

However, I also faced many challenges throughout the project. This included the lack of motivation to continue with the project for the prolonged period. I saw myself get easily bored with the tedious tasks and often procrastinate. But fortunately, the allocation of time for each task was lenient enough to not be problematic. And giving regular breaks to the project helped refresh my mind before spending time on the game again. In addition to this, another challenge I faced was not following along with the planned schedule. Instead, I would complete the tasks I felt like doing at that moment in time. Since in the end I was able to complete the tasks necessary, I would say this was not much of a concern.

Knowing what I do now after the entire process of making a game, I might consider giving myself a more realistic workload so that I could finish a game in the time limit. Perhaps switching to 2D might have significantly made the entire development of the game much easier due to the fact the art, the mechanics, and the user interface displays would all be in 2D rather than 3D. But at the same time, this would mean not being able to incorporate wall-running and mantling which were key mechanics I wanted to add. Perhaps in the future I may re-evaluate what I decide to do so that the end goal seems more within the scope of the time frame.

Moving forward, I think I have the capability to use my expertise in other game genres. In addition, I may learn how to use Blender to create my own 3D models giving me freedom of the theme of the game and have better control in the map design. Also, since game development has been a career path I have been considering for a while, I now have a better insight of the what the industry and making games is like.

Bibliography

1. From: Beluga. *Examples of Minimalism in Video Games.* [Image] Available from: https://medium.com/@jsuyasri/examples-of-minimalism-in-video-games-87d1872b6055/ [Accessed: 14th September 2021]

2. Google. *YouTube.* [Online] Available from: https://www.youtube.com/ [Accessed: 22nd September 2021]

3. Amazon. *Twitch.* [Online] Available from: https://www.twitch.tv [Accessed: 22nd September 2021]

4. Taylor DB, Chokshi N. *The Fortnite World Cup Winner Is 16 and $3 Million Richer.* [Online] Available from: https://www.nytimes.com/2019/07/29/us/fortnite-world-cup-winner-bugha.html/ [Accessed: 22nd September 2021]

5. Getty Images. From: Powell S. *Esports' popularity 'only scratching the surface'.* [Online] Available from: https://www.bbc.co.uk/news/newsbeat-56732659/ [Accessed: 15th September 2021]

6. Stanton R. *Can there be such a thing as a perfect video game?*. [Online] Available from: https://www.theguardian.com/technology/2014/oct/31/can-there-be-such-a-thing-as-a-perfect-video-game/ [Accessed: 9th August 2021]

7. Super D. *What are the elements of a good game?*. [Online] Available from: https://www.quora.com/What-are-the-elements-of-a-good-video-game/ [Accessed 9th August 2021]

8. Ask Gamedev. (2021) *The Best Game Engines of 2021.* [Video] https://www.youtube.com/watch?v=bPlRsmRxRdA/ [Accessed 27th September 2021]

9. OK Beast. (2018) *Breaking Down Video Game Mechanics.* [Video] https://www.youtube.com/watch?v=WW4OoUmSk38/ [Accessed 28th September 2021]

10. Nintendo Entertainment Planning & Development. *Super Mario Odyssey* [Computer game]. Kyoto: Nintendo; 2017.

11. Sony Interactive Entertainment Europe. *God of War.* [Computer game]. London: Sony Interactive Entertainment Europe; 2018.

12. Atlassian. *Trello*. [Online] Available from: https://trello.com/ [Accessed: 15th September 2021]

13. Yaden J. *The bestselling games of all time.* [Online] Available from: https://www.digitaltrends.com/gaming/bestselling-games-of-all-time/ [Accessed 2nd October 2021]

14. Nintendo. *Tetris.* [Computer game]. Kyoto: Nintendo; 1984.

15. Mojang Studios. *Minecraft.* [Computer game]. Stockholm: Mojang Studios; 2011.

16. Rockstar Games. *Grand Theft Auto V.* [Computer game]. New York: Rockstar Games; 2013.

17. Benoit JT. *The 15 Highest-Grossing Video Games of The Decade (And How Much They Made).* [Online] Available from: https://gamerant.com/best-selling-games-2010s-decade/ [Accessed 2nd October 2021]

18. PUBG Corporation. *PlayerUnknown's Battlegrounds.* [Computer game]. Washington: Microsoft Corporation; 2017.

19. NPD Group. *Top 10 Video Games.* [Online] Available from: https://www.npd.com/news/entertainment-top-10/2021/top-10-video-games/ [Accessed 2nd October 2021]

20. Treyarch. *Call of Duty: Black Ops Cold War.* [Computer game]. California: Treyarch; 2020.

21. San Diego Studio. *MLB The Show 21.* [Computer game]. New York: MLB Advanced Media; 2021.

22. Capcom: *Resident Evil Village.* [Computer game]. Osaka: Capcom; 2021.

23. Riot Games. *Valorant.* [Computer game]. Los Angeles: Riot Games; 2020.

24. Valve Corporation. *Counter-Strike: Global Offensive.* [Computer game]. Washington: Valve Corporation; 2012.

25. id Software. *DOOM Eternal.* [Computer game]. Maryland: Bethesda Softworks; 2020.

26. Webb K. *'Fortnite' was the most important video game of this decade, and it will be for the net one too.* [Online] Available from: https://www.businessinsider.com/fortnite-most-influential-video-game-decade-2019-12?r=US&IR=T/ [Accessed 8th October 2021]

27. Epic Games. *Fortnite: Battle Royale.* [Computer game]. North Carolina: Epic Games; 2017.

28. Respawn Entertainment. Titanfall 2. [Computer game]. California: Electronic Arts Inc; 2016.

29. Treyarch. Call of Duty: Black Ops III. [Computer game]. California: Treyarch; 2015.

30. Respawn Entertainment. Apex Legends. [Computer game]. California: Electronic Arts Int.; 2019.

31. Ubisoft Montreal. *Tom Clancy's Rainbow Six Siege.* [Computer game]. Montreuil. Ubisoft; 2015.

32. Ubisoft Montreal. *Hyper Scape.* [Computer game]. Montreuil. Ubisoft; 2020.

33. Metzger A. *Call of Duty: Black Ops Cold War – Nuketown '84 Review'*. [Online] Available from: https://lastwordongaming.com/2020/12/02/call-of-duty-black-ops-cold-war-nuketown-84-review/ [Accessed 8th October 2021]

34. TiMi Studio Group. *Call of Duty: Mobile.* [Computer game]. California: Activision; 2019.

35. Smed, J. Kaukoranta, T. & Hakonen, H. (2002). *Aspects of networking in multiplayer computer games*. Electronic Library, 20(2), 87–97. https://doi.org/10.1108/02640470210424392.

36. Tom Weiland. (2019) *Connecting Unity Clients to a Dedicated Server | C# Networking Tutorial - Part 1*. [Video] https://www.youtube.com/watch?v=uh8XaC0Y5MA&list=PLXkn83W0QkfnqsK8I0RAz5AbUxfg3bOQ5&index=1/ [Accessed 9th October 2021]

37. Tom Weiland. (2019) *Sending and Receiving Data Using TCP | C# Networking Tutorial - Part 2*. [Video] https://www.youtube.com/watch?v=4uHTSknGJaY&list=PLXkn83W0QkfnqsK8I0RAz5AbUxfg3bOQ5&index=2/ [Accessed 9th October 2021]

38. Tom Weiland. (2019) *Implementing UDP Communication Between Clients and the Server | C# Networking Tutorial - Part 3*. [Video] https://www.youtube.com/watch?v=QajkUJeypy4&list=PLXkn83W0QkfnqsK8I0RAz5AbUxfg3bOQ5&index=3/ [Accessed 9th October 2021]

39. Tom Weiland. (2019) *Spawning Players and Adding Basic Player Movement | C# Networking Tutorial - Part 4*. [Video] https://www.youtube.com/watch?v=_h6Ta-vxAzQ&list=PLXkn83W0QkfnqsK8I0RAz5AbUxfg3bOQ5&index=4/ [Accessed 9th October 2021]

40. Tom Weiland. (2019) Properly Handling Disconnections | C# Networking Tutorial - Part 5. [Video] https://www.youtube.com/watch?v=Q3G_BBpbCek&list=PLXkn83W0QkfnqsK8I0RAz5AbUxfg3bOQ5&index=5/ [Accessed 9th October 2021]

41. Tom Weiland. (2019) *Moving the Server Code into Unity | C# Networking Tutorial - Part 6*. [Video] https://www.youtube.com/watch?v=qkjr_rv4AIQ&list=PLXkn83W0QkfnqsK8I0RAz5AbUxfg3bOQ5&index=6/ [Accessed 9th October 2021]

42. Tom Weiland. (2019) *Server Side Collisions and Better Player Movement | C# Networking Tutorial - Part 7*. [Video] https://www.youtube.com/watch?v=fnc2WKGV2eA&list=PLXkn83W0QkfnqsK8I0RAz5AbUxfg3bOQ5&index=7/ [Accessed 9th October 2021]

43. Tom Weiland. (2019) *Player Shooting and Getting the Right IP to Connect to | C# Networking Tutorial - Part 8*. [Video] https://www.youtube.com/watch?v=yxQ0_TL1jw8&list=PLXkn83W0QkfnqsK8I0RAz5AbUxfg3bOQ5&index=8/ [Accessed 9th October 2021]

44. Tom Weiland. (2019) *Spawning Items on the Server and Picking Them Up | C# Networking Tutorial - Part 9*. [Video] https://www.youtube.com/watch?v=7Tvdla3lqwo&list=PLXkn83W0QkfnqsK8I0RAz5AbUxfg3bOQ5&index=9/ [Accessed 9th October 2021]

45. Tom Weiland. (2019) *Synchronizing Projectiles and Explosions Across the Network | C# Networking Tutorial - Part 10*. [Video] https://www.youtube.com/watch?v=lVX7qgiSYgY&list=PLXkn83W0QkfnqsK8I0RAz5AbUxfg3bOQ5&index=10/ [Accessed 9th October 2021]

46. Tom Weiland. (2019) *Implementing Server-Side AI Enemies in Unity | C# Networking Tutorial - Part 11*. [Video] https://www.youtube.com/watch?v=IhctHRcI3gs&list=PLXkn83W0QkfnqsK8I0RAz5AbUxfg3bOQ5&index=11/ [Accessed 9th October 2021]

47. Amazon. *Amazon Web Services.* [Online] Available from: https://aws.amazon.com/?nc2=h_lg/ [Accessed: 19th October 2021]

48. Will-S-Dev. (2020) *Unity-Parkour-Example.* [Source Code]. https://github.com/Will-S-Dev/Unity-Parkour-Example/ [Accessed: 7th January 2022]

49. Valve. *Steam.* [Online] Available from: http://store.steampowered.com/ [Accessed: 15th January 2022]