DEDICATED SERVER IMPLEMENTATION IN MULTIPLAYER FIRST PERSON SHOOTER(FPS) BATTLE

LOH YONG XUAN

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

DEDICATED SERVER IMPLEMENTATION IN MULTIPLAYER FIRST PERSON SHOOTER(FPS) BATTLE

LOH YONG XUAN

This report is submitted in partial fulfillment of the requirements for the Bachelor of Game Technology with Honours.

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TEKNIKAL MALAYSIA MELAKA

2024

DECLARATION

I hereby declare that this project report entitled

DEDICATED SERVER IMPLEMENTATION IN MULTIPLAYER FIRST PERSON SHOOTER(FPS) BATTLE

is written by me and is my own effort and that no part has been plagiarized without citations.

STUDENT : Loh Date : 4/9/24
(LOH YONG XUAN)

I hereby declare that I have read this project report and found this project report is sufficient in term of the scope and quality for the award of Bachelor of Game Technology with Honours.

SUPERVISOR : [illegible] Date : 6/9/2024
(MISS SHAFINA BINTI ABD KARIM ISHIGAKI)

DEDICATION

To my beloved parents,

Your unwavering support, endless encouragement, and boundless love have been the foundation of my journey. This work is a testament to the values and lessons you've instilled in me. Thank you for always believing in me, even when I doubted myself. I dedicate this achievement to you with all my heart.

ACKNOWLEDGEMENTS

I am sincerely grateful to Miss Shafina Binti Abd Karim Ishigaki, my supervisor, for her invaluable guidance and constant support throughout the completion of this project. Her thoughtful advice and constructive feedback over the last three months have played a crucial role in shaping the direction and quality of my work. Miss Shafina's expertise and commitment to excellence have deepened my understanding of the subject and motivated me to strive for higher standards. Her patience and willingness to assist at every stage of this process have been greatly appreciated. Thank you, Miss Shafina, for being an exceptional mentor and for your unwavering dedication to my academic and professional development.

I would also like to extend my deepest thanks to my beloved parents, whose constant support and encouragement have been my greatest source of strength throughout this project. Their unwavering belief in me, along with their sacrifices, has allowed me to focus on my studies and successfully complete this project. Their love and confidence in my abilities have fueled my academic journey, and I am forever grateful. Thank you, Mom and Dad, for always being there and providing endless support.

ABSTRACT

This project explores the development of an engaging multiplayer First-Person Shooter (FPS) game by addressing prevalent networking issues, particularly network latency and server instability, which significantly affect player experience. Network latency, or lag, is the delay between a player's input and the corresponding game response, often leading to frustration and reduced immersion. FPS games require precise hand-eye coordination and are highly sensitive to network performance, making latency a critical challenge. The study aims to develop a multiplayer FPS game using the Photon Fusion network to ensure synchronized and smooth gameplay. The project objectives include studying networking issues such as latency and delay in multiplayer FPS games, designing a game utilizing the Photon Fusion network, and integrating a stable dedicated server to maintain a seamless and stable gameplay experience. Testing revealed positive user feedback, with most usability and performance ratings exceeding 4 out of 5. Expert evaluations highlighted the prototype's robust environment design and smooth performance while recommending improvements such as weapon variety and user interface optimization. By achieving these results, the project demonstrates the potential of using Photon Fusion to enhance player engagement and satisfaction while identifying areas for further refinement.

ABSTRAK

Projek ini meneroka pembangunan permainan First-Person Shooter (FPS) berbilang pemain yang menarik dengan menangani isu rangkaian yang lazim, terutamanya latensi rangkaian dan ketidakstabilan pelayan, yang memberi kesan ketara kepada pengalaman pemain. Latensi rangkaian, atau lag, adalah kelewatan antara input pemain dan respons permainan yang sepadan, yang sering menyebabkan kekecewaan dan mengurangkan imersi. Permainan FPS memerlukan koordinasi tangan-mata yang tepat dan sangat sensitif terhadap prestasi rangkaian, menjadikan latensi sebagai cabaran kritikal. Kajian ini bertujuan untuk membangunkan permainan FPS berbilang pemain menggunakan rangkaian Photon Fusion untuk memastikan permainan yang terselaras dan lancar. Objektif projek ini termasuk mengkaji isu rangkaian seperti latensi dan kelewatan dalam permainan FPS berbilang pemain, mereka bentuk permainan menggunakan rangkaian Photon Fusion, dan mengintegrasikan pelayan berdedikasi yang stabil untuk mengekalkan pengalaman permainan yang lancar dan stabil. Ujian menunjukkan maklum balas positif daripada pengguna, dengan kebanyakan penilaian kebolehgunaan dan prestasi melebihi 4 daripada 5. Penilaian pakar menonjolkan reka bentuk persekitaran prototaip yang kukuh dan prestasi yang lancar sambil mengesyorkan penambahbaikan seperti variasi senjata dan pengoptimuman antara muka pengguna. Dengan mencapai keputusan ini, projek ini menunjukkan potensi penggunaan Photon Fusion untuk meningkatkan penglibatan dan kepuasan pemain sambil mengenal pasti bidang yang memerlukan penambahbaikan lanjut.

TABLE OF CONTENTS

LIST OF TABLES

PAGE

LIST OF FIGURES

LIST OF ABBREVIATIONS

FYP	-	Final Year Project
FPS	-	First Person Shooter
MUD	-	MultiUser Dungeon
LAN	-	Local Area Network
LoL	-	League of Legends
VR	-	Virtual Reality
MMOs	-	Massive Multiplayer Online games
PCs	-	Personal Computers
RPG	-	Role-Playing Game
MMORPGs	-	Multiplayer role-playing games
NPCs	-	Non-player characters
AI	-	Artificial Intelligence
WoW	-	World of Warcraft
MMOFPS	-	Multiplayer first-person shooter
MMORTS	-	Massively multiplayer online real-time strategy game
RTS	-	Real-time strategy
QoE	-	Quality of Experience
NICs	-	Network Interface Cards
GUI	-	Graphical user interface
HUD	-	Heads-Up Displays
K/D	-	Kills per death

LIST OF ATTACHMENTS

CHAPTER 1: INTRODUCTION

1.1 Introduction

The First-Person Shooter (FPS)game genre has been a very famous genre for the past few decades as it provides an immense and intensive gaming experience to the players. FPS is a game where players point a fake gun at simulated targets and the eyesight in a way that is very similar to the experience of holding a gun and shooting at the target, it is defined by the same basic skills of lining up a reticule with a target and timing the trigger and reload actions (Therrien 2015).

As the Internet has become more advanced, it has become a large part of everyday life. Hence, networked games have become more widely spread, 67% of teens now regularly play some game online (Rideout, Foehr et al. 2010). Multiplayer online games become popular when mass adoption of broadband internet spread throughout the world. Multiplayer games are games that allow a group of players to play together at the same time and in the same virtual world (Molyneux, Vasudevan et al. 2015). Top teams and players can compete, leading to a more challenging and more immersive experience.

A dedicated server in FPS games uses a client-server architecture, which simplifies network traffic compared to peer-to-peer models. The step to find available game servers includes two procedures: Each listed server is probed after the client's first query a master server for a list of registered game servers. A client estimates the round-trip time (RTT) to each server based on the time between emitting a probe and receiving a reply (Armitage 2008). The dedicated server will also offer full control to the user.

1.2 Problem Background

In the dynamic landscape of multiplayer FPS games, developers encounter a multitude of challenges ranging from game mechanics to server infrastructures. In the realm of FPS gaming, player immersion and responsiveness are paramount, demanding high-speed interactions and fluid gameplay experiences. However, as the complexity and graphical fidelity of FPS games continue to evolve, such arrangements increase the complexity of the hardware requirement (Li 2008), causing players to need to have high-end hardware to enjoy full experience in FPS games.

In multiplayer FPS games, the interaction between players adds layers of complexity to game design and network architecture. The real-time nature of FPS gameplay requires precise synchronization of player actions across the network, necessitating robust server infrastructure and efficient net code. However, the inherent latency in online connections can disrupt the fluidity of gameplay, leading to issues such as lag and desynchronization, requiring players to need a stable network connection to have a seamless experience while playing FPS games, typically requires latency below 150-200 ms (Armitage, Claypool et al. 2006).

The backbone of multiplayer FPS games lies in the server infrastructure that facilitates player interactions and data synchronisation. As player counts increase and gameplay complexity grows, developers face the challenge of scaling server resources to accommodate growing player populations while maintaining performance and stability. Server-side optimizations, load balancing, and network protocols play pivotal roles in ensuring smooth gameplay experiences for all players. typical response times to ensure fluent play must be between 100 milliseconds in online FPS action games (Beigbeder, Coughlan et al. 2004). However, the cost and complexity of managing robust server infrastructure present ongoing challenges for developers seeking to deliver high-quality multiplayer FPS games in a competitive gaming landscape. The cost of maintaining a high-quality server requires a high cost. For example, Massively Multiplayer Online Game (MMOG) operators need to install and operate a large dedicated multiserver infrastructure, with hundreds to thousands of computers hosting the distributed load of each game (Bartle 2004).

1.3 Problem Statement

In FPS games, there are several common network issues and one of them is network latency, this has been a problem for quite some time. Lag refers to the delay between the generation of a player’s input and the presentation of its outcome to the related players in the game (Lee and Chang 2015) .The server needs to be optimized so that it can ensure that the player actions can be processed smoothly otherwise it will cause frustration among the players and affect the player's immersion and engagement towards the game.

FPS requires more than one player to be fun because they can relax and chat with others in the game, where they can join clans and involve themselves in more social interaction (Jansz and Tanis 2007). The weaker players may also enjoy the game when they don’t have additional delay while the stronger player has, putting the weaker player at a greater advantage (Dick, Wellnitz et al. 2005). For the players to connect, a network connection is required. FPS games have the most stringent requirement on network performance such as delay because of the highly interactive of such games (Liu, Wang et al. 2006).

Server instability and performance issues are inevitable when it comes to multiplayer FPS games (Süselbeck, Schiele et al. 2009). In FPS games, players require quick-hand-eye coordination in moving the crosshairs of a gun to target the enemy while others only require team players to move strategically. This diversity of player actions is impacted differently by frame rate as some acts are less sensitive to frame rate such as selecting a troop while a sniper rifle aiming at a moving opponent is impacted greatly by frame rate (Claypool and Claypool 2007).

1.4 Project Aim

This project aims to develop an engaging multiplayer FPS game by using photon fusion network to ensure smooth and synchronized gameplay.

1.5 Project Objectives

To achieve the project's aim, three specific objectives must be met.

1) To study the networking issues in FPS multiplayer games such as network latency and delay.

2) To design and develop a multiplayer FPS game using Photon Fusion network.

3) To evaluate a stable dedicated server into multiplayer FPS game to ensure smooth and stable gameplay experience for all players.

1.6 Project Scope

The project scope centers on developing a multiplayer FPS game where any player can become the host by using photon fusion network. This project allows multiplayer players to choose the same virtual space and interact in real-time, where the players are categorized as host and client. Players can dynamically assume the role of either a host or a client, ensuring seamless gameplay experiences regardless of their roles. The host is responsible for managing the game state and synchronizing it with all connected clients. Hence, the host acts as a dedicated server that hosts the game session and handles authoritative decisions and game logic to ensure consistency across all clients. Clients in the other hand are player machines that connect to host to participate in game session, they send their input such as shooting, movement to the host and receive updates on the game state from the host. Hence, this means when the host player leaves, the game is unable to be continued as the dedicated server is no longer there. However, any players can be the new host to make sure that the game state continue to synchronize in this project.

1.7 Project Importance

By implementing photon fusion network into our multiplayer FPS game, Any player is allowed to become the host to reduce disruptions and players can still interact in real-time within the same virtual world, ensuring a more immersive multiplayer experience even when the original host leaves the game.

The ability for any player to become the host makes the game more scalable and flexible. This approach can accommodate various network conditions and has a higher tolerance towards network issues such as delay. By leveraging photon fusion network in this project, reliable communication between hosts and clients can be ensured and low latency can be handled for fast-paced FPS games.

1.8 Report Organization

This thesis is structured into six chapters, outlined as follows:

Chapter 1 serves as the introductory part of the project. It elaborates on the purpose of the project, including the background, problem statement, objectives, scope, and significance of the project.

Chapter 2 provides a detailed review of relevant literature and methodologies for developing a multiplayer FPS game. It begins with an introduction to set the context, followed by an exploration of the FPS game genre an analysis of existing games within this category and a detailed comparison to highlight their strengths and weaknesses. Following this, the project methodology is outlined, detailing the approach and processes used in the development of the game. The chapter concludes with a summary of key findings and methodological insights that guide the project's implementation.

Chapter 3 focuses on the project's design. It begins with an introduction and explanation of the game architecture. The majority of the chapter is divided into two main sections: gameplay and game art. The gameplay section discusses the game's mechanics and dynamics, while the game art section covers the visual and auditory elements necessary for creating an immersive gaming experience.

Chapter 4 addresses the implementation of the project. It covers the creation of game art and the integration of game components as described in Chapter 3. The chapter also includes game configuration management and provides an update on the implementation status. It concludes by summarizing the key points from the implementation phase.

Chapter 5 is the project's outcome and testing. It outlines the test plan and describes how tests were carried out. The chapter then presents and analyzes the test results. Finally, it concludes by summarizing the testing phase's outcomes.

Chapter 6 concludes the overall research on this project. It reflects on the strengths and weaknesses observed throughout the project, proposes improvements for future iterations, and outlines the project's contributions to the field. Finally, the chapter wraps up by summarizing the project's findings and concluding remarks.

CHAPTER 2: LITERATURE REVIEW AND PROJECT METHODOLOGY

2.1 Introduction

This chapter will focus on conducting a literature review. The three primary areas of focus for this project are multiplayer, first-person shooter (FPS) games, and server technology. Each of these main topics will have subtopics that will be discussed, along with an analysis of existing similar games. Additionally, this chapter will include a discussion of previous research or related findings

2.2 Multiplayer

2.2.1 Evolution of multiplayer games

At the start, there were no online multiplayer games that we have now. Back then, the first ever documented multiplayer game is Tennis for Two (as shown in **Figure 2.1**), which was created by William Higinbotham (as shown in **Figure 2.2**) in 1958 at Brookhaven National Laboratory. William Higinbotham used an analog computer designed to calculate ballistic trajectories to build one the game, Tennis for Two.(De La Cruz and Ryan 2015)

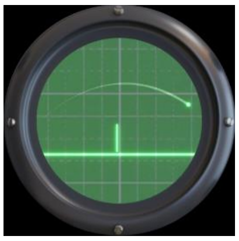


Figure 2.1 Tennis for Two was a game in which the goal was to hit a ball back and forth on an oscilloscope screen with a net drawn in the middle.

Figure 2.2 William Higinbotham, creator of Tennis for Two

However, Tennis for Two was created using hard-wired circuitry rather than a computer for gameplay. Therefore, the true first multiplayer computer game is considered to be Spacewar! (as shown in **Figure 2.3**) The game was created by Steve Russell and a group of other hackers at MIT in 1962 (as shown in **Figure 2.4**) as a demonstration program for the PDP-1 minicomputer by Digital Equipment Corporation (DEC). (Monnens and Goldberg 2015)

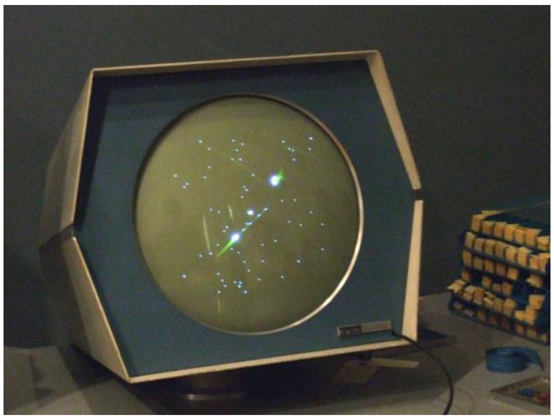


Figure 2.3 Spacewar! A game without sound and particle effects but with addictive gameplay

Figure 2.4 Steve Russell and his gang creating the game Spacewar! At MIT

However, Spacewar! was not widely spread across the community as the hardware required to play the game was very expensive, the PDP-1 cost over $100,000. A version of it however was successful, which is the game Galaxy War, appearing on campuses in Stanford in the early 1970s (as shown in **Figure 2.5**)

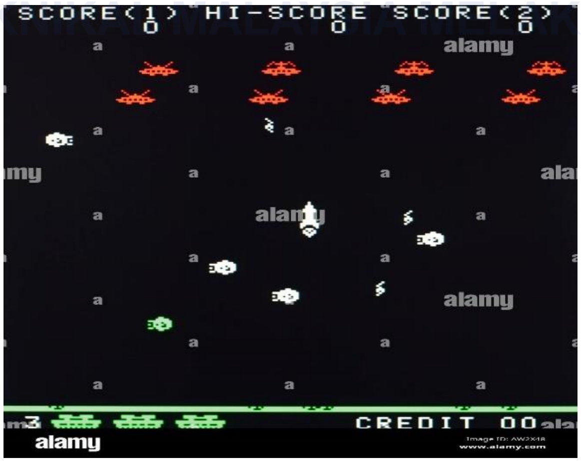


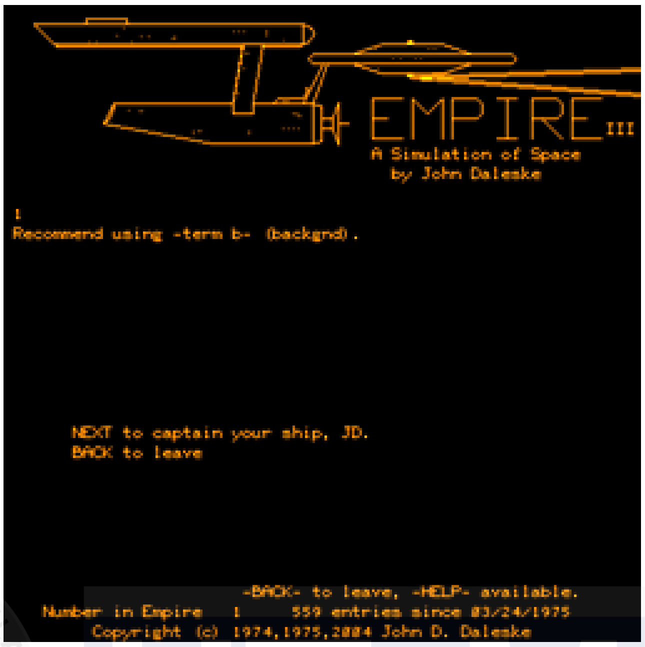
Figure 2.5 Galaxy War is an arcade game. Image reproduced with permission from Id Software, Inc.

Nolan Bushnell , the founder of Atari—a company dedicated to producing video arcade games—introduced Pong in 1972, an arcade-friendly version of Tennis for Two (as shown in **Figure 2.6**). Later, Atari released Gauntlet, a dungeon-crawling game designed for up to four players simultaneously. (as shown in **Figure 2.7**)

Figure 2.6 Pong, one of the best known of the early multiplayer games

Figure 2.7 Gauntlet, featured two to four players in cooperative

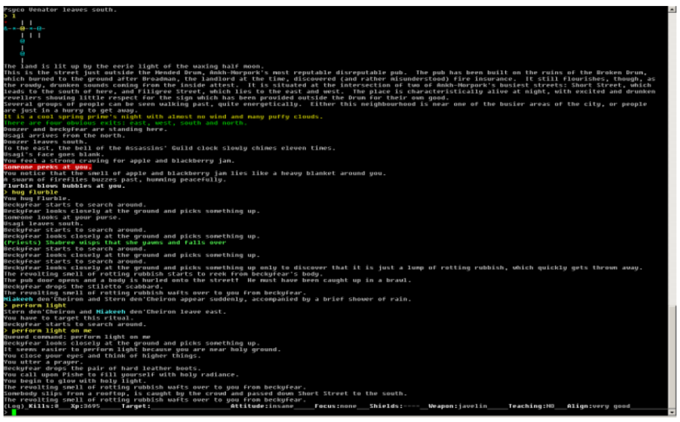
The first potential online network community was PLATO. PLATO included various communication tools such as email and online games. One of the popular games on PLATO was Empire, a multiplayer space simulation game. (as shown in **Figure 2.8**) These early games networked only in the sense that a terminal was connected to a mainframe, something like a remote login shell of the day. (Woolley 1994)



```
                              EMPIRE III
                              A Simulation of Space
                                by John Daleske

1
Recommend using -term b- (backgnd).




      NEXT to captain your ship, JD.
      BACK to leave



                      -BACK- to leave, -HELP- available.
   Number in Empire   1      559 entries since 03/24/1975
        Copyright (c) 1974,1975,2004 John D. Daleske
```

Figure 2.8 Empire, a multiplayer space simulation game

MultiUser Dungeons (MUDs) gained popularity after PLATO, as MUDs offered a virtual environment where users could interact with the world and each other, incorporating some gameplay elements. Early MUDs used text-based interfaces that required players to type in basic commands, such as "go east" or "open doors" (as shown in **Figure 2.9**). MUDs used a client-server architecture where the MUD administrator would run the server and MUD players will join the game by connecting to the MUD server with a simple Telnet program initially from a terminal (as shown in **Figure 2.10**). (Ito 1994)



```
Psyco Venator leaves south.
> l
*    | |
&-*-@-*-@-
     | | |
     @
     |
     @
     |
The land is lit up by the eerie light of the waxing half moon.
This is the street just outside the Mended Drum, Ankh-Morpork's most reputable disreputable pub.  The pub has been built on the ruins of the Broken Drum,
which burned to the ground after Broadman, the landlord at the time, discovered (and rather misunderstood) fire insurance.  It still flourishes, though, as
the rowdy, drunken sounds coming from the inside attest.  It is situated at the intersection of two of Ankh-Morpork's busiest streets: Short Street, which
leads to the south of here, and Filigree Street, which lies to the east and west.  The place is characteristically alive at night, with excited and drunken
revellers showing little respect for the sign which has been provided outside the Drum for their own good.
Several groups of people can be seen walking past, quite energetically.  Either this neighbourhood is near one of the busier areas of the city, or people
are just in a hurry to get away.
It is a cool spring prime's night with almost no wind and many puffy clouds.
There are four obvious exits: east, west, south and north.
Doozer and beckyfear are standing here.
Usagi arrives from the north.
Doozer leaves south.
To the east, the bell of the Assassins' Guild clock slowly chimes eleven times.
Usagi's face goes blank.
You feel a strong craving for apple and blackberry jam.
Someone peeks at you.
You notice that the smell of apple and blackberry jam lies like a heavy blanket around you.
A swarm of fireflies buzzes past, humming peacefully.
Flurble blows bubbles at you.
> hug flurble
You hug Flurble.
Beckyfear starts to search around.
Beckyfear looks closely at the ground and picks something up.
Someone looks at your purse.
Usagi leaves south.
Beckyfear starts to search around.
Beckyfear looks closely at the ground and picks something up.
(Priests) Shabree wisps that she yawns and falls over
Beckyfear starts to search around.
Beckyfear looks closely at the ground and picks something up.
Beckyfear starts to search around.
Beckyfear looks closely at the ground and picks something up only to discover that it is just a lump of rotting rubbish, which quickly gets thrown away.
The revolting smell of rotting rubbish starts to reek from beckyfear's body.
The pub door opens and a body is hurled onto the street!  He must have been caught up in a brawl.
Beckyfear drops the stiletto scabbard.
The revolting smell of rotting rubbish wafts over to you from beckyfear.
Miakeeh den'Cheiron and Stern den'Cheiron appear suddenly, accompanied by a brief shower of rain.
> perform light
Stern den'Cheiron and Miakeeh den'Cheiron leave east.
You have to target this ritual.
Beckyfear starts to search around.
> perform light on me
Queued command: perform light on me
Beckyfear looks closely at the ground and picks something up.
It seems easier to perform light because you are near holy ground.
You close your eyes and think of higher things.
You utter a prayer.
Beckyfear drops the pair of hard leather boots.
You call upon Pishe to fill yourself with holy radiance.
You begin to glow with holy light.
The revolting smell of rotting rubbish wafts over to you from beckyfear.
Somebody slips from a rooftop, is caught by the crowd and passed down Short Street to the south.
The revolting smell of rotting rubbish wafts over to you from beckyfear.
(Log)_Kills:0___Xp:3695_____Target:__________________Attitude:insane_____Focus:none___Shields:----__Weapon:javelin_____Teaching:NO___Align:very good_____
>
```

Figure 2.9 Screen shot of one of the games using MUDs

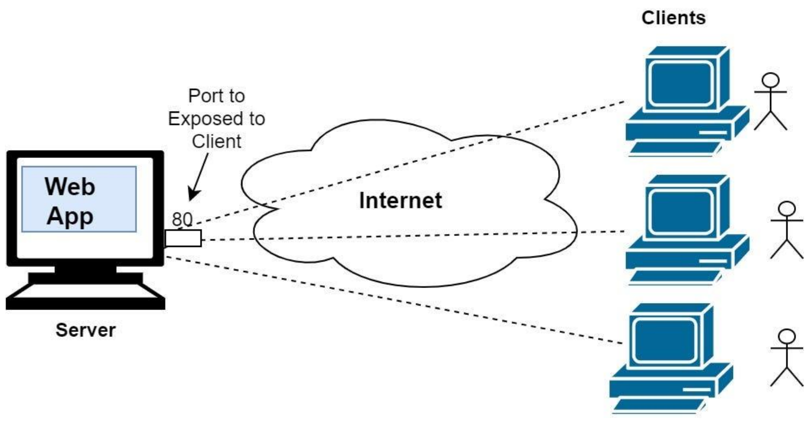


Figure 2.10 Basic client-server topology used in early MUD games

In the 1990s, online multiplayer gaming started to become more popular. Game such as Doom (as shown in **Figure 2.11**) were introduced in 1993 as the internet becomes more accessible, allowing players to compete over LAN (Local Area Network) connections. Doom became immensely popular, with total sales reaching 1.5 million copies. Communication between Doom peers relied on Ethernet broadcasts for all its traffic. A unique feature of Doom is its software renderer, which is particularly important for RL algorithms that run on CPUs rather than GPUs. One of the advantages of this is that Doom can effortlessly be run without a desktop environment and accessing the screen buffer does not require transferring it from the graphic card. (Wydmuch, Kempka et al. 2018)

Figure 2.11 Sample screen from Doom showing the first-person perpective

Starting from the late 1990s, console gaming was also starting to embrace online multiplayer. Sega included an in-built modem in their consoles, “Dreamcast” which allow users to search the Web as well as participate in online games. Sega also established a dedicated gaming portal on the Internet under the title “SegaNet”. While Sony also published PlayStation 2 and Microsoft launched the Xbox which also supports online gaming features. Xbox supports broadband access through Digital Subscriber Line (DSL) or cable modern connections which shows Microsoft is predicting a surge in online gaming. (Finn 2002)

Online multiplayer games went uncontrollable in the 2000s. PC games such as World of Warcraft, released in 2004, gained more than 10 million players, and players can add on anything they want, which allows a high level of game customisation These add-ons could tailor the game by customizing user interfaces and providing warning notifications. Similarly, League of Legends (LoL), a MOBA genre game released in late 2009, quickly gained a massive player base, reaching 70 million registered users within just three years. As a free-to-play game, LoL fostered a vast community. It requires users to authenticate immediately after a game update and upon launching the game. The game spectator system allows other players to log into the game to watch live matches. (Chen, Shashidhar et al. 2016)

The 2010s marked the rise of mobile gaming, making multiplayer experiences more accessible than ever. Titles like Clash of Clans (2012) and Pokémon GO (2016) expanded multiplayer gaming to a wider audience, emphasising casual, on-the-go play. One significant development of this decade was the introduction of cross-platform play, allowing players on different systems to compete together. Most Online Games are deployed using client–server system architecture. They interact using what is known as client software to gain access to the entire playing world. As more people venture into gaming experience a service provider will have to scale up to meet the demands of the users, such situations have been observed in massively multiplayer online role-playing game. (Kangiwa 2014)

VR is slowly emerging as a new star in the late 2010s, and early 2020s for multiplayer gaming. VR offers an immersive experience compared to PC in the same game (Carroll, Osborne et al. 2019) Such as Fortnite and Among Us.

2.2.2 Types of multiplayer games

Massive Multiplayer Online games (MMOs) are played with large numbers of players, ranging from hundreds to thousands. However, this does not mean that multiplayer games with less than hundreds of players are not considered MMOs. The number was chosen because of the Battle Royale genre, which usually has a hundred players. (Fernandes, Castanho et al. 2018)

MMOs are typically designed for Personal Computers (PCs), but they are also available on video game consoles and even mobile devices, such as Black Desert Mobile and Albion Online. Depending on gameplay style, MMOs encompass a variety of subgenres, including First-Person Shooter (FPS), Simulation, Real-Time Strategy, Role-Playing Game (RPG), and many other foundational game genres.

In MMORPGs (Multiplayer role-playing games), the virtual world is generally expansive, with thousands of real players sharing the space. Additionally, the virtual world is populated by numerous NPCs (non-player characters) controlled by AI (Artificial Intelligence). One of the most popular MMORPGs is World of Warcraft (WoW). In WoW, players can customize their avatars and align with one of two factions at war with each other. (WoW) is a game with its own unique time, place, and space, governed by both explicit rules defined by the game design (such as killing monsters to earn experience points and level up) and unwritten rules (like healing instead of fighting if you are a healer). It offers players a variety of goals to pursue. Some of the defined goals do not lead to quantifiable goals such as leveling up. (Rettberg, Corneliussen et al. 2008)

In an FPS game, a virtual world—often limited in size—is shared by numerous players, typically organized into teams. Each player controls an avatar to complete a mission or eliminate all enemies. The game is viewed from a first-person perspective, allowing the player to see their avatar's hands, weapon, and the immediate environment in front of them. A key characteristic of this genre is its high level of interactivity, featuring fast-paced movements and shooting where precise aiming is essential. An example of a multiplayer first-person shooter (MMOFPS) is Destiny. First released in 2014 for PlayStation 3, PlayStation 4, Xbox 360, and Xbox One, Destiny combines RPG and MMOG elements in a shared world setting. Developed by Bungie, the game is set in a science fiction universe where players defend the last city on Earth from an encroaching darkness. Before the 2017 sequel, four expansions were released to expand the game's content.

Massively multiplayer online real-time strategy games, or "MMORTS," combine real-time strategy (RTS) elements with a persistent world. In these games, players usually assume the role of a general, king, or similar leader, commanding an army in battle while managing the resources needed for warfare. These games are typically set in sci-fi or fantasy universes and are distinct from single-player or small-scale multiplayer RTS games due to the large player base and the use of a persistent world—usually hosted by the game's publisher—that continues to evolve even when players are offline. Rokkatan is an example of an MMORTS that takes place on a game map, which is described as an editable text file where teams compete. Players must coordinate to achieve various objectives, such as occupying and defending flags that have already been captured.

All game genres can incorporate multiplayer features to become a massively multiplayer online game (MMO) if they meet the following requirements:

- The game must include some form of interaction between players.
- It must support a large number of players within the same game instance.

2.2.3 Issues in multiplayer games

In a multiplayer game, it can be said that the player is interacting with the virtual world in "real time" because the virtual world responds to the player's actions within a few milliseconds—faster than what the human perception can detect. However, we also need to account for additional latency introduced by the network in receiving responses.

For example, let's consider a simple chicken-hunting game using a client-server architecture (as shown in Figure 2.12) with four clients as an example. Imagine that three clients experience network delays of 50, 100, 150, and 200 milliseconds one way from their computers to the server, respectively. Now, assume the second client shoots the target at a specific moment. The action takes 200 milliseconds to travel to the server and back. At the 100-millisecond mark, only the server will be aware that the duck has been shot; at 150 milliseconds, client 1 will know that the chicken has been shot, and by then, the server has already recorded the dog retrieving the chicken. At 200 milliseconds, client 2 will see he has hit the duck, while clients 3 and 4 still see the duck alive. (Oluwatosin 2014)

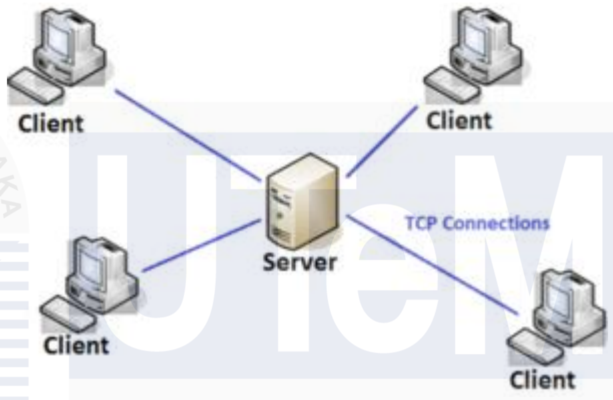


Figure 2.12 Client server architecture

Hence, there are three main issues while playing multiplayer games which are latency, packet loss and bandwidth.

refers to the time required to transmit the player’s input from the client application layer to the server's application layer. This is known as the one-way delay. If the return path (from server to client) is considered, we get the round-trip time, commonly referred to as “ping” by gamers. In an FPS game, if player A, with a latency of 200ms, is aiming at player B, who is moving at 1000 units per second perpendicular to player A, and uses an instant-hit weapon, then without considering latency in the game logic, player A would need to aim 100 units ahead of player B to score a hit. This is because by the time the shooting message arrives at player B's computer, the player B's spatial location would have been updated (as shown in **Figure 2.13**). (Jiang, Safaei et al. 2005)

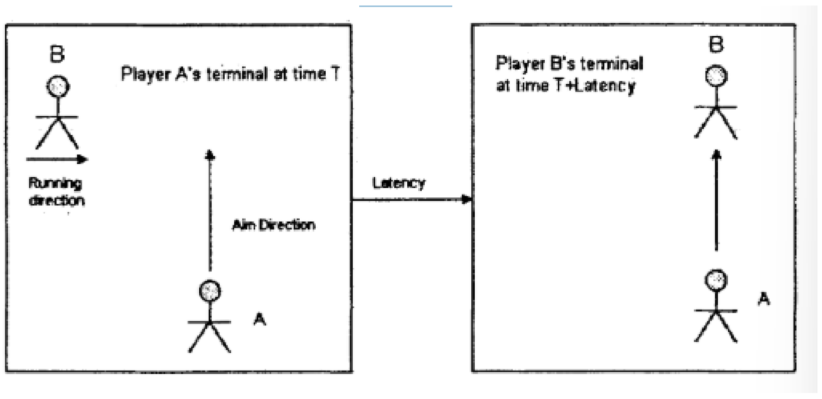

B
Player A's terminal at time T
Running direction
Aim Direction
A
Latency
Player B's terminal at time T+Latency
B
A

Figure 2.13 Latency

If the latency is high, the virtual world does not respond in real time, and the player perceives their actions as "delayed." This significantly affects the player's Quality of Experience (QoE) and diminishes their immersion in the virtual world. The situation becomes even more complex with multiple players experiencing different latencies. These latency differences can lead to inconsistencies and anomalies in the virtual world, the most well-known being the "shooting behind the wall" problem.

The second major issue in multiplayer games is packet loss. A packet is a small unit of data routed by a network protocol between an origin and a destination on the internet or any other packet-switched network. Network packets usually contain data such as source and destination addresses, protocols, or identification numbers. Packet loss happens when one or more transmitted data packets fail to reach their destination. There are several causes of packet loss. Some common causes include defective Network Interface Cards (NICs) on hardware (PC/Console), faulty network drivers, interference (commonly in Wi-Fi connections), faulty network cables or ports, home routers with firmware issues, network congestion (where something or someone is maxing out/downstream bandwidth).

For example, opening or closing doors or viewing the contents of chests. In these cases the client will typically perform the action preemptively assuming that the current state of the player and the interactive item is correct. However, if confirmation from the server indicates the current information such as player position or item state is incorrect, the game would be forced to rewind the interactions, in some cases teleporting players back behind closed doors or causing items to disappear from inventories. (Dueck 2021)

The third significant issue in multiplayer games is bandwidth. Bandwidth is the maximum amount of data that can be transmitted over an internet connection within a given period. While bandwidth itself isn’t a network impairment, its scarcity can lead to high latency, jitter, and packet loss. Additionally, background traffic (i.e., bandwidth utilization) is another critical factor; a congested network is not suitable for playing online games.

Some multiplayer games, like cloud gaming, provide high-quality content but at the cost of requiring significant bandwidth and server capacity. Cloud gaming demands considerable bandwidth, particularly for the downlink. If the connection cannot supply enough bandwidth, video quality may need to be reduced—whether in resolution or frame rate—thereby decreasing the overall subjective quality of the gaming experience. Another option is to suffer packet loss which severely degrades the quality of the video stream (blocking and blurring effects) and therefore of the game as well.(Saldana and Suznjevic 2015)

2.3 First-Person Shooter

2.3.1 History of FPS games

When we talked about First-Person Shooter (FPS) games, we need to know that the first real attempt at creating this genre of game was in 1973, Maze War, installed at the NASA Ames Research Center. However, some may argue that Maze War (as shown in **Figure 2.14**) was not a proper shooter because the player did not actually aim, since pressing the spacebar would send a bullet and eliminate any opponent occupying the same space, thus making the game closer to a maze chase. (Arsenault 2009)

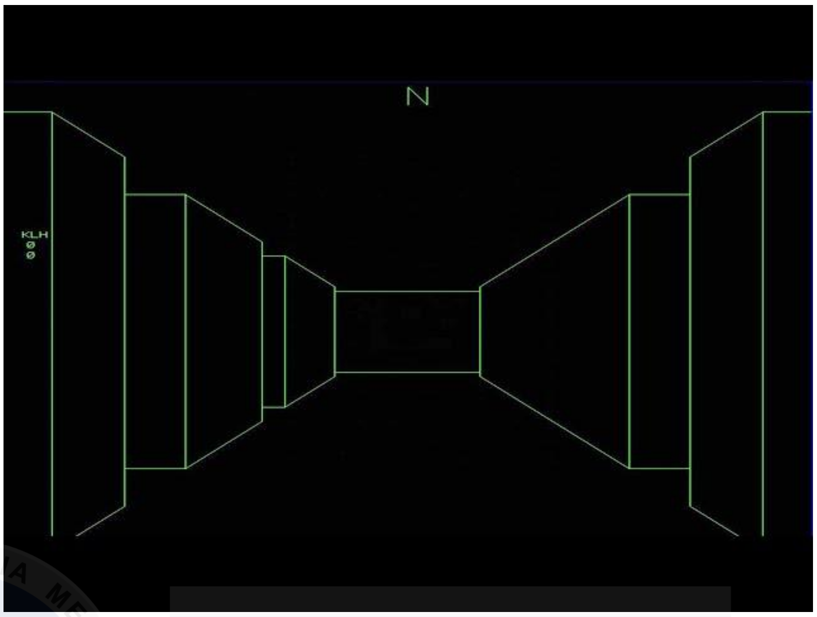


Figure 2.14 Maze War

In the 80s, arcades were popular and was the place where you see most gamers would be at. It was at that time one of the ancestors of today's FPS games emerged, Atari's Battlezone (as shown in **Figure 2.15**). This game lets players control a powerful assault tank, navigating a somewhat plain land that is surrounded by enemies. The tank could move and rotate in any direction, allowing players to destroy adversaries (just as a tank should!). Battlezone offers a first-person perspective and utilizes wireframe 3D graphics. A version of the game was later released for home computers in 1983.

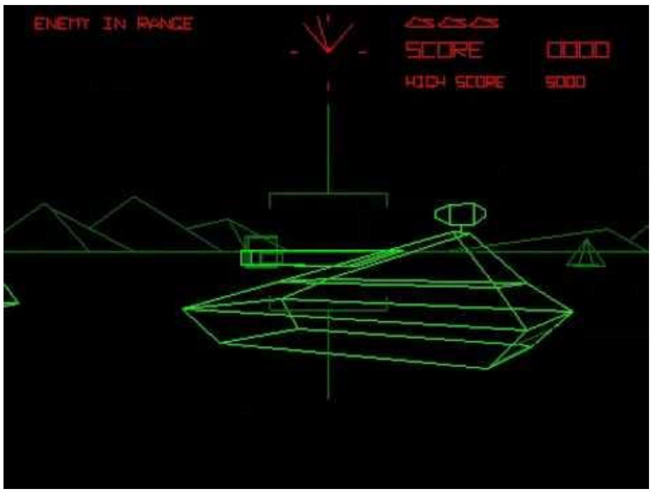


Figure 2.15 Atari's Battlezone

Released the same year as Doom, Taito's Gun Buster (as shown in **Figure 2.16**) was the first free-roaming, sprite-based FPS to appear in arcades. It featured a unique control setup, using a joystick for movement and a light gun for aiming and shooting. Additionally, the cabinets could be networked for multiplayer deathmatches.

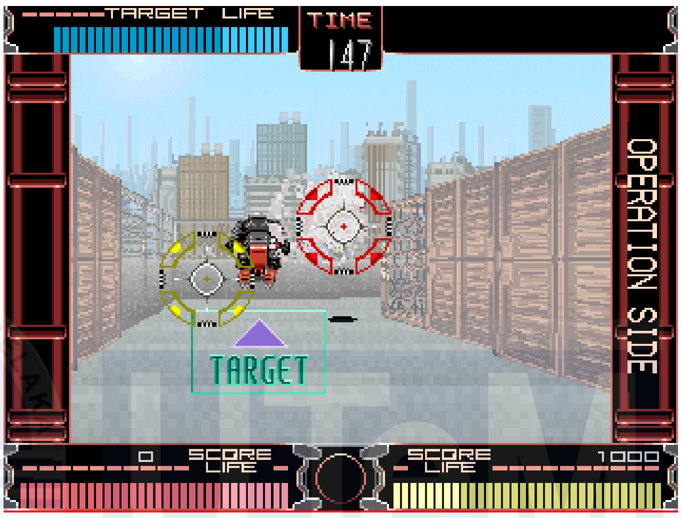


Figure 2.16 Gun Buster

However, all the games above are either for other purposes or played in arcade, none of them are for PC players. The first true FPS game in PC was Midi Maze by Hybrid Arts (as shown in **Figure 2.17**), designed for the Atari ST and released in 1987, Midi Maze featured players as Pac-Man-like orbs navigating a maze, where they could move in any direction and shoot deadly bubbles at other similar enemies. The game's key features that made it enjoyable were two main elements:

Its networking capability allowed up to sixteen players to compete in the same maze, though it often suffered from significant lag. Additionally, players could design their own mazes using a simple text editor.

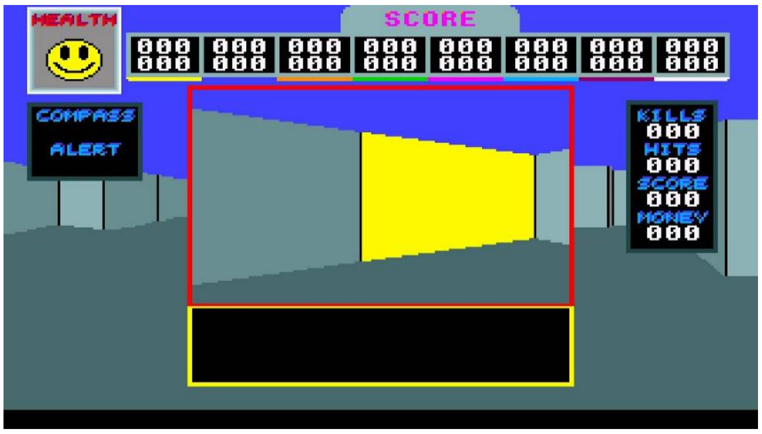


Figure 2.17 Midi Maze

The trends in FPS during this time included the following:

– The earliest FPS games used simple tile-based movement, allowing players to move from one tile to the next and turn only in 90-degree angles.

– These games usually grid-based or employed ray-casting technology. Titles like Wayout used ray-casting to render environments and wall segments according to the player's position and viewing angles.

– MIDI Maze was among the first LAN games, though it took some time for the concept to gain widespread traction, despite its multiplayer mode developing a cult following.

The true evolution of the FPS subgenre took place in the 1990s. iD Software, founded by John Carmack, John Romero, Tom Hall, and Adrian Carmack—all former employees of publisher Softdisk—played a pivotal role in this change. John Carmack, the team's resident genius, introduced innovations that revolutionized the gaming industry. He created a technique to render 3D images as quickly as 2D ones and introduced the concept of ray casting, which allowed the PC to draw only what the player could see rather than rendering the entire game world, significantly boosting rendering speeds.

The first game published was Wolfenstein 3D (as shown in Figure 2.18), the precursor to all 3D shooters. In the game, you play as William “B.J.” Blazkowicz, a tough, shotgun-wielding Jewish-American spy with a knack for taking down Nazis. He was gaming’s answer to John Rambo—the ultimate one-man army, a trope that still endures. The game kept players constantly on edge, with a pace faster than any other game of its time.

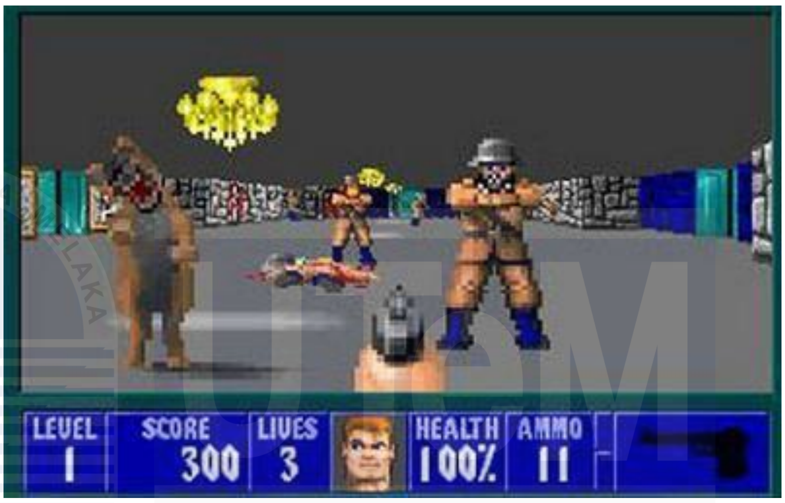


Figure 2.18 Wolfenstein 3D

After creating Hovertank 3-D (which improved rendering speed) and Catacomb 3-D (which introduced texture mapping). They could now map bitmap textures onto 3D objects, create floors at different altitudes, and illuminate areas with varying levels of light.

Thus, was born Doom (as shown in **Figure 2.19**), the game that single-handedly transformed FPS gaming generated unprecedented anticipation. So many users attempted to access the FTP server where id Software was uploading the game that they were unable to connect. The administrators had to disconnect all other users in order to upload the game. Thirty minutes later, 10,000 people tried to download it simultaneously, crashing the network after the upload was completed.

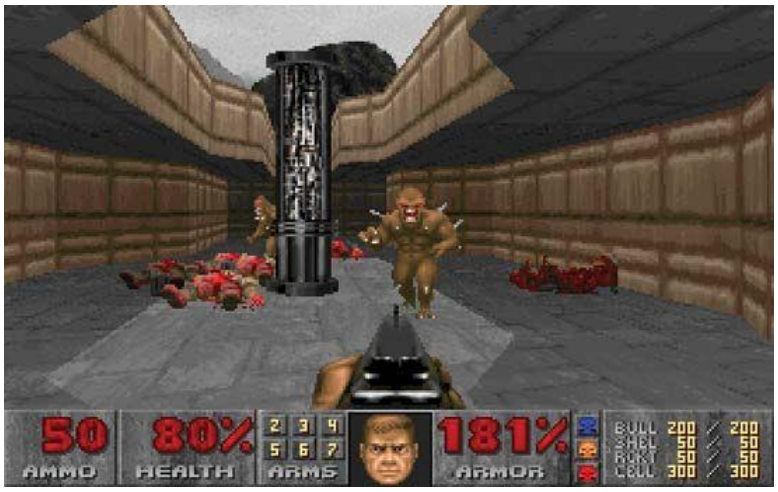


Figure 2.19 Doom

FPS trends during this era includes the following feature:

– Wolfenstein 3D is credited with pioneering the modern FPS genre and creating a shooter game design template that still influences games today.

– The fast-paced gameplay and intense music of Doom created one of the most immersive experiences of its time, leading to the release of numerous Doom clones.

– The concept of deathmatches which are competitive multiplayer matches was further refined in Doom, building on earlier efforts like MIDI Maze, and marked the first major success in large-scale multiplayer gaming.

– Doom also featured impressive graphics for its time, including variable-height floors and ceilings, as well as basic lighting effects that became standard in many subsequent games.

iD once again led the way when they published Quake in 1996 (as shown in **Figure 2.20**). The game is a shooter set in a dark medieval world, featuring music by Trent Reznor. Quake introduced fully 3D environments, offering players new possibilities with movement, including the concept of "rocket jumping," where the blowback from explosive weapons could propel the character high into the air and across the level. The game was a blast. Quake also featured different multiplayer modes, including online multiplayer after the Quake World update, making it one of the first major competitive games and a precursor to modern esports.

Figure 2.20 Quake

When Epic released Unreal Tournament in 1999, it was then competitive multiplayer gaming became popular and one of the mainstreams in the gaming field. Unreal Tournament was the first FPS game designed specifically for multiplayer. While it offered a single-player mode for training against bots, the main focus of the game was online and LAN play.

In1998 and 1999, the release of two of the most famous FPS game in the world, Rainbow Six and Counter Strike (as shown in **Figure 2.21**) was a huge hit as they were the first shooters to emphasize tactics similar to those used by real-life military or SWAT teams. In these games, planning your attack and making every move count was crucial. This emphasis on teamwork and precision is what made them some of the first mainstream tactical FPS esports titles, earning them a place among the all-time great video games.

Figure 2.21 Counter Strike Global Offensive

Before this, most FPS games focus lightly on the narrative itself as they focus more on the shooting aspect itself. However, this took a twist after the release of Half-Life (as shown in **Figure 2.22**) in 1998. The story centered around Gordon Freeman, a groundbreaking physicist working at the Black Mesa lab. As a silent protagonist, reminiscent of those in Japanese RPGs, Freeman navigates a seamless world with no cutscenes, making the experience more immersive.

Figure 2.22 Half-Life

FPS trends during this era included the following characteristics:

– Quake was the first FPS to feature fully 3D maps, enemies, and power-ups, with no restrictions on angles or surface lengths.

– Quake also emphasized online gaming and introduced various game modes that continue to be staples in FPS games today.

– LAN parties became a core part of gaming culture, with events like QuakeCon, where players competed to prove their skills, originating from Quake.

– Tactical FPS games, which required more strategic thinking and often lacked player respawns, started gaining popularity, which Counter-Strike being a prime example.

– Significant advancements in video game storytelling were made in games like Half-Life, which used immersive environments and carefully crafted linear experiences, giving rise to a new niche of narrative-driven FPS games.

In the 2000s, FPS games began incorporating more realism, including in-game dialogue and developed characters. Automatic saves at specific locations or plot points, known as checkpoints, became standard, and features like regenerating shields or health were also introduced to enhance gameplay. One of the most famous game that started with these features is Halo: Combat Evolved, released in 2001 (as shown in **Figure 2.23**).

Figure 2.23 Halo: Combat Evolved

The FPS environment was took to a whole new level when it comes to Far Cry 2 (as shown in **Figure 2.24**) , the open world looks like the real world and responded to the players' behaviors. Players could experience illness or injuries, wildfires spread naturally, and the enemy AI was impressively realistic.

Figure 2.24 Far Cry 2

In 2009, the released of Borderlands (as shown in **Figure 2.25**) had everything nailed down. The game features standout art, incredibly fun music, excellent shooter mechanics, and a variety of quirky, entertaining loot, elevating FPS games to a new level. Its sequel continues the trend with a humorous and captivating story, along with phenomenal visual design.

Figure 2.25 Borderlands

Overwatch (as shown in **Figure 2.26**) which was released in 2014, the game was a resounding success. It divides players into two teams of six, with each player choosing from a diverse roster of heroes, each with unique abilities. Teams must work together to complete map-specific objectives within a limited time frame. This can examine a team coordination as they need to select the effective combination of hero characters' skills to have a bigger chance of winning the game.

Figure 2.26 Overwatch

A new subgenre is also popular among FPS games, battle royale. The game introduces the last-person-standing concept to online multiplayer, involving up to hundreds of players who start with no equipment and must scavenge for gear while eliminating other opponents. To ensure constant encounters, the play area continuously shrinks. The last player or team standing wins. This formula became a phenomenon with the release of PUBG: Battlegrounds (as shown in **Figure 2.27**). It was a massive hit for several reasons: the quick, addictive gameplay, multiple strategies for victory—whether by eliminating everyone or taking advantage of ongoing fights—and the excitement of finding and looting gear, all contributed to PUBG's success.

Figure 2.27 PUBG: Battlegrounds

Finally, FPS games have become incredibly popular in the competitive world of esports and online gaming. Their popularity surged as LAN networks advanced, allowing for more seamless and competitive multiplayer experience.

2.3.2 Game Mechanics

In the game design community, the term "game mechanics" is preferred over "game rules" because rules are seen as printed instructions that the player is aware of, while mechanics refer to the underlying systems that are hidden from the player and are implemented within the software. Rules and mechanics are related but mechanics are more detailed and concrete so programmers can turn them into code without confusion.(Adams and Dormans 2012)

Usually when the game mechanics of the FPS games are discussed, the first few terms that come to our mind are shooting, aiming, and moving. However, there are seven types of categories of game mechanics: Space, Time, Objects (Attributes, and states), Actions, Rules, Skill and Chance. (Wang 2021)

FPS games are similar in space game mechanics as they have discrete play zones which are made by cutscenes and dungeons such as Fortnite and Apex Legends. Most FPS games are three-dimensional but there are certain differences on the vertical level of the maps. For Quake III Arena, it is the only 1 versus 1 game among the 12 games, its maps are the smallest horizontally, but they have the most vertical layers. In games like CSGO, Call of Duty: Modern Warfare, Battlefield 4, Valorant, most parts of the map are plain ground or two layers. For games like Halo 5, Overwatch, Rainbow Six: Siege, one can see the development of map design for the competitive nature of FPS maps.

For the time mechanics, it depends on how big the map is because the boundaries will limit the play zones of the players. For example, PUBG takes 20-30 minutes per round while CSGO can take only 2-3 minutes per round. Each FPS game may have the same objective, but they vary in detail or quantities. Action mechanics in FPS games also varies according to the games, it might have buy and sell items, scouting and others. Rules mechanics differ in FPS games too, to score in FPS game might require you to kill or survive or reaching the objectives. Different FPS games requires different skills mechanics (memory, puzzle solving) from the players and different chances mechanics are also present in different FPS games (retake site, random airdrops).

2.3.3 Comparison between FPS and other shooting games

Type of shooting	First- person shooter	Third- person shooter
Camera perspective	Player experience game through eyes of character	Player views and controls the character located behind and slightly above or to the side of character
View of game world	Camera positioned at character's eye level, providing direct view of environment from that perspective	Players can see the character they control, often giving them a broader view of their surroundings compared to FPS games.
View of character	Players can see character's hands, weapons, any relevant HUD (health, ammo and objectives)	Players can see character's whole body, allow player to observe movements and interaction with environment

Environmental interaction	Players interact with environment primarily with character’s hands such as opening doors, picking up objects	Incorporate more visible interactions between characters and environments such as climbing obstacles and vaulting over barriers
Aiming and shooting mechanism	Rely on player’s ability to align their view with target using in-game crosshairs, affected by recoil, weapon sway and player movement	More dynamic aiming system where players can see character’s body and aim using cursor. Allow more precise aiming
Example	Call of Duty, Halo	Uncharted, GTA

2.4 Dedicated Server

2.4.1 Definition and function of dedicated server

A dedicated server is a type of Internet hosting in which the client leases an entire server not shared with anyone else. Those big companies or organizations have complete control and this is what makes them secure. High end performance can be expected from dedicated server while they remain expensive.

Dedicated servers are mostly used where company’s computing demands are high. However, they have shortcomings such as exact specifications, limited storage specifications, non-resilient nature and hardware failures. Another drawback of dedicated servers is that the client always pay for the maximum power. (Abidi and Singh 2013)

2.4.2 Types of dedicated servers

Cheap Dedicated server is a great option for individual or small organization that want a greater reliability and performance than that offered by shared website hosting providers.

Standard dedicated servers are usually customized based on customers’ needs. Some companies also offer managed solutions support with migrations and other useful services to manage the account. These servers typically have elements such as air conditioning systems, network connections, backup power, and data backup structures. Compared to low-cost dedicated servers, standard dedicated servers are built better and perform effectively. They are particularly fit for companies that have exceeded their shared hosting or recognize the need for increased performance.

Enterprise Dedicated servers have the highest level of performance and dependability. These servers have hardware that is like which an IT department will but for its application system. Dedicated internet connectivity, cloud direct connections, email account, firewalls and more features can also be added to this dedicated server.

There are also high-performance dedicated servers which are develop with artificial intelligence (AI) and machine learning in mind. It gets utilized by government organizations, big data businesses, and institutions to process and analyze data for strong research activities. For large data analytics and academic computations, high-performance dedicated servers are often used. They contain exclusive GPU hardware made by renowned manufacturers.

2.5 Previous Works

Two gamers, Minh "Gooseman" Le and Jess "Cliffe" Cliffe, developed a mod for Half-Life in the late 1990s. This mod, released in June 1999, stood out from typical FPS games by combining tactical gameplay with team-based action.

The success of the original mod prompted Valve to collaborate with its creators to develop a standalone game, Counter-Strike 1.0 (as shown in **Figure 2.28**), which was released in November 2000. This was a significant milestone in Counter-Strike's history, with Counter-Strike 1.0 rapidly gaining widespread popularity.

Figure 2.28 Counter-Strike 1.0

Blackshot is a multiplayer FPS game developed by Vertigo Games, initially released in Korea for Microsoft Windows on November 15, 2007 .

The first release of BlackShot (as shown in **Figure 2.29**) featured two main game modes: Team Flag Match and Explosion Mission (now Search and Destroy). In Team Flag Match, teams protect flag runners on small maps with infinite respawns, earning points for flag captures and kills. Explosion Mission involves attackers planting a bomb and defenders preventing it. In 2011, Team Death Match was added, where the first team to reach a set number of kills wins.

Figure 2.29 Blackshot

2.6 Discussion

To make a multiplayer game, whether it’s an FPS game or any other kind of genre of game, a dedicated server is needed for all the players to be in the same virtual world to interact with each other. Hence, to make a multiplayer FPS game, a server for players to connect with each other through networking is needed.

In this project, a cheap dedicated server as we plan the total number of players in our players to be less than 20 or 30 persons like that, so a high-performance server will not be needed.

CHAPTER 3: DESIGN

3.1 Introduction

In game development, understanding game architecture is crucial to creating an immersive and engaging experience. This chapter delves into the critical components that constitute a game's architecture and also discusses game design and game art.

Game Design explores the foundational elements in a game that drives the player's experience, the key elements to decide whether a game is engaging and fun or not. Gameplay is the pattern defined through game rules, building a connection between the player and the game, challenges and overcoming them. Core mechanics are the basic elements of interaction that make up the game system. Flowboard, on the other hand, is a tool for planning and visualizing the player's journey through the game, while level progression is analyzing the structure and the pace of the game to maintain player engagement. Lastly, the User interface and interaction model is where the importance of intuitive controls and seamless interaction between the players and the game is highlighted.

Game art focuses on the visual and auditory elements of the game to make the player's engagement in the game stronger. The game world is the creation of immersive environments that set the stage for gameplay. Principles behind creating compelling and relatable characters will be discussed in character design, and how the characters the player controls are created. The camera model is then examined to consider how different perspectives and camera movements can enhance the player's experience.

Finally, we will discuss the role of sound in building atmosphere in audio and sound effects as this is the aspect that makes the game more realistic.

By the end of this chapter, readers will gain a comprehensive understanding of the intricate details involved in game architecture. They will understand how meticulous design and art contribute to crafting memorable gaming experiences that captivate players from start to finish.

3.2 Game Architecture

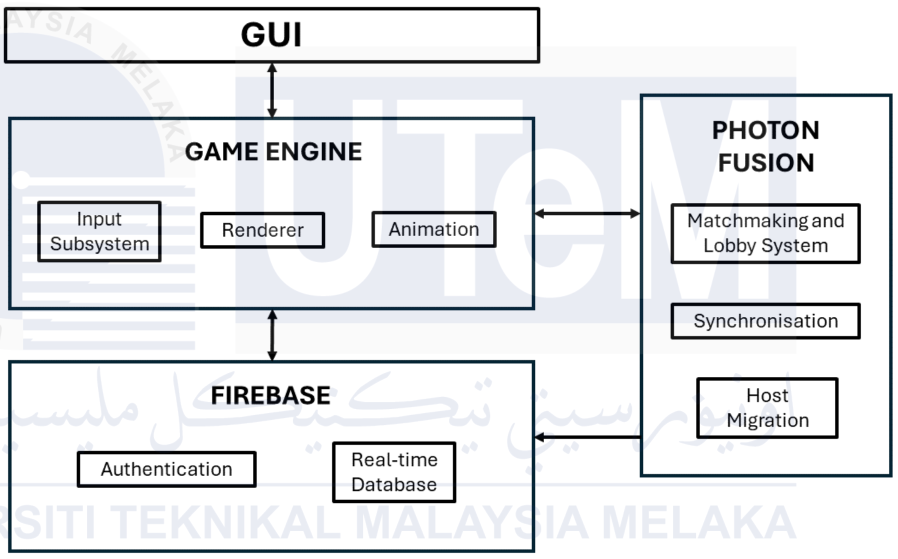


Figure 3.1 Overall Game Architecture of this project

Four Main components are included in this project's game architecture: graphical user interface (GUI), game engine, Firebase, and photon fusion (as shown in Figure 3.1). The architecture will utilize Unity as the game engine, Photon Fusion for real-time networking purposes, and Firebase for backend services.

Unity serves as the core development environment where this project is being created and executed. It provides a foundation for rendering graphics, an input subsystem that captures and responds to the player inputs from their devices, which in this project will be a mouse and keyboard, and animation for game objects to make it more realistic using keyframes and state machines. There are also other components in Unity, such as scenes (game world), audio and sound effects, which will be covered in 3.4 and other components.

GUI refers to elements like menus, Heads-Up Displays (HUD) and interaction prompts that can be developed using the UI system in Unity such as canvas, panels, buttons, etc. Inputs from the GUI are captured and processed using Unity's input system and trigger corresponding game actions by the players.

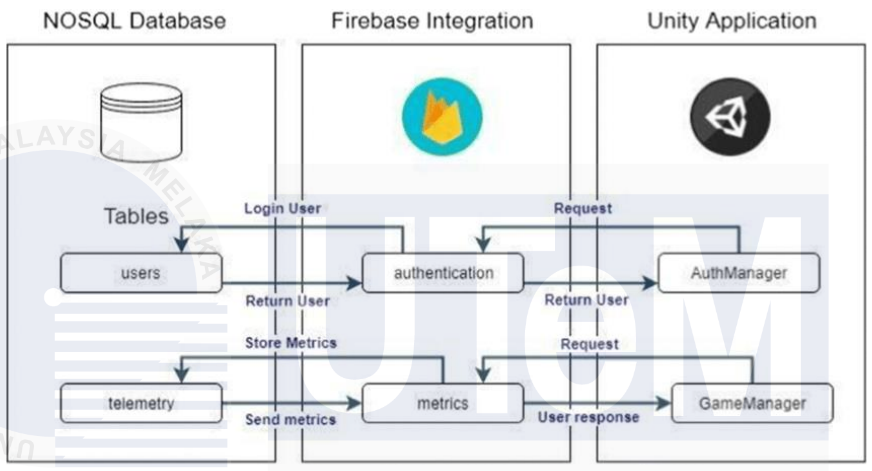


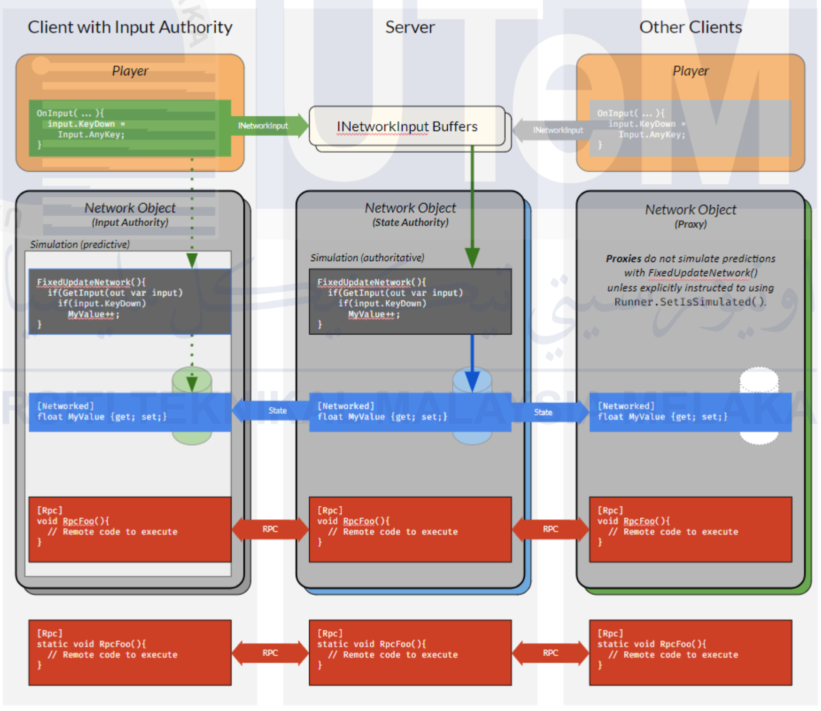
Figure 3.2 Interaction between SQL, Firebase and Unity

The architecture above outlines a clear interaction between the Unity application, firebase integration with its database (NoSQL database) to manage user authentication (login and register account system) and telemetry data (name, kills and deaths) (as shown in **Figure 3.2**).

When a user attempts to log in from the Unity application, the AuthManager sends a login request to the Firebase authentication component. The authentication component processes the login request and queries the "users" table in the NoSQL database whether this user registered before in this database and is entering the correct password. If the user exists, the authentication component retrieves the user data (name, kills and deaths) from the table and returns it to the AuthManager in the Unity application. The AuthManager receives the user data and uses it to manage the user's session within the Unity application.

The GameManager in the Unity application will send the metrics data (kills and deaths) to the Firebase metrics component. The metrics component stores the received metrics data in the telemetry data in the NoSQL database. The GameManager can also request metrics data from Firebase. The metrics component queries the telemetry table for the requested data and sends it back to the GameManager after it retrieved the data. The GameManager receives the metrics data and uses it to update the game state and display the analytics to the user.

By structuring the architecture in this manner, the system ensures efficient user management and telemetry processing, enhancing the overall functionality and user experience of the multiplayer FPS game.



```
OnInput( ... ){
  input.KeyDown =
    Input.AnyKey;
}
```

```
FixedUpdateNetwork(){
  if(GetInput(out var input)
    if(input.KeyDown)
      MyValue++;
}
```

```
[Networked]
float MyValue {get; set;}
```

```
[Rpc]
void RpcFoo(){
  // Remote code to execute
}
```

```
[Rpc]
static void RpcFoo(){
  // Remote code to execute
}
```

Figure 3.3 Core Fusion APIs

The client with input authority captures the input ("OnInput" event) and performs predictive simulation to maintain a smooth game experience (as shown in **Figure 3.3**). The server, which is the player with the state authority, will receive the input and execute authoritative simulation and update the game state, and replicates them to all clients to ensure consistency. Other clients will update their local game state based on the server's authoritative updates. The server maintains the authoritative state and replicates it to all clients to ensure synchronization and game state is consistent across all clients.

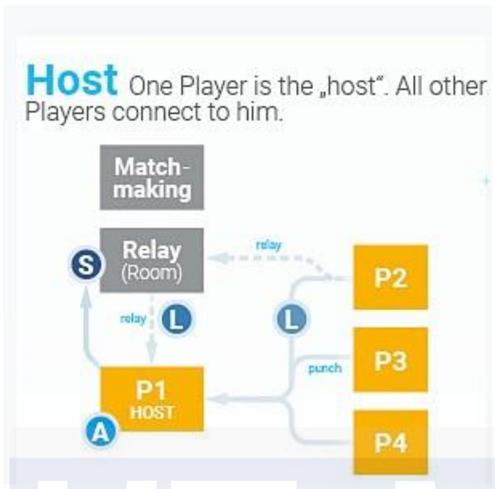

Host One Player is the „host". All other Players connect to him.
Match-making
Relay (Room)
S
relay
L
relay
L
punch
P1 HOST
A
P2
P3
P4

Figure 3.4 Fusion Network Topologies

The player who creates the game session is designated as the host. Hence, the host has full authority and plays the role of a server while other players connect to him (as shown in **Figure 3.4**). When the host disconnects or experiences high latency, Photon Fusion selects a new host among the remaining players and sends them the game state information. The new host then takes over the responsibility of running the game.

In Photon Fusion, players can create their rooms, and other players can search for existing rooms to join and play in the same virtual space. The player who creates the room acts as the host. When the players are in the same room, they have a synchronized view of the game state, including player positions, action, and other game events.

3.3 Game Design

3.3.1 Gameplay

Game Objective

The primary objective of the game is for players to achieve the highest number of kills and the fewest deaths before the countdown timer runs out. Players who obtain the highest kills per death (K/D) will win the match

Player Controls

Movement: WASD or arrow keys for movement.

Shooting: Left mouse button for shooting

Reloading: R key for reloading

Jumping: Space bar for jumping

Throwing grenade: G button for throwing grenade

Chat: T button for typing in chat

Weapons and Items

Primary Weapons: Pistol

Grenades: Frag grenades

Game Environment

Maps: Various maps with different terrains and structure to provide strategic advantages and challenges

Spawn Points: Designated areas where players respawn after being eliminated

3.3.2 Core Mechanics

Player Movement

Players can use basic movement controls, WASD, to move in any direction based on their input. They can also navigate over obstacles or reach higher ground by jumping with the spacebar button.

Aiming and Shooting

Players can use the aiming UI to aim their weapons and the left mouse button to shoot at enemies. They can also use the R key to reload their weapons when ammunition is depleted, but they need to keep track of their ammo count and manage their reloads strategically to increase their chances of winning the match.

Health and Damage

Players have a health bar that decreases when they take damage. Different kinds of weapons and attacks deal varying types of damage.

Communication

Players can communicate with each other using text chat in the game.

Player Customization

Before entering the game, players can customize their characters based on their body parts in the lobby. They can change their heads, tops, bottoms, and other body parts as they desire.

3.3.3 Flowboard

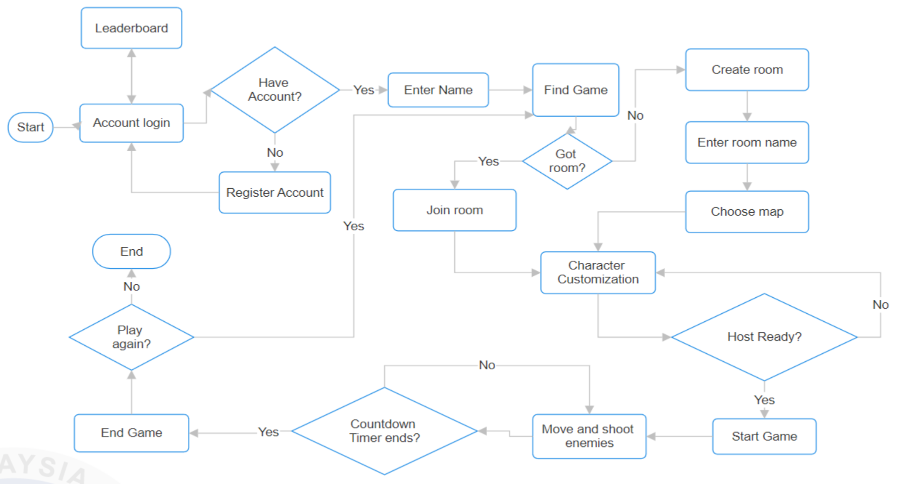


Figure 3.5 Project Flowboard

When players first enter the game, players need to register an account before logging into the game. Players will enter the in-game name, players can join any room in the lobby session. If there are no rooms available, players can create their room by entering the room name and selecting the maps they want to play. Players can customize their characters before the host is ready. After the host is ready, the game will start and players will start to kill each other until the countdown timer of the match ends. The result of the players will be shown, and players can choose to play again or quit the game. Players will be returned to the find room session if they choose to play again (as shown in **Figure 3.5**).

3.3.4 User Interface / Interaction Model

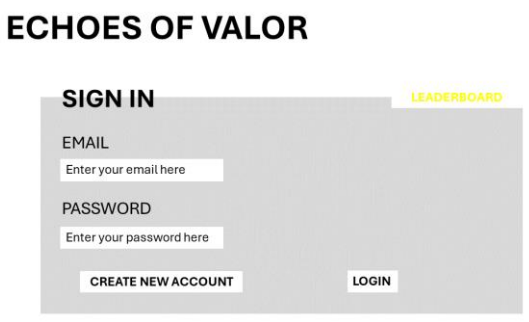


Figure 3.6 Account Login Page

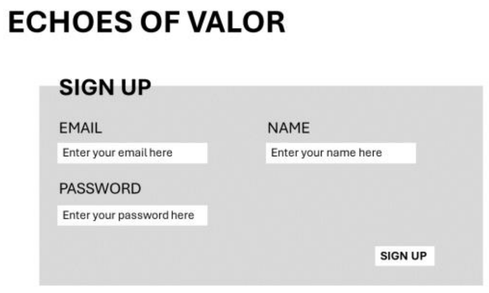


Figure 3.7 Account Register Page

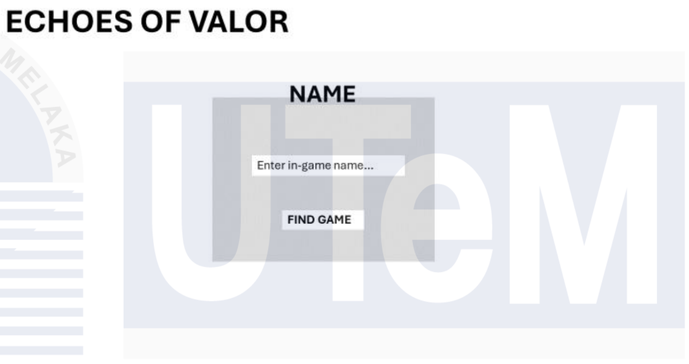


Figure 3.8 Name Insert Page

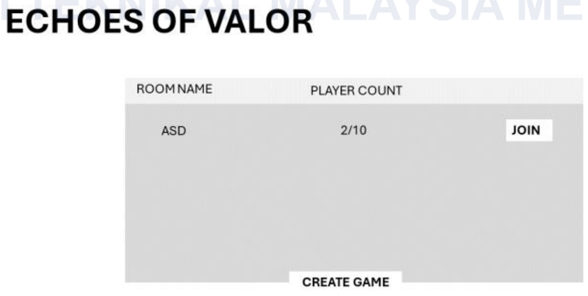


Figure 3.9 Room Session Page

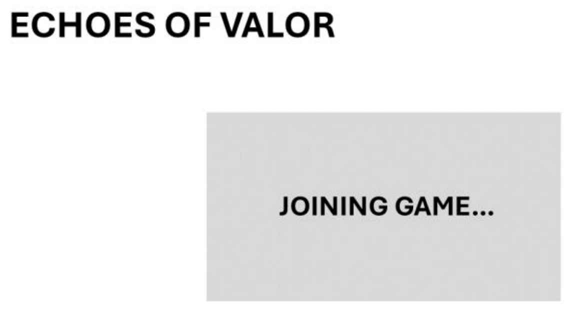


Figure 3.10 Joining Room Page

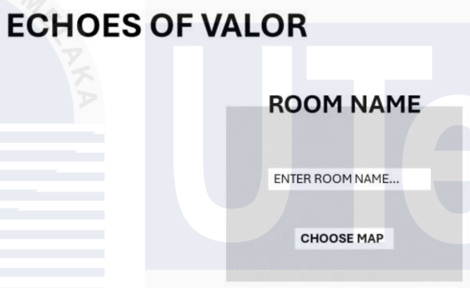


Figure 3.11 Room Naming Page

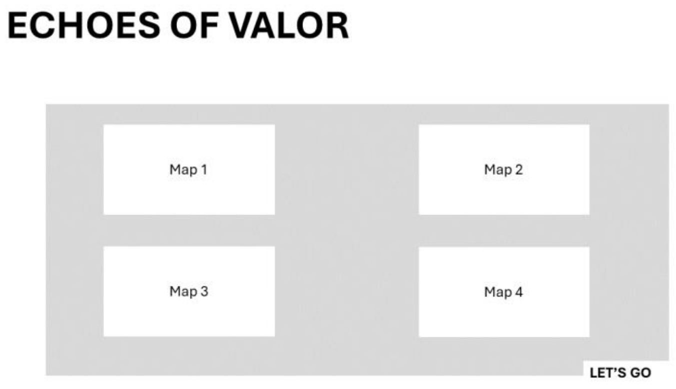


Figure 3.12 Map Selection Page

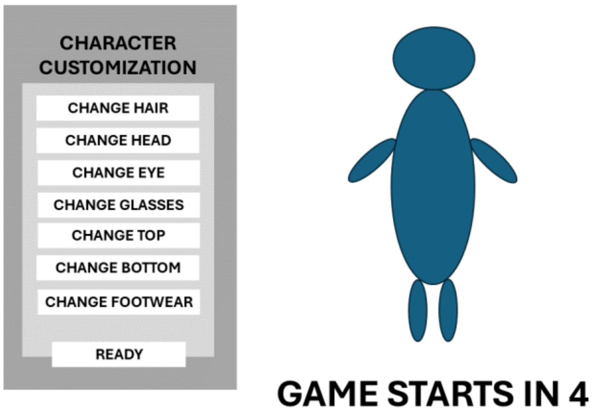


Figure 3.13 Character Customization Page

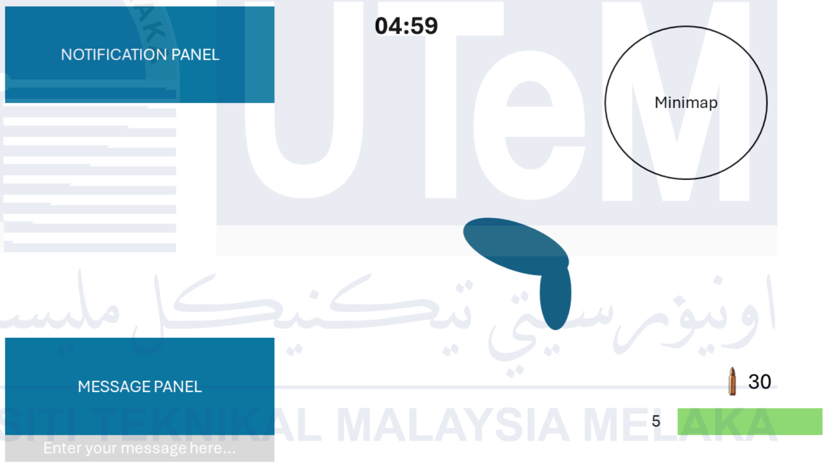


Figure 3.14 In-Game UI Interface

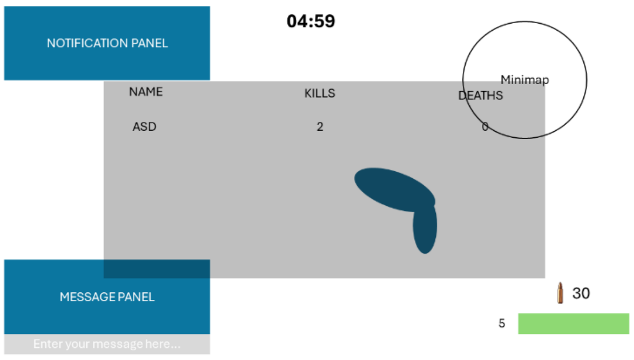


Figure 3.15 In-Game UI with scoreboard Interface

3.4 Game Art

3.4.1 Game World

In this project, players are transported to two diverse environments that they can choose from while creating the rooms: the Factory area and the Jungle Area. Each map offers unique tactical challenges and visual aesthetics that the players can use to create different playstyles and strategies.

The factory area is an industrial zone filled with sprawling warehouses and intricate piping systems. This map is characterized by its post-apocalyptic ambience, providing a gritty and intense battlefield for players. The factory offers a mix of narrow passageways and expansive open areas, allowing for close-quarters combat as well as long-range engagements. Rooftops and multi-level structures provide opportunities for vertical combat and ambush from above. Numerous buildings and industrial equipment are scattered across the map, offering ample cover and strategic positions for ambushes and defensive manoeuvres.

The Jungle area emphasizes stealth and guerrilla warfare, a stark contrast to the industrial factory setting. The uneven terrain includes hills, lakes, and rocky outcrops. Players can hide in the hills, making an unexpected ambush on enemies. The jungle map features abandoned buildings that can be used as strategic points for ambushes.

3.4.2 Camera Model

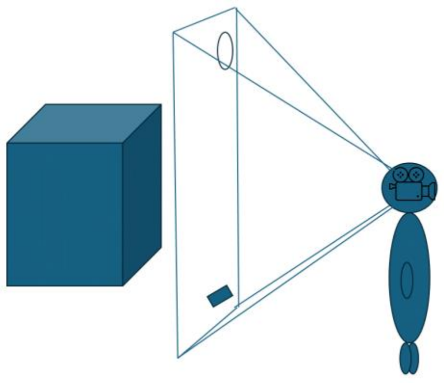


Figure 3.16 Player Camera

In the game's architecture, a camera is integrated as a sub-child of the player prefab, serving as the primary perspective through which players view the game world. This camera is positioned to offer a first-person perspective, meaning players experience the game through the eyes of their character, enhancing immersion and situational awareness.

To complement this setup, a UI canvas is also attached as a sub-child of the camera. This canvas includes all essential UI elements, such as the minimap, health bar, ammo count, and other crucial in-game information. By parenting the canvas to the camera, the UI elements remain consistently in view from the player's perspective, regardless of how the player moves or rotates within the game world. This ensures that vital information is always accessible and properly aligned within the player's field of vision, maintaining an intuitive and seamless gaming experience.

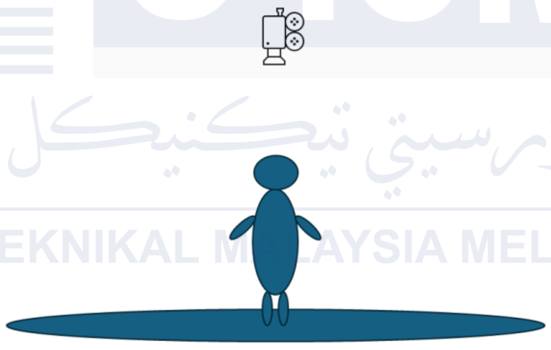


Figure 3.17 Minimap Camera

A secondary camera is integrated into the player's prefab as a sub-child object. This camera is strategically positioned above the player's model to effectively track and follow the player's movements along the x and z axes on the map. The y-axis position of the camera remains fixed to maintain a consistent overhead view, providing a tactical perspective of the player's surroundings.

By attaching this camera as a child of the player prefab, it inherently inherits the player's positional changes. As the player navigates the game environment, the camera dynamically updates its position to mirror the player's x and z coordinates. This ensures that the camera moves seamlessly with the player, offering a real-time, bird's-eye view of the player's location and movements within the game world.

3.4.3 Audio/Sound Effect

Audio and sound effects play a crucial role in creating an immersive experience in multiplayer FPS games. Sound can provide players with vital information about their surroundings and contribute to the overall enjoyment of the game.

Gunfire should be realistic and convey the power of the weapon. The sound of the gunfire should also vary based on the distance and direction from one player to another player, with closer shots sounding louder and more immediate. Sounds associated with reloading provide feedback and make weapon interactions more satisfying.

3.5 Conclusion

This chapter discusses the game architecture needed to build the multiplayer FPS game. The overall architecture includes a GUI for players to interact with, a game engine (Unity), a firebase for authentication, and photon fusion for networking.

The gameplay and the game mechanics are listed in this chapter, the flowboard of the overall game is drafted and the user interface for each scene of the game is also sketched.

The visual and audio side of the game is also determined in this chapter to ensure how the overall game will look like in the players’ eyes and how the audio is presented to the players.

CHAPTER 4: IMPLEMENTATION

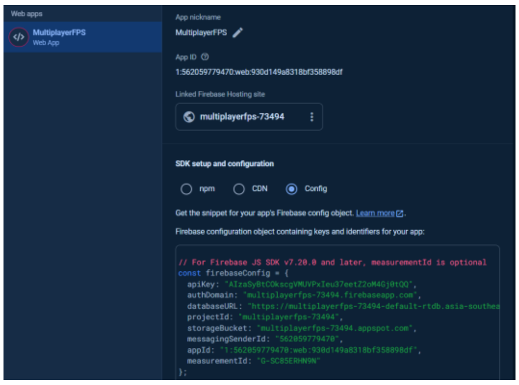
4.1 Introduction

This chapter will focus on how we integrate the game architecture to make the game work and how it achieves its objectives. The game art will also be delved into to discuss how the audio and visual elements are implemented into the game. This chapter will also discuss the project's version control procedures.

4.2 Integration of Game Architecture

4.2.1 Authentication

The first step is creating a Firebase project to store all our project's accounts and data. After choosing the web platform for our Firebase project, the URL to the database and the API key of the Firebase configuration are found in the Firebase configuration and declared in the class of the project to connect our project to our Firebase.



App nickname
MultiplayerFPS

App ID
1:562059779470:web:930d149a8318bf358898df

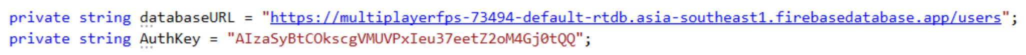
Linked Firebase Hosting site
multiplayerfps-73494

Get the snippet for your app's Firebase config object. Learn more.

Firebase configuration object containing keys and identifiers for your app:

```
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "AIzaSyBtCOkscgVMUVPxIeu37eetZ2oM4Gj0tQQ",
  authDomain: "multiplayerfps-73494.firebaseapp.com",
  databaseURL: "https://multiplayerfps-73494-default-rtdb.asia-southea
  projectId: "multiplayerfps-73494",
  storageBucket: "multiplayerfps-73494.appspot.com",
  messagingSenderId: "562059779470",
  appId: "1:562059779470:web:930d149a8318bf358898df",
  measurementId: "G-SC85ERHN9N"
};
```

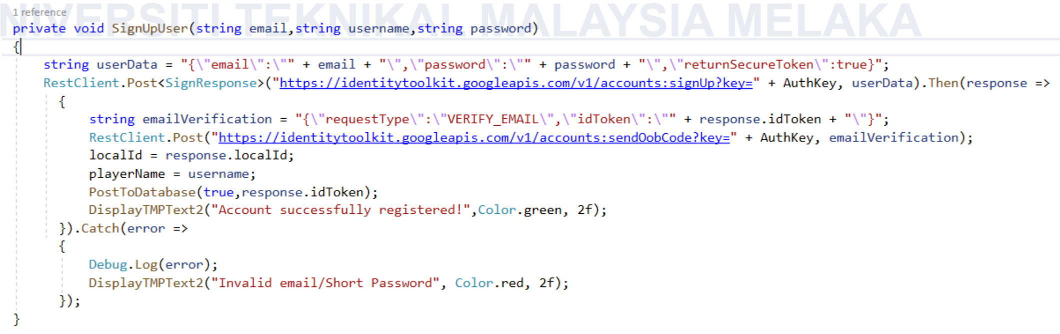
Figure 4.1 Firebase configuratiration



```
private string databaseURL = "https://multiplayerfps-73494-default-rtdb.asia-southeast1.firebasedatabase.app/users";
private string AuthKey = "AIzaSyBtCOkscgVMUVPxIeu37eetZ2oM4Gj0tQQ";
```

Figure 4.2 Declared of URL and auth key in class

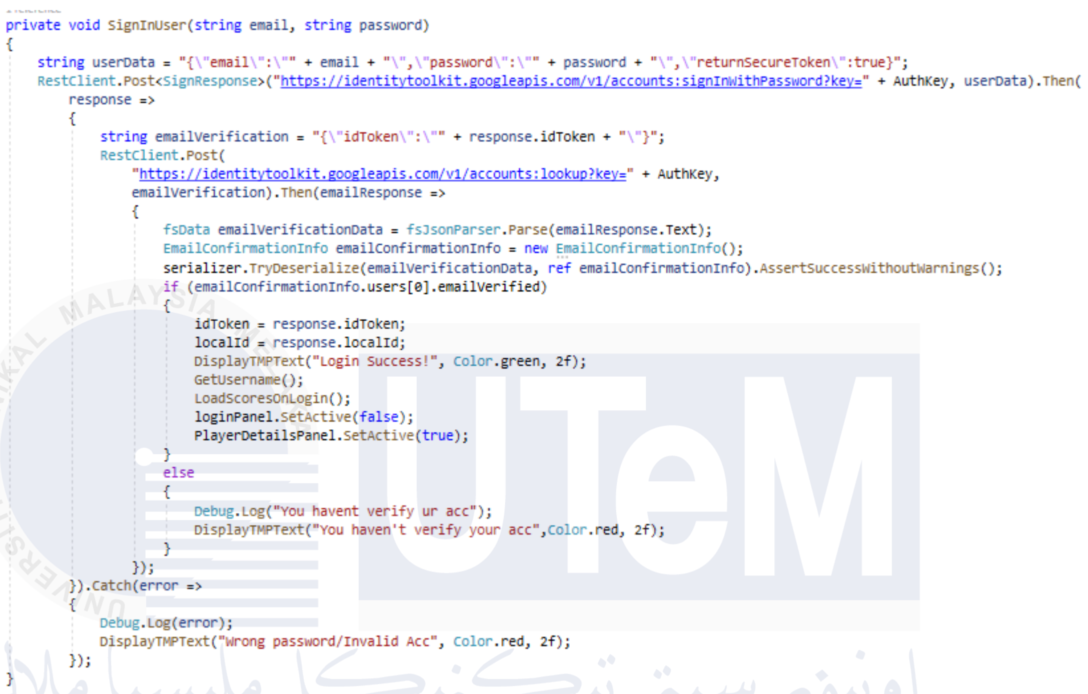
After the connection is established, the first function created by interacting with Firebase Authentication is registering the player account. The SignUpUser function will take the player's email address, username, and password from the input in the game. A JSON string "userData" will contain the email, username, password, and a flag to request a secure token. This JSON string will be sent to Firebase Authentication to create a new user account. A POST request is made to the Firebase Authentication endpoint to sign up a new user. The AuthKey, which is the API key, is appended to the URL to authenticate the request. If the signup is successful, the Then block is executed. Another JSON string is created to request email verification, including the idToken from the signup response. A POST request is sent to Firebase to send a verification email to the newly registered user. The message "Account successfully registered" will be shown on the user interface, while the message "Invalid email/Short Password" will be displayed if the player fails to enter a valid email or short password (as shown in **Figure 4.3**).



```
1 reference
private void SignUpUser(string email,string username,string password)
{
    string userData = "{\"email\":\"" + email + "\",\"password\":\"" + password + "\",\"returnSecureToken\":true}";
    RestClient.Post<SignResponse>("https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=" + AuthKey, userData).Then(response =>
     {
         string emailVerification = "{\"requestType\":\"VERIFY_EMAIL\",\"idToken\":\"" + response.idToken + "\"}";
         RestClient.Post("https://identitytoolkit.googleapis.com/v1/accounts:sendOobCode?key=" + AuthKey, emailVerification);
         localId = response.localId;
         playerName = username;
         PostToDatabase(true,response.idToken);
         DisplayTMPText2("Account successfully registered!",Color.green, 2f);
     }).Catch(error =>
     {
         Debug.Log(error);
         DisplayTMPText2("Invalid email/Short Password", Color.red, 2f);
     });
}
```

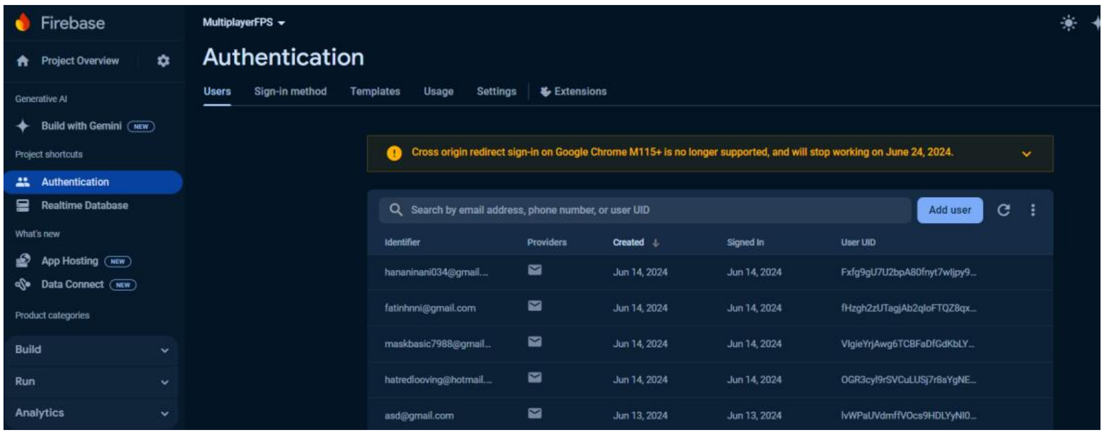
Figure 4.3 SignUpUser function

The SignInUser function authenticates an existing user by verifying their email and password with Firebase Authentication. If the authentication is successful and the email is verified, the function retrieves user data and updates the UI accordingly. A JSON string "userData" will contain the email, username, password, and a flag to request a secure token. The JSON string will be sent to the Firebase Authentication to sign in the user. The POST request is made to the Firebase Authentication endpoint to sign in the user. The AuthKey is appended to the URL to authenticate the request. Then, the email verification status is checked, and the local variables “idToken” and “localId” are updated if the email is verified. It displays appropriate messages to the user based on the outcome of the sign-in process (as shown in **Figure 4.4**).



```
private void SignInUser(string email, string password)
{
    string userData = "{\"email\":\"" + email + "\",\"password\":\"" + password + "\",\"returnSecureToken\":true}";
    RestClient.Post<SignResponse>("https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=" + AuthKey, userData).Then(
        response =>
        {
            string emailVerification = "{\"idToken\":\"" + response.idToken + "\"}";
            RestClient.Post(
                "https://identitytoolkit.googleapis.com/v1/accounts:lookup?key=" + AuthKey,
                emailVerification).Then(emailResponse =>
                {
                    fsData emailVerificationData = fsJsonParser.Parse(emailResponse.Text);
                    EmailConfirmationInfo emailConfirmationInfo = new EmailConfirmationInfo();
                    serializer.TryDeserialize(emailVerificationData, ref emailConfirmationInfo).AssertSuccessWithoutWarnings();
                    if (emailConfirmationInfo.users[0].emailVerified)
                    {
                        idToken = response.idToken;
                        localId = response.localId;
                        DisplayTMPText("Login Success!", Color.green, 2f);
                        GetUsername();
                        LoadScoresOnLogin();
                        loginPanel.SetActive(false);
                        PlayerDetailsPanel.SetActive(true);
                    }
                    else
                    {
                        Debug.Log("You havent verify ur acc");
                        DisplayTMPText("You haven't verify your acc",Color.red, 2f);
                    }
                });
        }).Catch(error =>
        {
            Debug.Log(error);
            DisplayTMPText("Wrong password/Invalid Acc", Color.red, 2f);
        });
}
```

Figure 4.4 SignInUser function



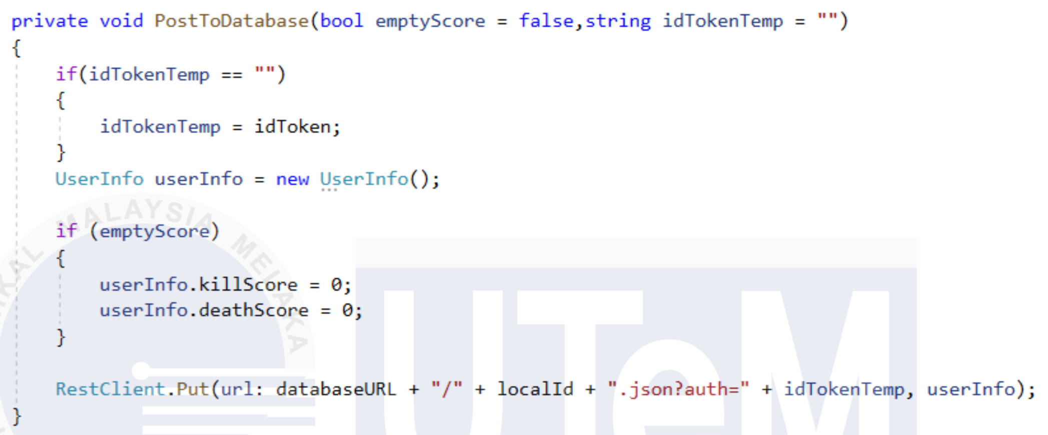
Cross origin redirect sign-in on Google Chrome M115+ is no longer supported, and will stop working on June 24, 2024.

Identifier	Providers	Created	Signed In	User UID
hananinani034@gmail...	✉	Jun 14, 2024	Jun 14, 2024	Fxfg9gU7U2bpA80fnyt7wljpy9...
fatinhnni@gmail.com	✉	Jun 14, 2024	Jun 14, 2024	fHzgh2zUTagjAb2qloFTQZ8qx...
maskbasic7988@gmail...	✉	Jun 14, 2024	Jun 14, 2024	VlgieYrjAwg6TCBFaDfGdKbLY...
hatredlooving@hotmail...	✉	Jun 14, 2024	Jun 14, 2024	OGR3cyl9rSVCuLUSj7r8sYgNE...
asd@gmail.com	✉	Jun 13, 2024	Jun 13, 2024	lvWPaUVdmffVOcs9HDLYyNI0...

Figure 4.5 Firebase Authentication

4.2.2 Real-time Database System

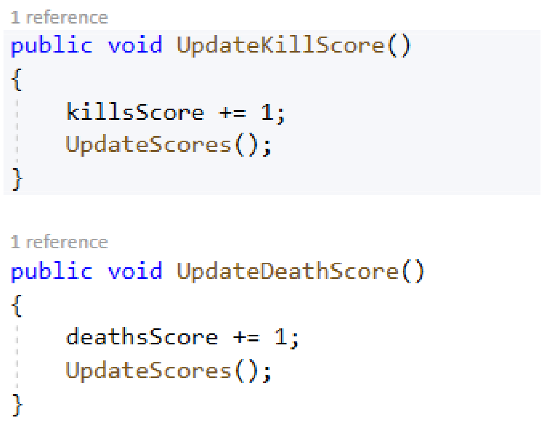
The PostToDatabase function will first set up the players' kill score and death score to be 0 when they first register in the game by sending a put request to the Firebase at the URL corresponding to the user's localId to find out which player it is (as shown in **Figure 4.6**).



```
private void PostToDatabase(bool emptyScore = false,string idTokenTemp = "")
{
    if(idTokenTemp == "")
    {
        idTokenTemp = idToken;
    }
    UserInfo userInfo = new UserInfo();

    if (emptyScore)
    {
        userInfo.killScore = 0;
        userInfo.deathScore = 0;
    }

    RestClient.Put(url: databaseURL + "/" + localId + ".json?auth=" + idTokenTemp, userInfo);
}
```

Figure 4.6 PostToDatabase function

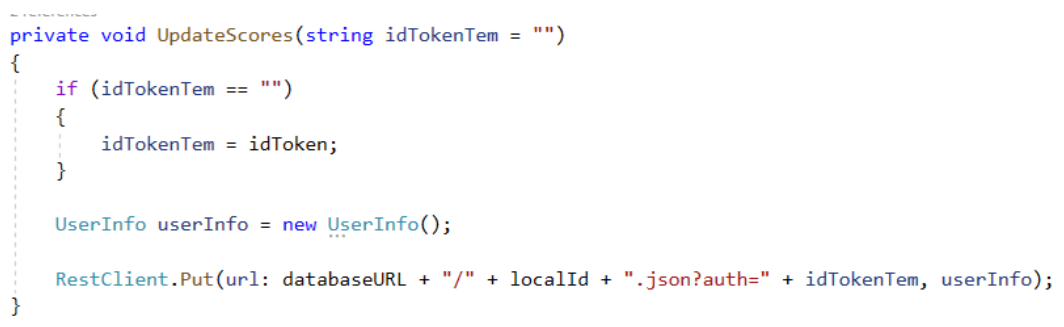
The function will update players' kill and death scores (as shown in Figure **4.7**).



```
1 reference
public void UpdateKillScore()
{
    killsScore += 1;
    UpdateScores();
}

1 reference
public void UpdateDeathScore()
{
    deathsScore += 1;
    UpdateScores();
}
```

Figure 4.7 UpdateKillScore and UpdateDeathScore function

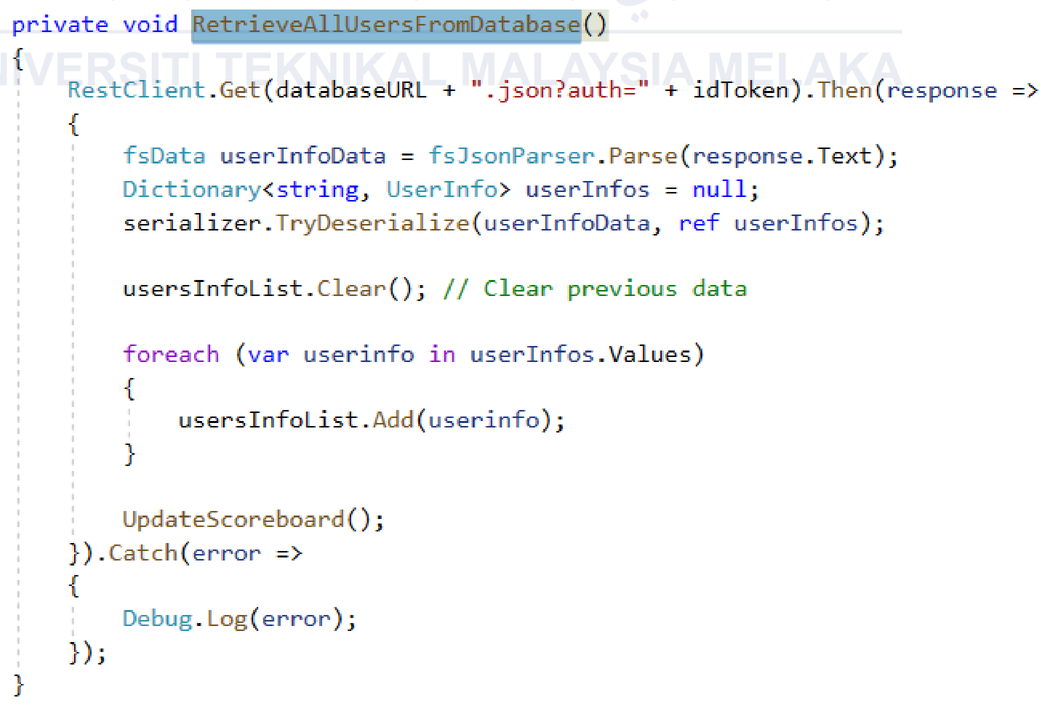
The UpdateScores function will update the score in the Firebase Realtime Database. It creates a new userInfo object to replace the current kill and death scores with the new data. Then, the Put requests to update the user's information at the specified URL, the project's Firebase system (as shown in **Figure 4.8**).



```
private void UpdateScores(string idTokenTem = "")
{
    if (idTokenTem == "")
    {
        idTokenTem = idToken;
    }

    UserInfo userInfo = new UserInfo();

    RestClient.Put(url: databaseURL + "/" + localId + ".json?auth=" + idTokenTem, userInfo);
}
```

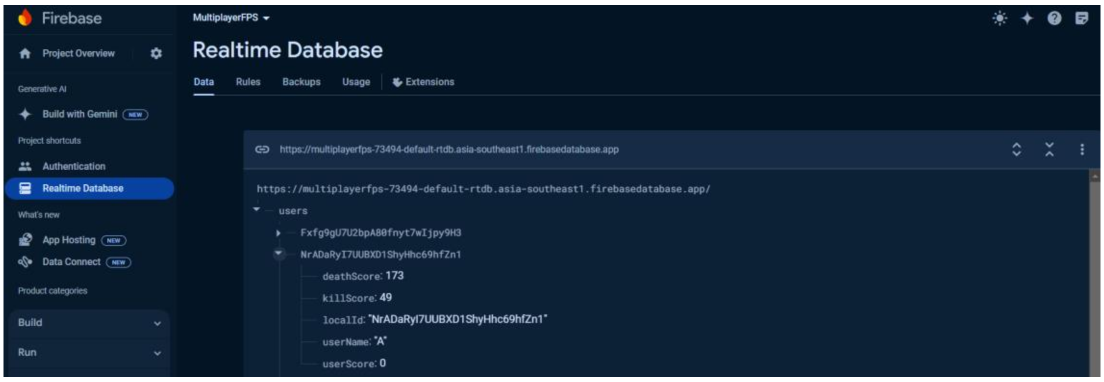
Figure 4.8 UpdateScores function

The RetrieveAllUsersFromDatabase function will then retrieve all users from the Firebase Realtime Database. The function sends a GET request to Firebase to fetch all users' data. It parses the response text into fsData, deserialises the fsData into a dictionary and clears the existing userInfoList. Finally, it adds the userInfo from the dictionary that obtained the data from the firebase to the userInfoList, and the scores are updated on the scoreboard (as shown in **Figure 4.9**).



```
private void RetrieveAllUsersFromDatabase()
{
    RestClient.Get(databaseURL + ".json?auth=" + idToken).Then(response =>
    {
        fsData userInfoData = fsJsonParser.Parse(response.Text);
        Dictionary<string, UserInfo> userInfos = null;
        serializer.TryDeserialize(userInfoData, ref userInfos);

        usersInfoList.Clear(); // Clear previous data

        foreach (var userinfo in userInfos.Values)
        {
            usersInfoList.Add(userinfo);
        }

        UpdateScoreboard();
    }).Catch(error =>
    {
        Debug.Log(error);
    });
}
```

Figure 4.9 RetrieveAllUsersFromDatabase function

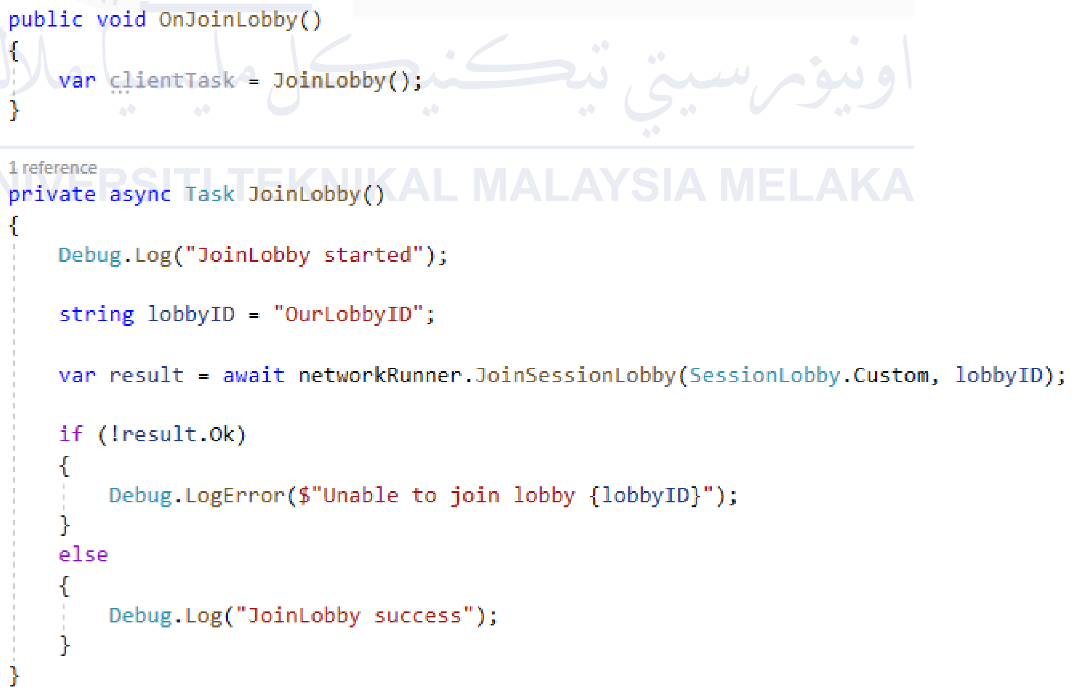


```
https://multiplayerfps-73494-default-rtdb.asia-southeast1.firebasedatabase.app

https://multiplayerfps-73494-default-rtdb.asia-southeast1.firebasedatabase.app/
users
    Fxfg9gU7U2bpA80fnyt7wIjpy9H3
    NrADaRyI7UUBXD1ShyHhc69hfZn1
        deathScore: 173
        killScore: 49
        localId: "NrADaRyI7UUBXD1ShyHhc69hfZn1"
        userName: "A"
        userScore: 0
```

Figure 4.10 Firebase Database System

4.2.3 Matchmaking

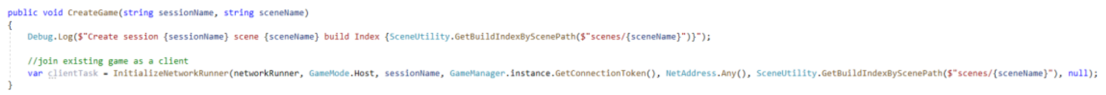
Players will first join a lobby where they can find rooms to play. The JoinLobby function is called asynchronously, as different players join the game at different times. The players then use the network runner to join a session lobby, allowing players to find and join games within that lobby (as shown in **Figure 4.11**).



```
public void OnJoinLobby()
{
    var clientTask = JoinLobby();
}

1 reference
private async Task JoinLobby()
{
    Debug.Log("JoinLobby started");

    string lobbyID = "OurLobbyID";

    var result = await networkRunner.JoinSessionLobby(SessionLobby.Custom, lobbyID);

    if (!result.Ok)
    {
        Debug.LogError($"Unable to join lobby {lobbyID}");
    }
    else
    {
        Debug.Log("JoinLobby success");
    }
}
```

Figure 4.11 Join lobby functions

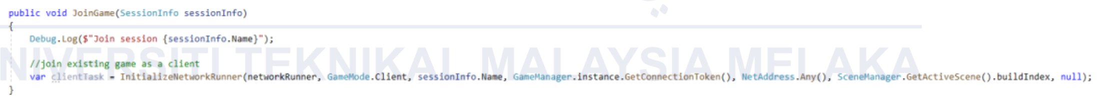
When creating a room, players start a new game session in which they act as the host. The sessionName will be where the host player names the room. The function retrieves the connection-token from the GameManager instance. The connection token is used for authentication and to ensure that only authorized players can join the session. The net address specifies that the network runner should listen for connections on any available network address so other players can see the room created by the host (as shown in **Figure 4.12**).



```
public void CreateGame(string sessionName, string sceneName)
{
    Debug.Log($"Create session {sessionName} scene {sceneName} build Index {SceneUtility.GetBuildIndexByScenePath($"scenes/{sceneName}")}");

    //join existing game as a client
    var clientTask = InitializeNetworkRunner(networkRunner, GameMode.Host, sessionName, GameManager.instance.GetConnectionToken(), NetAddress.Any(), SceneUtility.GetBuildIndexByScenePath($"scenes/{sceneName}"), null);
}
```

Figure 4.12 CreateGame function

Other players will join the hosted session as clients. sessionInfo.Name is the name of the game session to join. This is retrieved from the sessionInfo object passed to the function. Then, the clients will retrieve the connection-token from the GameManager instance. The connection token is used for authentication and ensuring that only authorized players can join the session (as shown in **Figure 4.13**).

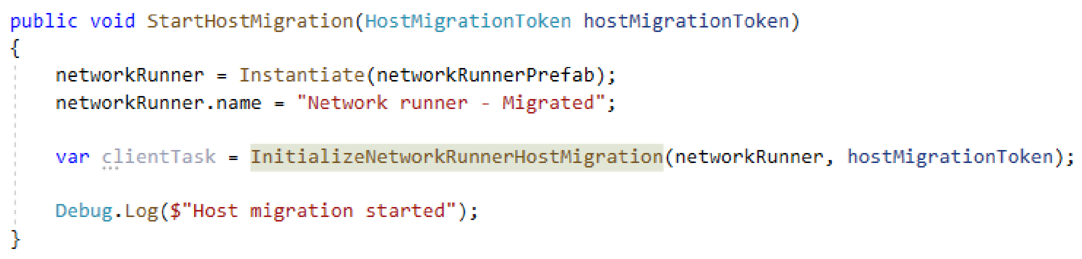


```
public void JoinGame(SessionInfo sessionInfo)
{
    Debug.Log($"Join session {sessionInfo.Name}");

    //join existing game as a client
    var clientTask = InitializeNetworkRunner(networkRunner, GameMode.Client, sessionInfo.Name, GameManager.instance.GetConnectionToken(), NetAddress.Any(), SceneManager.GetActiveScene().buildIndex, null);
}
```

Figure 4.13 JoinGame function

4.2.4 Host-Migration

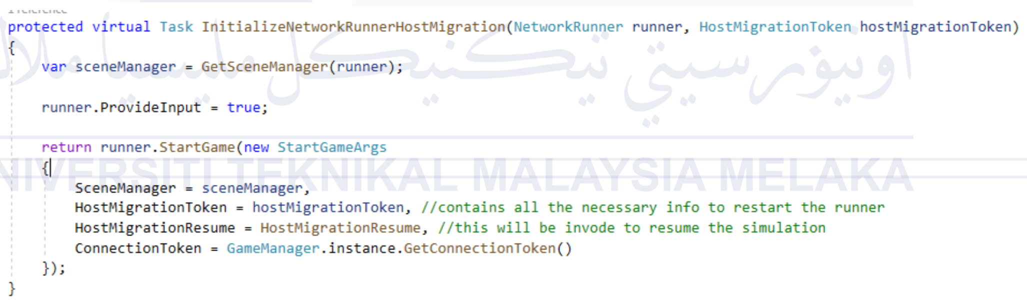
Host migration occurs when the original host disconnects or leaves the game, and the host's responsibilities are transferred to another player. The StartHostMigration function is initiated when the host leaves the game. A new instance of the network runner will be created to handle the migrated host responsibilities. Then, the InitializeNetworkRunnerHostMigration function will be called for the necessary parameters for host migration (as shown in **Figure 4.14**).



```
public void StartHostMigration(HostMigrationToken hostMigrationToken)
{
    networkRunner = Instantiate(networkRunnerPrefab);
    networkRunner.name = "Network runner - Migrated";

    var clientTask = InitializeNetworkRunnerHostMigration(networkRunner, hostMigrationToken);

    Debug.Log($"Host migration started");
}
```

Figure 4.14 StartHostMigration function

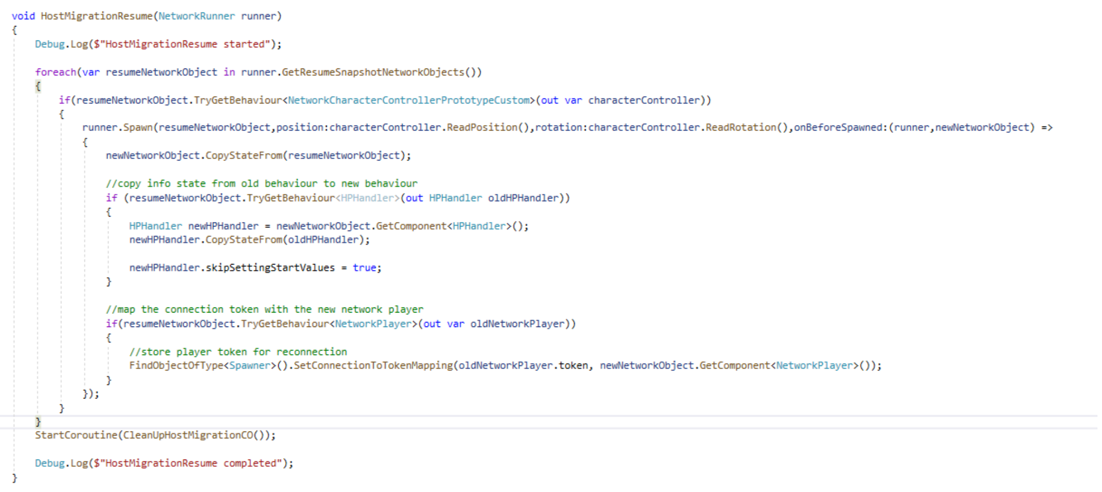
The InitializeNetworkRunnerHostMigration function sets up the new network runner to resume the game session with the host migration token. The GetSceneManager ensures the scene keeps running while the provideInput sets the network runner to handle the player inputs. Inside the StartGame, the sceneManager is used to handle the scenes, the hostmigrationtoken contains all the necessary information to restart the runner, hostmigrationresume to resume the simulation and connectionToken to retrieve the connectionToken from the GameManager (as shown in **Figure 4.15**).



```
protected virtual Task InitializeNetworkRunnerHostMigration(NetworkRunner runner, HostMigrationToken hostMigrationToken)
{
    var sceneManager = GetSceneManager(runner);

    runner.ProvideInput = true;

    return runner.StartGame(new StartGameArgs
    {
        SceneManager = sceneManager,
        HostMigrationToken = hostMigrationToken, //contains all the necessary info to restart the runner
        HostMigrationResume = HostMigrationResume, //this will be invode to resume the simulation
        ConnectionToken = GameManager.instance.GetConnectionToken()
    });
}
```

Figure 4.15 InitializeNetworkRunnerHostMigration function

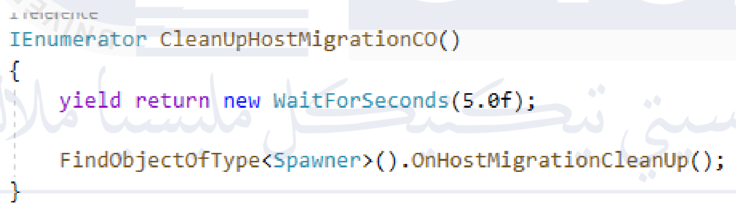
The HostMigrationResume is called to resume the game session after the host migration. The GetResumeSnapshotNetworkObjects retrieves the network objects that need to be resumed, such as the players' health and current position. Then, the cleanUpHostMigration function is called (as shown in **Figure 4.16**).



```
void HostMigrationResume(NetworkRunner runner)
{
    Debug.Log($"HostMigrationResume started");

    foreach(var resumeNetworkObject in runner.GetResumeSnapshotNetworkObjects())
    {
        if(resumeNetworkObject.TryGetBehaviour<NetworkCharacterControllerPrototypeCustom>(out var characterController))
        {
            runner.Spawn(resumeNetworkObject,position:characterController.ReadPosition(),rotation:characterController.ReadRotation(),onBeforeSpawned:(runner,newNetworkObject) =>
            {
                newNetworkObject.CopyStateFrom(resumeNetworkObject);

                //copy info state from old behaviour to new behaviour
                if (resumeNetworkObject.TryGetBehaviour<HPHandler>(out HPHandler oldHPHandler))
                {
                    HPHandler newHPHandler = newNetworkObject.GetComponent<HPHandler>();
                    newHPHandler.CopyStateFrom(oldHPHandler);

                    newHPHandler.skipSettingStartValues = true;
                }

                //map the connection token with the new network player
                if(resumeNetworkObject.TryGetBehaviour<NetworkPlayer>(out var oldNetworkPlayer))
                {
                    //store player token for reconnection
                    FindObjectOfType<Spawner>().SetConnectionToTokenMapping(oldNetworkPlayer.token, newNetworkObject.GetComponent<NetworkPlayer>());
                }
            });
        }
    }
    StartCoroutine(CleanUpHostMigrationCO());

    Debug.Log($"HostMigrationResume completed");
}
```

Figure 4.16 HostMigrationResume function

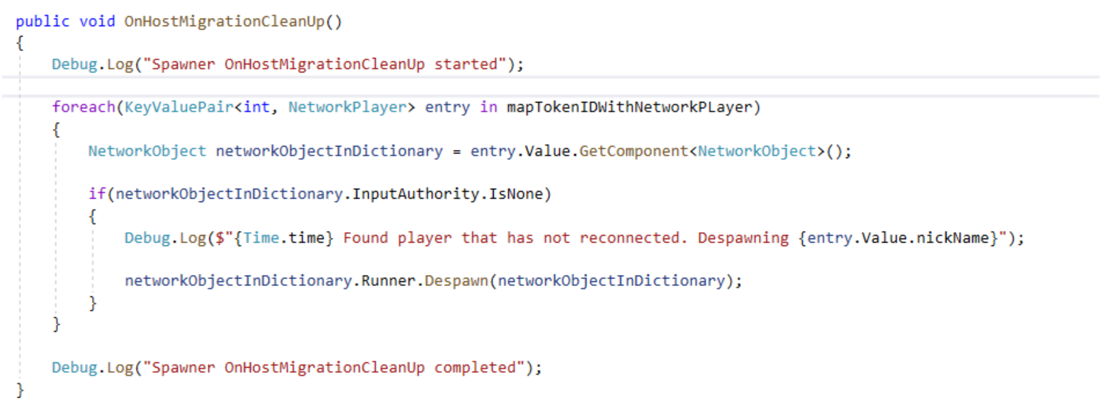
The CleanUpHostMigrationCo will wait 5 seconds before the host migration is completed and calls the OnHostMigrationCleanUp function to clean up the old host (as shown in **Figure 4.17**).



```
IEnumerator CleanUpHostMigrationCO()
{
    yield return new WaitForSeconds(5.0f);

    FindObjectOfType<Spawner>().OnHostMigrationCleanUp();
}
```

Figure 4.17 CleanUpHostMigrationCo coroutine

The OnHostMigrationCleanUp function will retrieve all the network players in the game through the for loop. Then, it will check whether the network object, the network player, has input authority or not. If the network object does not have input authority, the player is disconnected, and the player will be despawned to remove it from the game session (as shown in **Figure 4.18**).



```
public void OnHostMigrationCleanUp()
{
    Debug.Log("Spawner OnHostMigrationCleanUp started");

    foreach(KeyValuePair<int, NetworkPlayer> entry in mapTokenIDWithNetworkPLayer)
    {
        NetworkObject networkObjectInDictionary = entry.Value.GetComponent<NetworkObject>();

        if(networkObjectInDictionary.InputAuthority.IsNone)
        {
            Debug.Log($"{Time.time} Found player that has not reconnected. Despawning {entry.Value.nickName}");

            networkObjectInDictionary.Runner.Despawn(networkObjectInDictionary);
        }
    }

    Debug.Log("Spawner OnHostMigrationCleanUp completed");
}
```

Figure 4.18 OnHostMigrationCleanUp function

4.3 Creation of Game Art

4.3.1 Production of 3D Modelling

In this project, since character customization is implemented, the character will be produced containing different body parts we're using. The parts are the body, hair, head, the bottom outfit (trousers), footwear (shoes), and the top outfit (t-shirt, jacket, etc.). An armature is also rigged for the animation of characters (as shown in **Figure 4.19**).

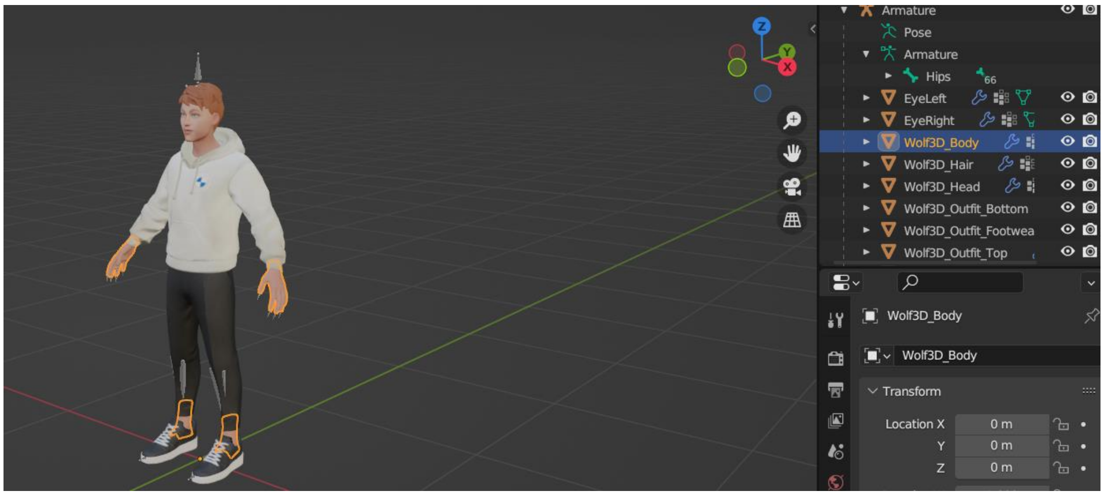


Figure 4.19 Characters made of different body parts

Different characters are produced first, so players can make various choices while characterizing their characters. The purpose of modelling the whole character instead of the body parts is to ensure the armature of the same body parts is in the same place so the animation in Unity can run smoothly (as shown in **Figure 4.20** and **Figure 4.21**).

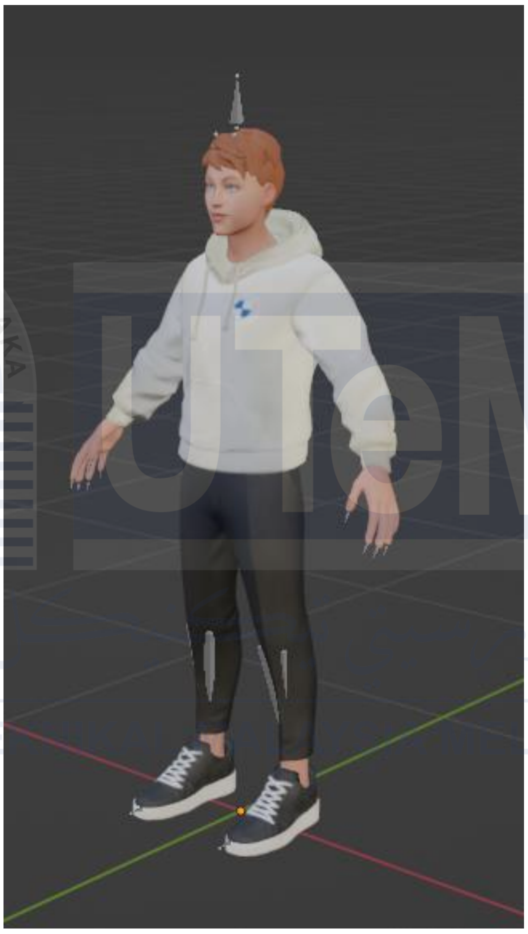


Figure 4.20 Character A

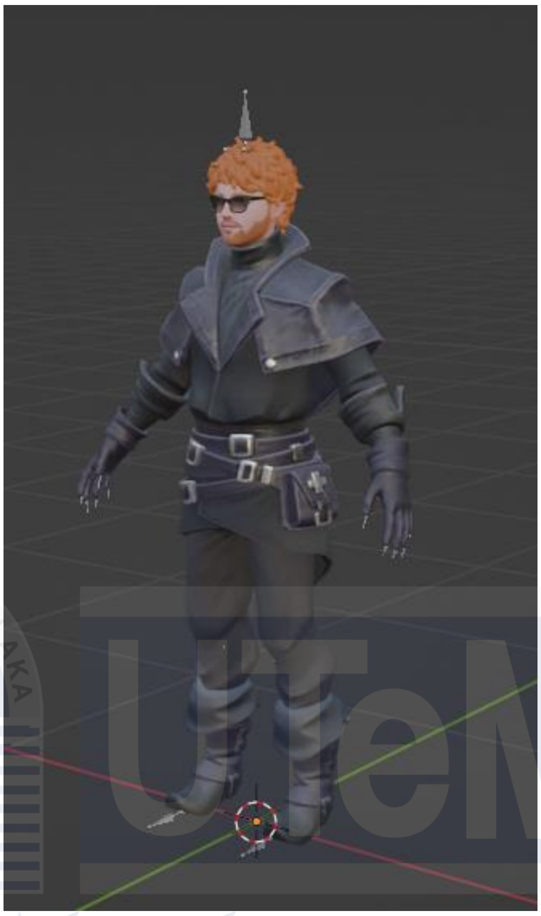


Figure 4.21 Character B

The body parts will then be exported into unity with their armature into Unity in fbx format and into different folders for character customization (as shown in **Figure 4.22**).

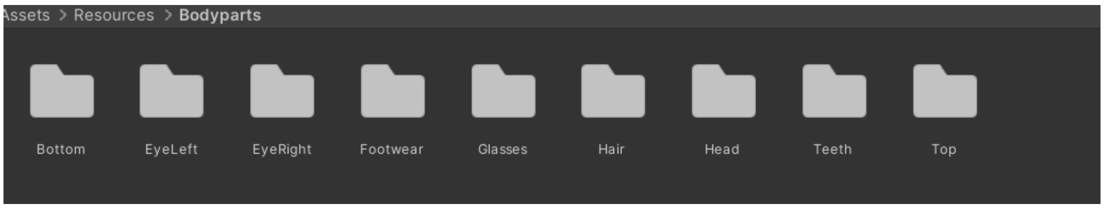
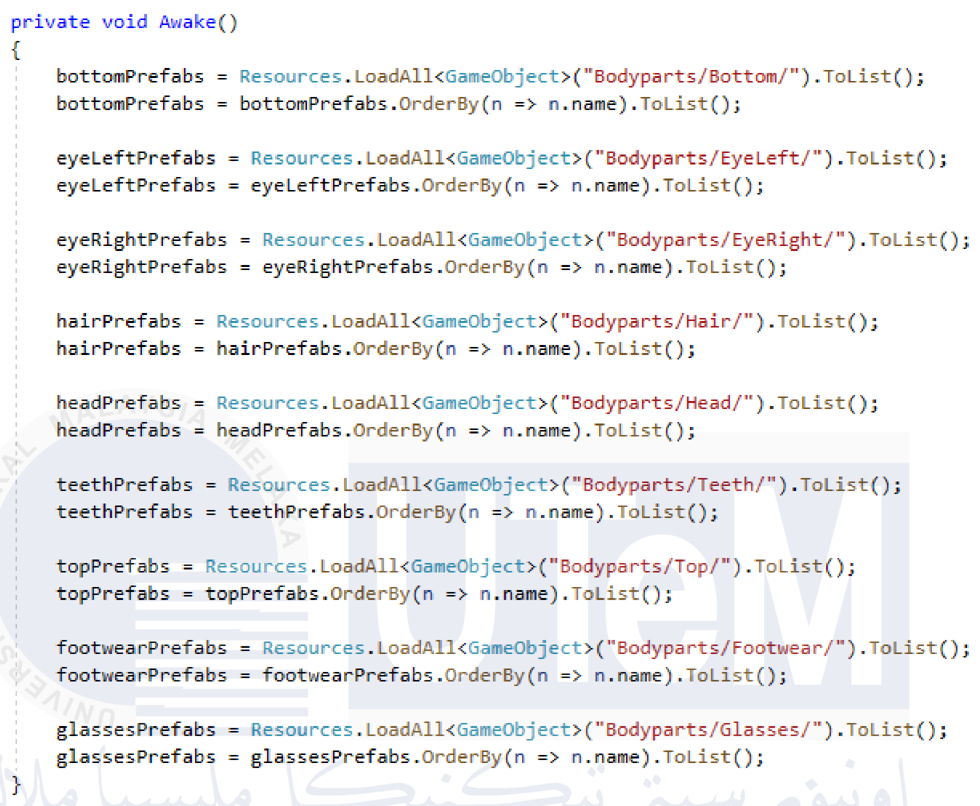


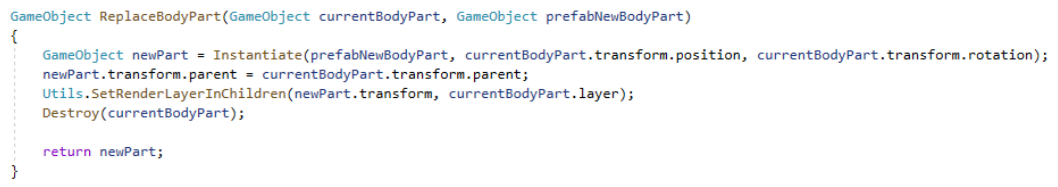
Figure 4.22 Bodypart Folders

Then, the characterOutfitHandler script will handle these body parts so players can customize their characters to the one they like. The prefabs (body parts) will be loaded in the script (as shown in **Figure 4.23**), and then the body parts will be replaced by the new body parts with the same transformation (location, scale, and rotation) while the old body parts will be destroyed (as shown in **Figure 4.24**).



```
private void Awake()
{
    bottomPrefabs = Resources.LoadAll<GameObject>("Bodyparts/Bottom/").ToList();
    bottomPrefabs = bottomPrefabs.OrderBy(n => n.name).ToList();

    eyeLeftPrefabs = Resources.LoadAll<GameObject>("Bodyparts/EyeLeft/").ToList();
    eyeLeftPrefabs = eyeLeftPrefabs.OrderBy(n => n.name).ToList();

    eyeRightPrefabs = Resources.LoadAll<GameObject>("Bodyparts/EyeRight/").ToList();
    eyeRightPrefabs = eyeRightPrefabs.OrderBy(n => n.name).ToList();

    hairPrefabs = Resources.LoadAll<GameObject>("Bodyparts/Hair/").ToList();
    hairPrefabs = hairPrefabs.OrderBy(n => n.name).ToList();

    headPrefabs = Resources.LoadAll<GameObject>("Bodyparts/Head/").ToList();
    headPrefabs = headPrefabs.OrderBy(n => n.name).ToList();

    teethPrefabs = Resources.LoadAll<GameObject>("Bodyparts/Teeth/").ToList();
    teethPrefabs = teethPrefabs.OrderBy(n => n.name).ToList();

    topPrefabs = Resources.LoadAll<GameObject>("Bodyparts/Top/").ToList();
    topPrefabs = topPrefabs.OrderBy(n => n.name).ToList();

    footwearPrefabs = Resources.LoadAll<GameObject>("Bodyparts/Footwear/").ToList();
    footwearPrefabs = footwearPrefabs.OrderBy(n => n.name).ToList();

    glassesPrefabs = Resources.LoadAll<GameObject>("Bodyparts/Glasses/").ToList();
    glassesPrefabs = glassesPrefabs.OrderBy(n => n.name).ToList();
}
```

Figure 4.23 Load all bodypart prefabs to list



```
GameObject ReplaceBodyPart(GameObject currentBodyPart, GameObject prefabNewBodyPart)
{
    GameObject newPart = Instantiate(prefabNewBodyPart, currentBodyPart.transform.position, currentBodyPart.transform.rotation);
    newPart.transform.parent = currentBodyPart.transform.parent;
    Utils.SetRenderLayerInChildren(newPart.transform, currentBodyPart.layer);
    Destroy(currentBodyPart);

    return newPart;
}
```

Figure 4.24 Replacing bodyparts

The players will play their game in one of the two environments: the factory and the jungle. The factory environment is built from assets taken from the Unity store and arranged on the map so the players can strategically plan their ambush and hiding places (as shown in **Figure 4.25**). The assets include barrels, boxes, buildings, containers, dumpsters, fences, oil tanks, roads, and other props.

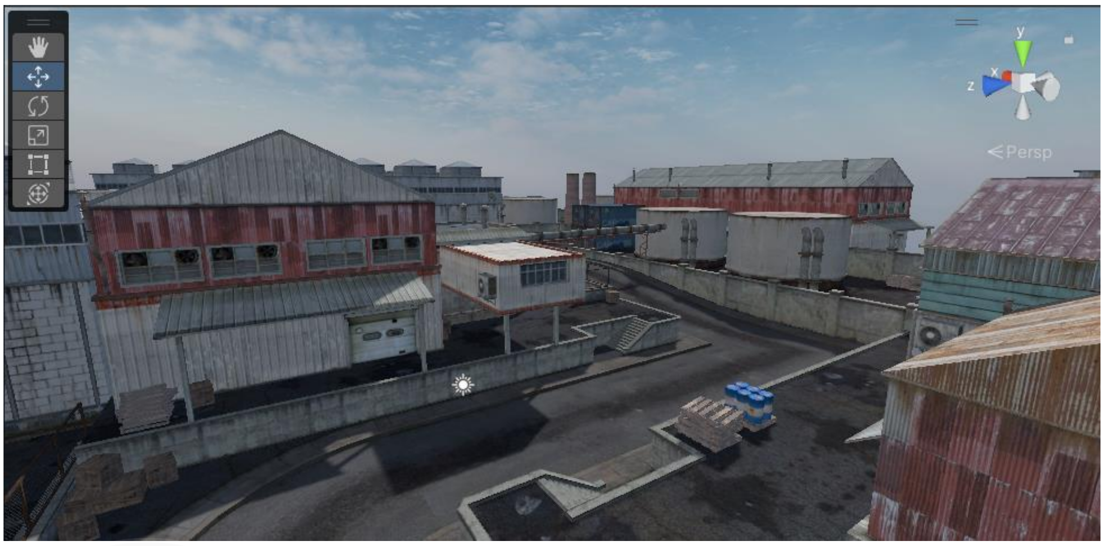
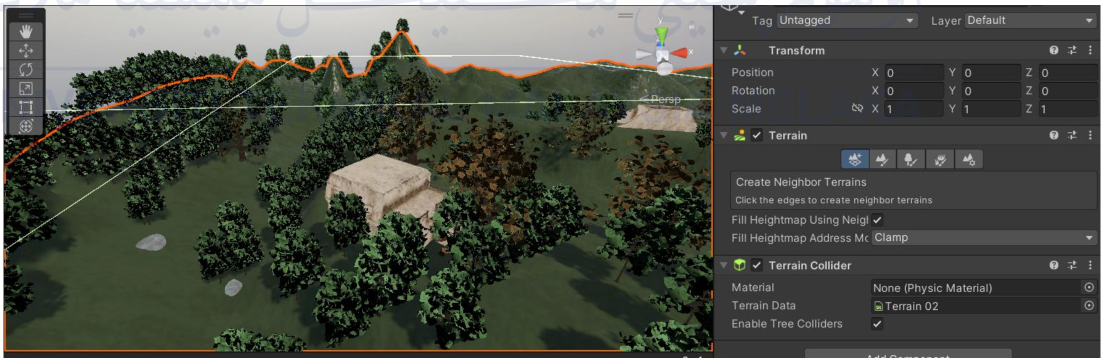


Figure 4.25 Factory map

For the jungle environment, the terrain feature in Unity adds a landscape to the jungle map, simulating what a real jungle looks like. The terrain features help to adjust the height and appearance of the landscape and add trees and grass to it (as shown in **Figure 4.26**).



Tag Untagged Layer Default

Transform

Position	X	0	Y	0	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Terrain ✓

Create Neighbor Terrains
Click the edges to create neighbor terrains

Fill Heightmap Using Neigl	✓
Fill Heightmap Address Mc	Clamp

Terrain Collider ✓

Material	None (Physic Material)
Terrain Data	Terrain 02
Enable Tree Colliders	✓

Figure 4.26 Forest map

4.3.2 Production of Audio/Sound effect

The audio and sound effects are taken from the Pixabay website. The first sound effect implemented into the game is the footstep because footsteps are essential in FPS games (as shown in **Figure 4.27**), as footstep sounds are crucial for situational awareness in FPS games. Players rely on these sounds to detect the presence and movement of enemies. By listening to the direction and proximity of footsteps, players can anticipate enemy positions, plan ambushes, or avoid potential threats.. Also, landing sound is important in FPS games because players will know there are enemies nearby when they hear a loud thump sound (as shown in **Figure 4.28**).

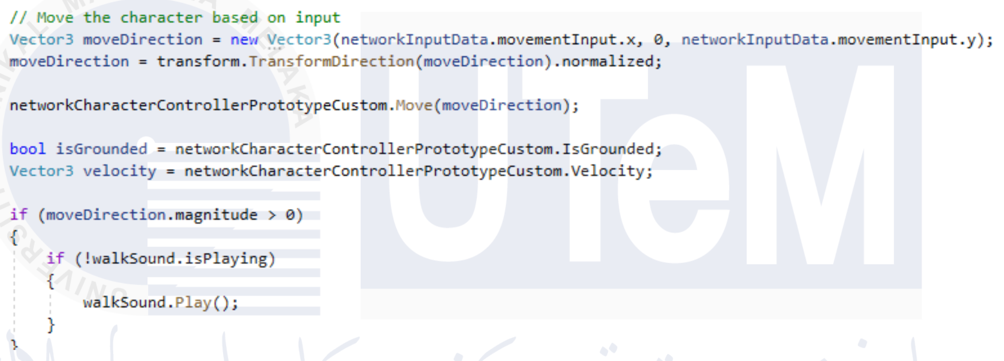

```
// Move the character based on input
Vector3 moveDirection = new Vector3(networkInputData.movementInput.x, 0, networkInputData.movementInput.y);
moveDirection = transform.TransformDirection(moveDirection).normalized;

networkCharacterControllerPrototypeCustom.Move(moveDirection);

bool isGrounded = networkCharacterControllerPrototypeCustom.IsGrounded;
Vector3 velocity = networkCharacterControllerPrototypeCustom.Velocity;

if (moveDirection.magnitude > 0)
{
    if (!walkSound.isPlaying)
    {
        walkSound.Play();
    }
}
```

Figure 4.27 Calling walking sound

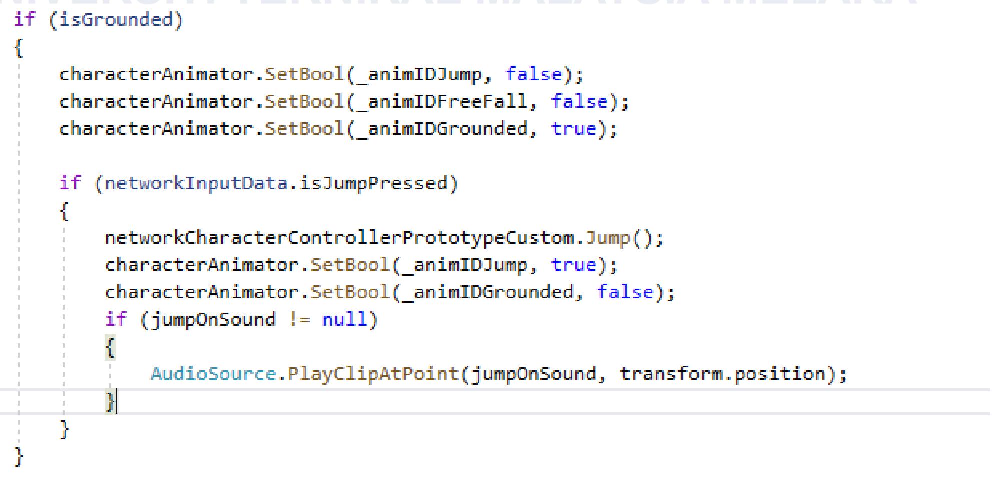
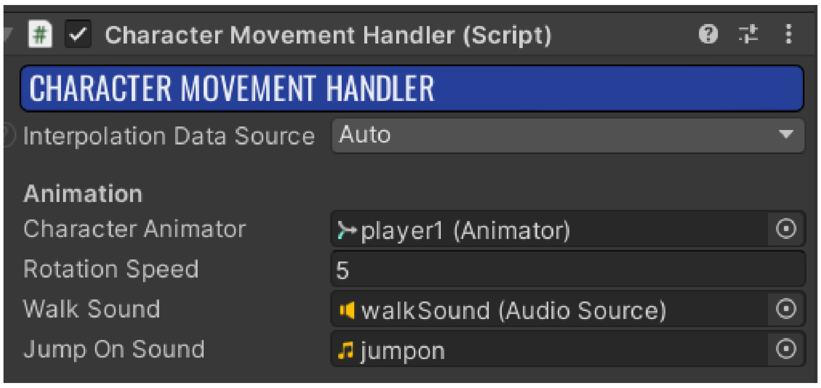

```
if (isGrounded)
{
    characterAnimator.SetBool(_animIDJump, false);
    characterAnimator.SetBool(_animIDFreeFall, false);
    characterAnimator.SetBool(_animIDGrounded, true);

    if (networkInputData.isJumpPressed)
    {
        networkCharacterControllerPrototypeCustom.Jump();
        characterAnimator.SetBool(_animIDJump, true);
        characterAnimator.SetBool(_animIDGrounded, false);
        if (jumpOnSound != null)
        {
            AudioSource.PlayClipAtPoint(jumpOnSound, transform.position);
        }
    }
}
```

Figure 4.28 Calling jumping sound



CHARACTER MOVEMENT HANDLER

Character Movement Handler (Script)

Interpolation Data Source: Auto

Animation

Character Animator: player1 (Animator)

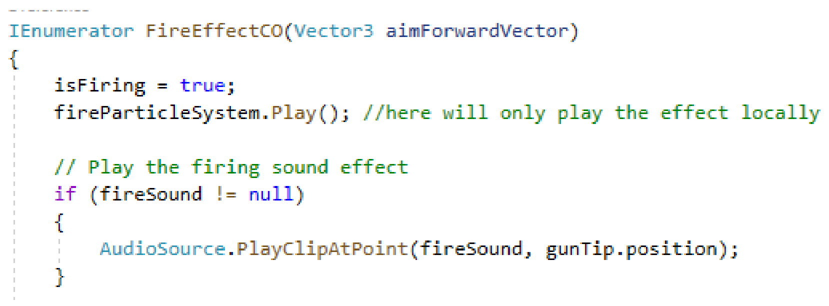
Rotation Speed: 5

Walk Sound: walkSound (Audio Source)

Jump On Sound: jumpon

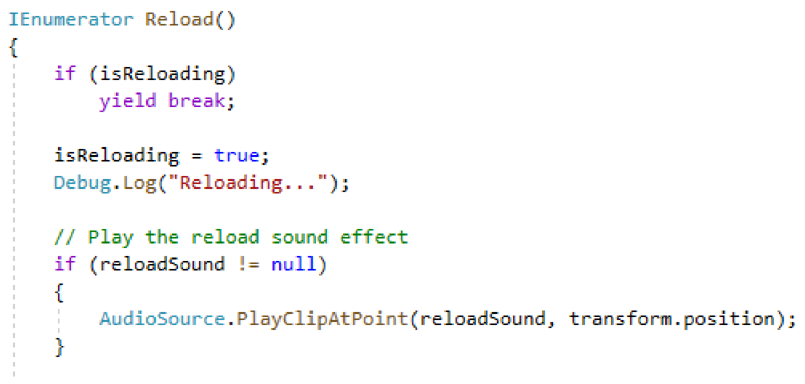
Figure 4.29 Assigning walk and jump sound

Gun shooting sounds play an important role in FPS games because it lets players know where the battle is happening. Gun shooting sounds significantly contribute to situational awareness in FPS games. Players rely on these sounds to understand the location and intensity of ongoing battles, helping them make strategic decisions, such as moving toward the action or avoiding danger. The remote gun shooting sound ensures that the shooting sound only takes place on who shoots the gun (as shown in **Figure 4.30**). Reload sounds are an auditory confirmation that the player has initiated and completed the reloading process (as shown in **Figure 4.31**). This feedback is crucial, especially during intense combat situations where visual cues might be less noticeable. Knowing that their weapon is reloaded allows players to plan their next actions without second-guessing.



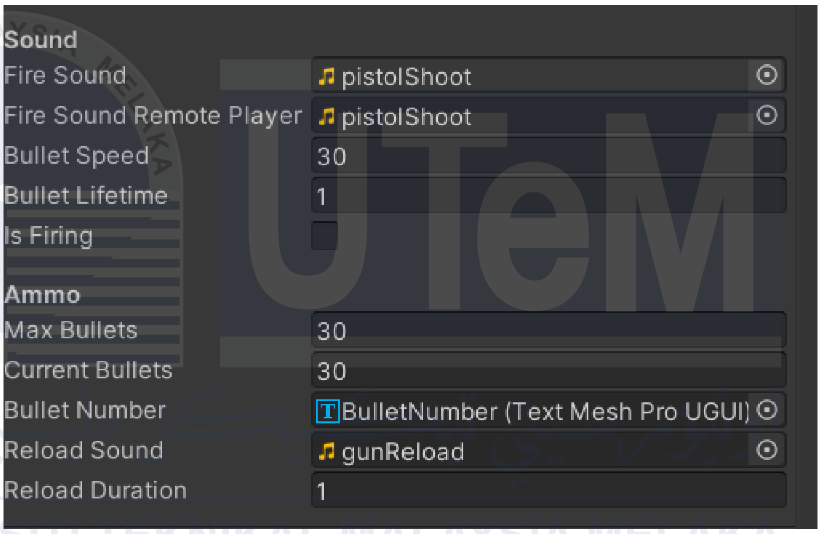
```
IEnumerator FireEffectCO(Vector3 aimForwardVector)
{
    isFiring = true;
    fireParticleSystem.Play(); //here will only play the effect locally

    // Play the firing sound effect
    if (fireSound != null)
    {
        AudioSource.PlayClipAtPoint(fireSound, gunTip.position);
    }
```

Figure 4.30 Calling shooting sound



```
IEnumerator Reload()
{
    if (isReloading)
        yield break;

    isReloading = true;
    Debug.Log("Reloading...");

    // Play the reload sound effect
    if (reloadSound != null)
    {
        AudioSource.PlayClipAtPoint(reloadSound, transform.position);
    }
```

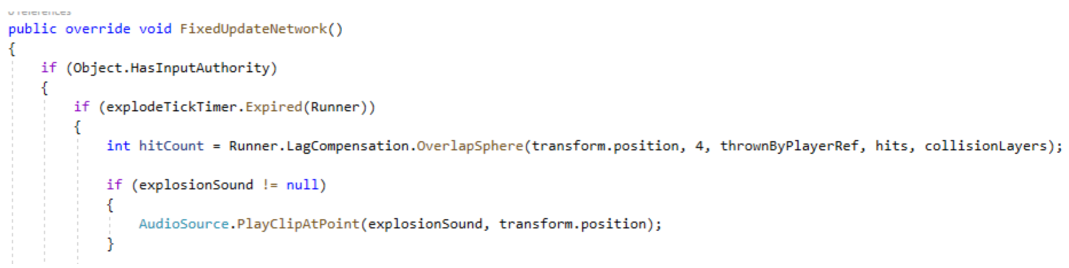
Figure 4.31 Calling reload sound



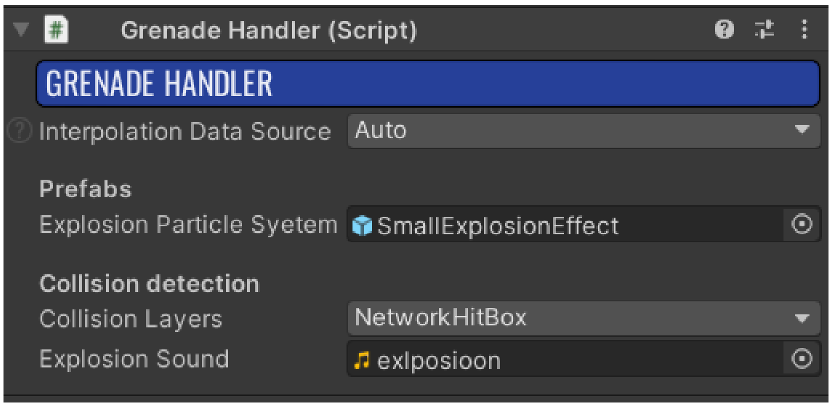
Sound	
Fire Sound	pistolShoot
Fire Sound Remote Player	pistolShoot
Bullet Speed	30
Bullet Lifetime	1
Is Firing	☐
Ammo	
Max Bullets	30
Current Bullets	30
Bullet Number	BulletNumber (Text Mesh Pro UGUI)
Reload Sound	gunReload
Reload Duration	1

Figure 4.32 Assigning shoot and reload sound

Grenade explosion sounds are essential in FPS games, significantly enhancing situational awareness, tactical decision-making, immersion, emotional impact, and gameplay feedback. Incorporating high-quality explosion sounds can create a more realistic and engaging gaming experience.



```
public override void FixedUpdateNetwork()
{
    if (Object.HasInputAuthority)
    {
        if (explodeTickTimer.Expired(Runner))
        {
            int hitCount = Runner.LagCompensation.OverlapSphere(transform.position, 4, thrownByPlayerRef, hits, collisionLayers);

            if (explosionSound != null)
            {
                AudioSource.PlayClipAtPoint(explosionSound, transform.position);
            }
```


Grenade Handler (Script)

GRENADE HANDLER

Interpolation Data Source: Auto

Prefabs

Explosion Particle Syetem: SmallExplosionEffect

Collision detection

Collision Layers: NetworkHitBox

Explosion Sound: exlposioon

4.3.3 Production of User Interface

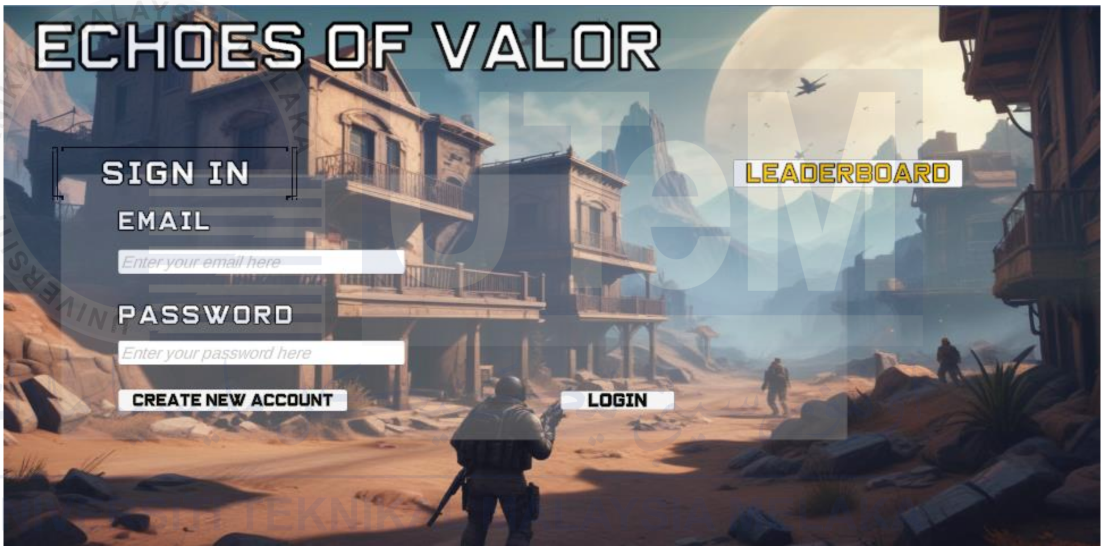


Figure 4.33 Login Page

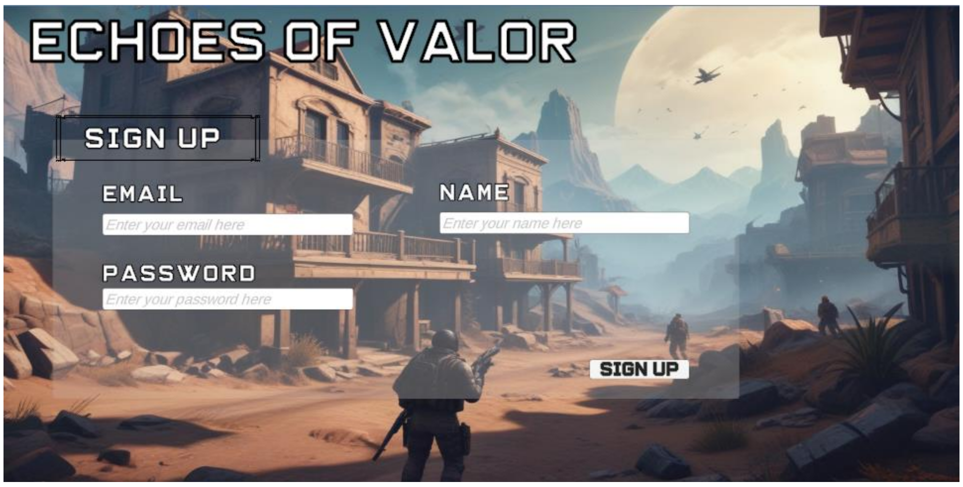
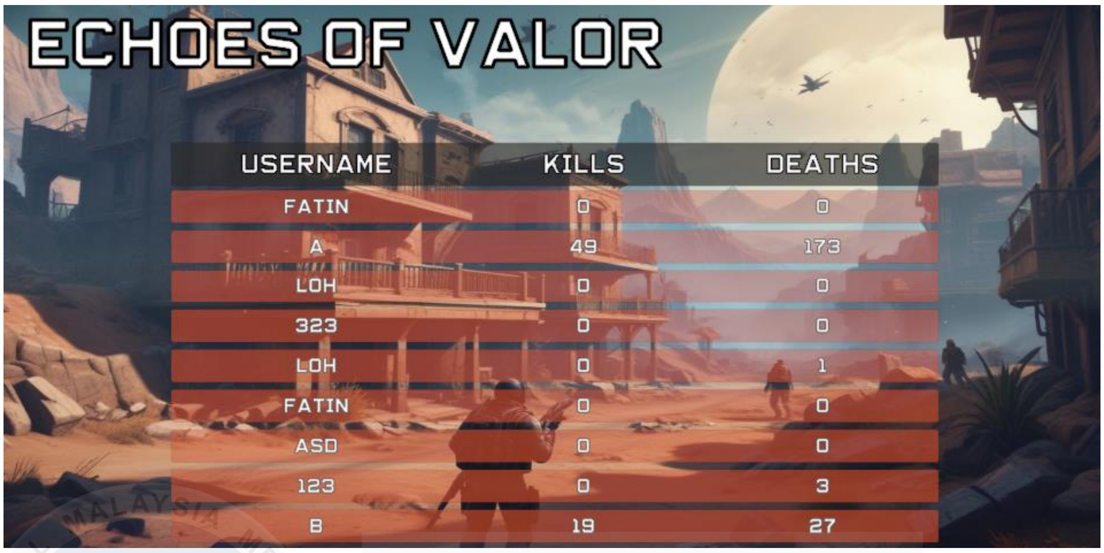


Figure 4.34 Register Page



USERNAME	KILLS	DEATHS
FATIN	0	0
A	49	173
LOH	0	0
323	0	0
LOH	0	1
FATIN	0	0
ASD	0	0
123	0	3
B	19	27

Figure 4.35 Leaderboard Page

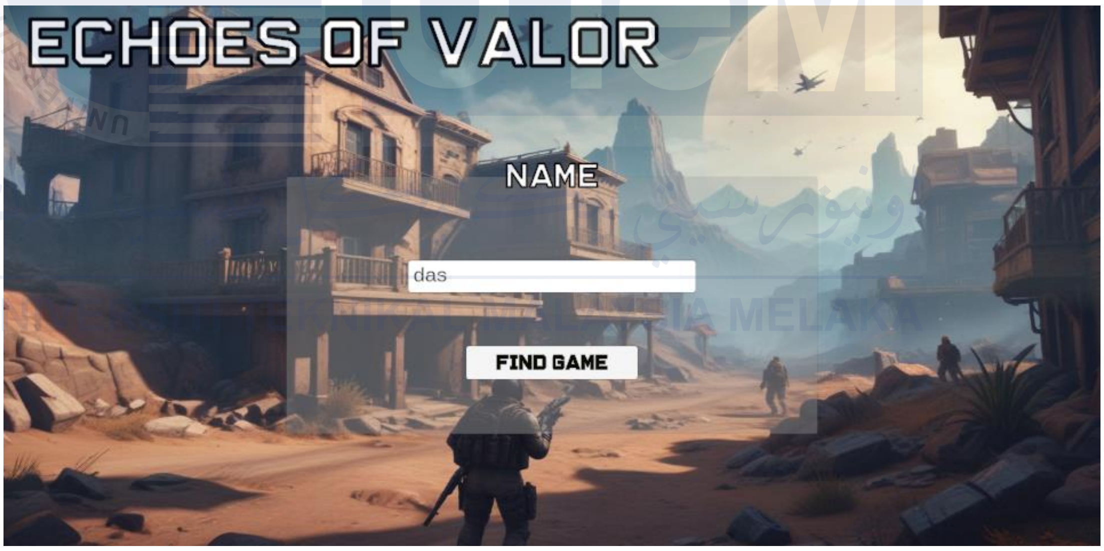


Figure 4.36 Enter in-game name page

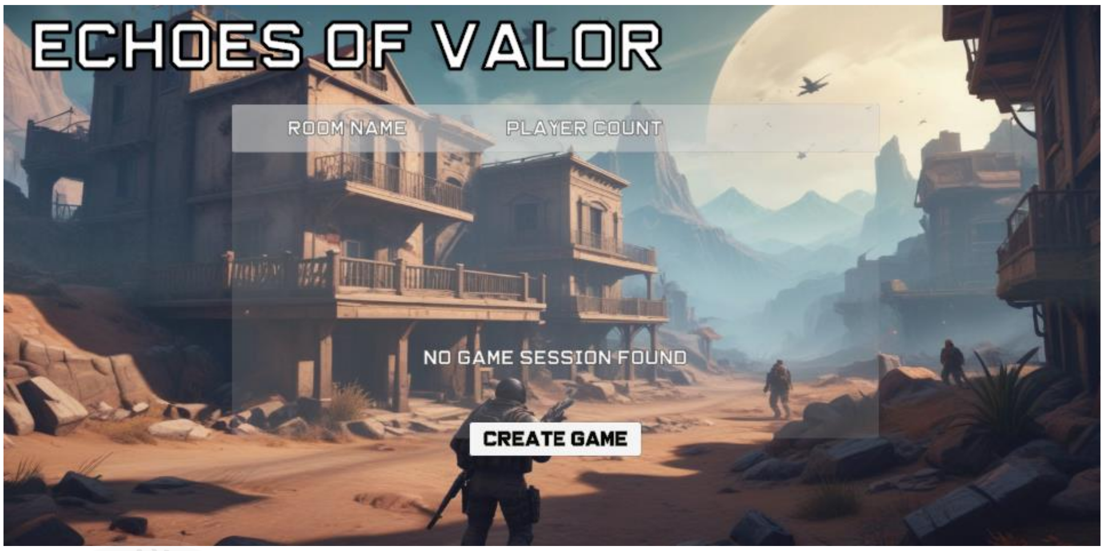


Figure 4.37 Room session page when there are no rooms

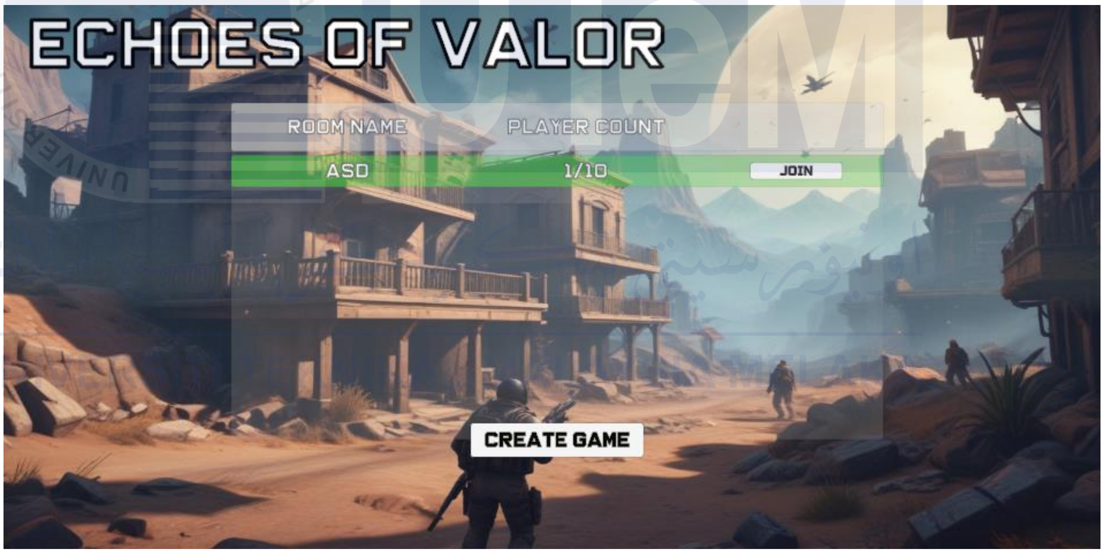


Figure 4.38 Room session page when there is room available

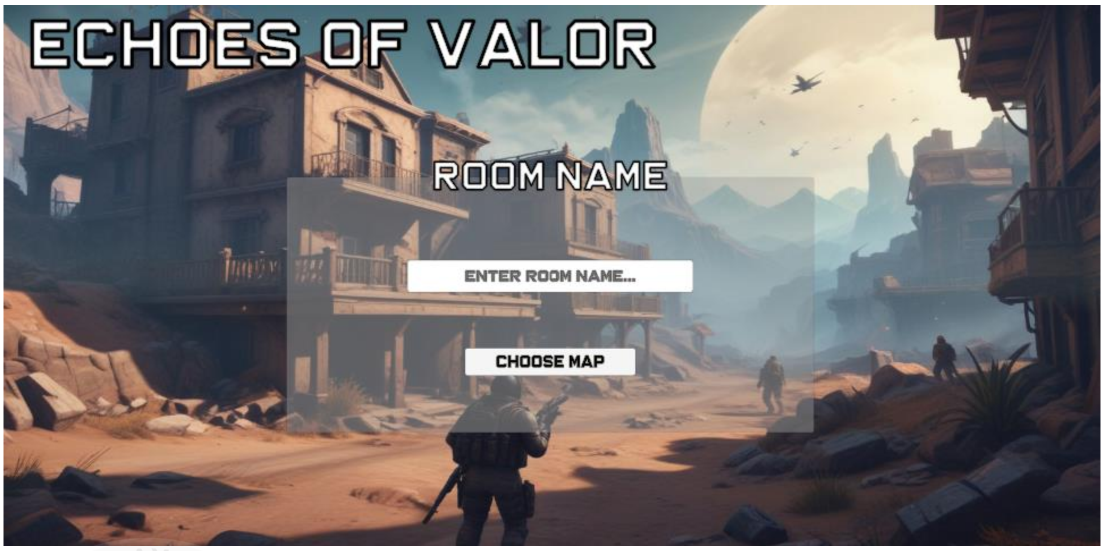


Figure 4.39 Room naming page

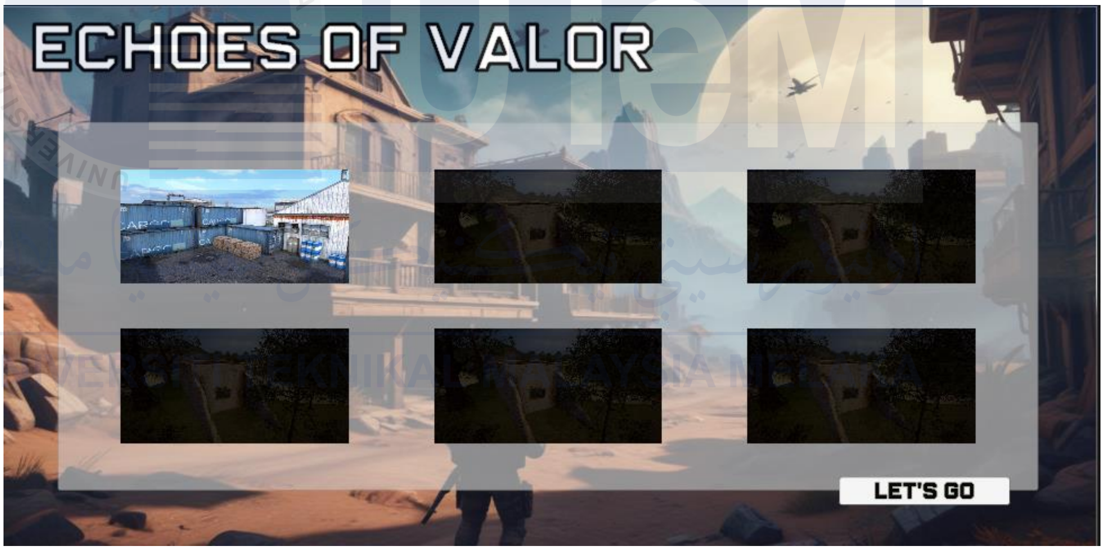


Figure 4.40 Map selection page

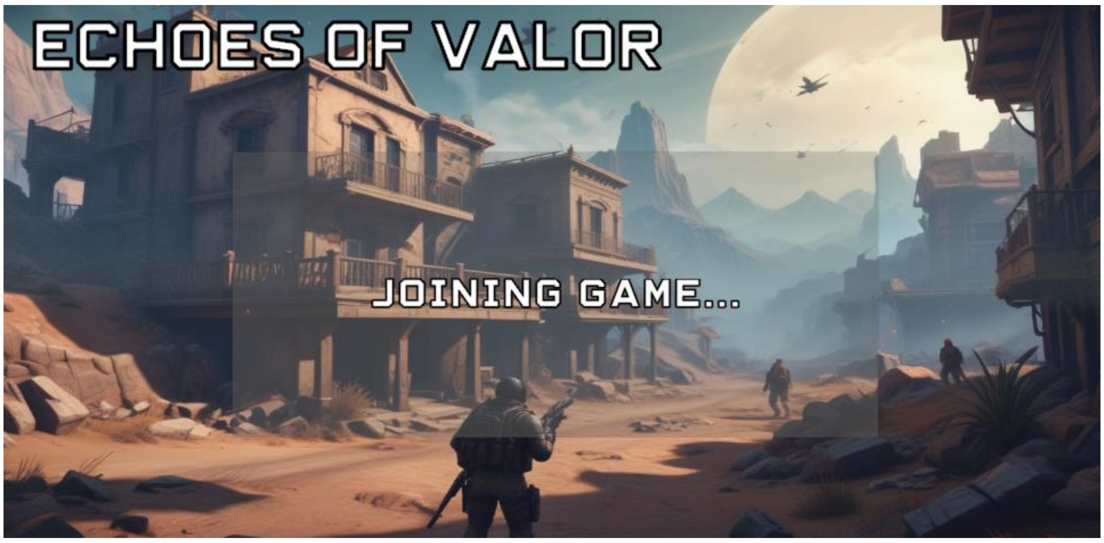


Figure 4.41 Joining room page

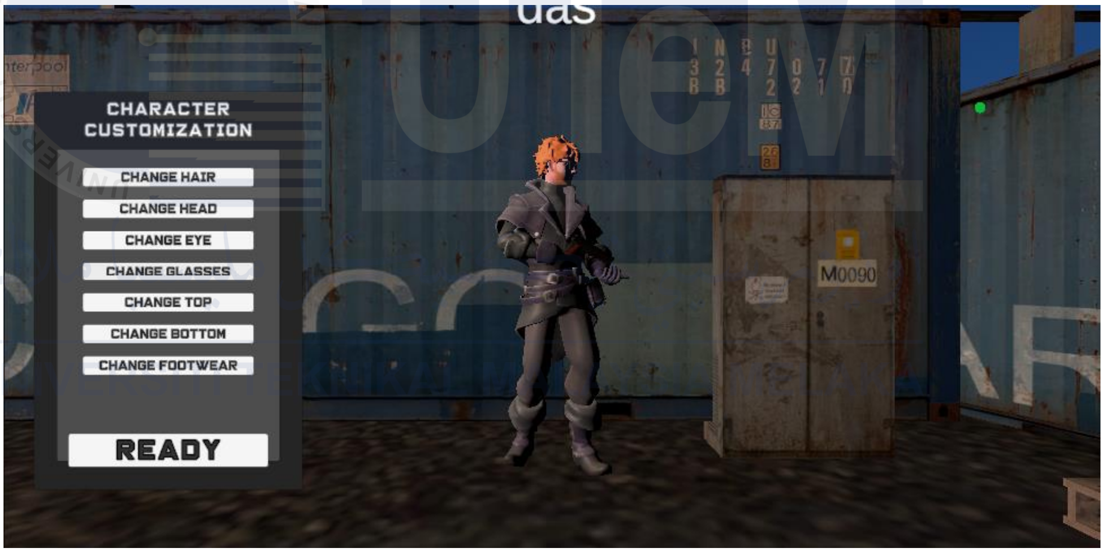


Figure 4.42 Character customization page

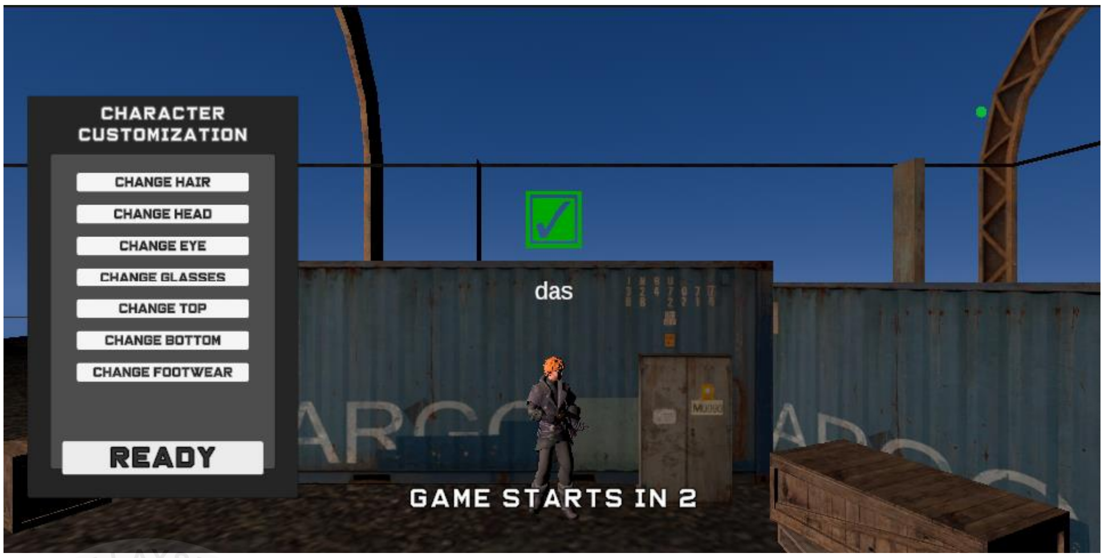


Figure 4.43 Players ready interface

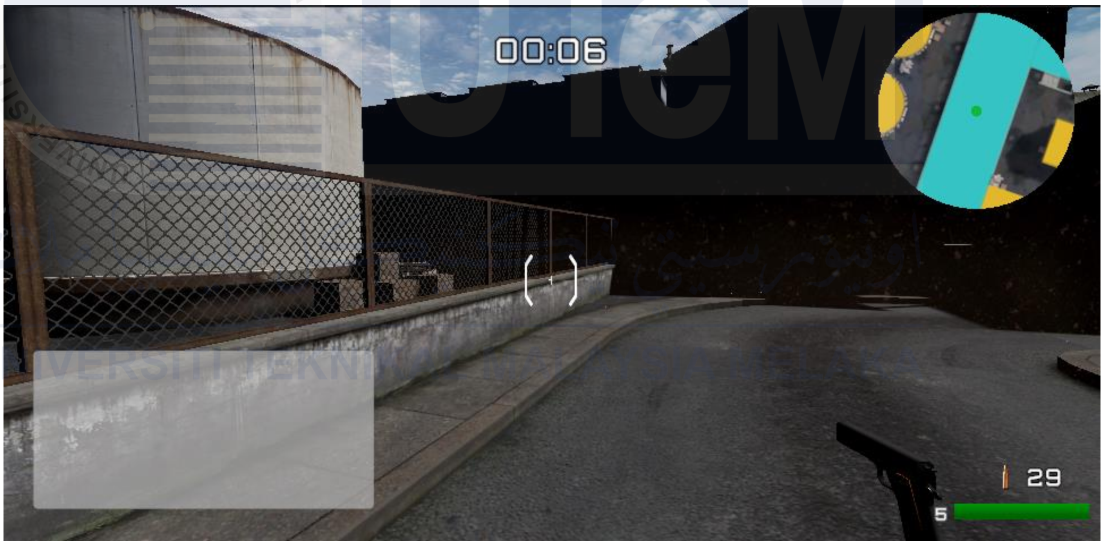


Figure 4.44 Players in-game interface

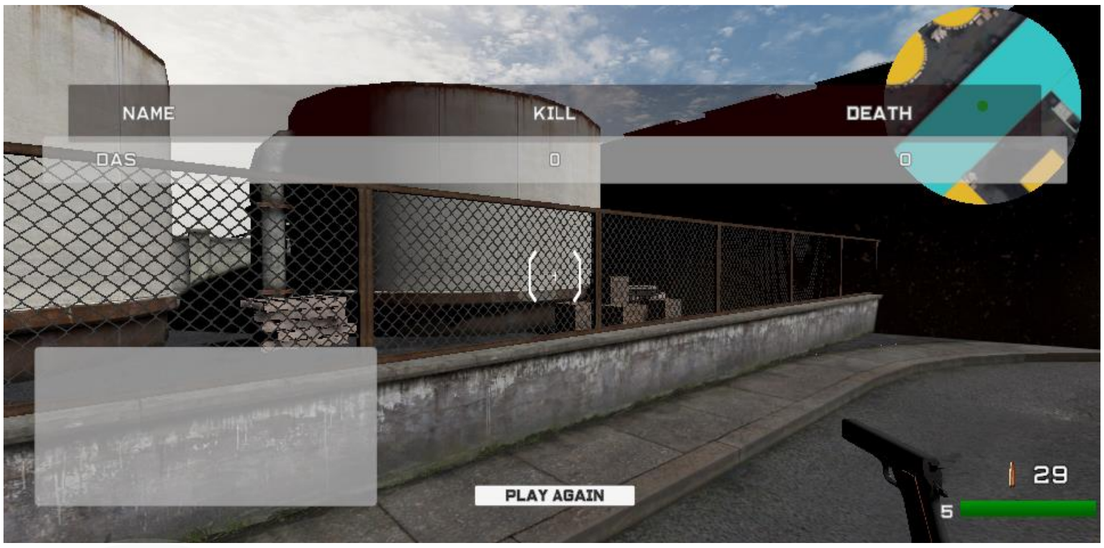


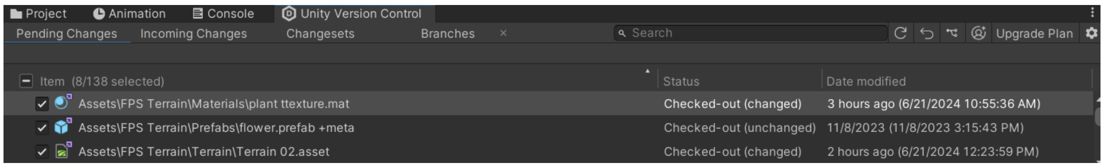
Figure 4.45 End game interface

4.4 Game Configuration Management

4.4.1 Version Control Procedure

This project uses Unity Version Control to track its progress. Changes in the workspace can be committed to the repository for backup to prevent data loss. The Unity Version Control is downloaded in the Unity Hub and will initialize a new repository for the current project.

First, a regular workspace needs to be set up in the project. It is a personal copy of the repository where changes can be made independently. The workspace is regularly synchronized with the repository to keep the local copy up to date with any changes. After making changes, they can be committed to the local workspace. Each commit is a snapshot of the project at a specific point in time, allowing progress to be tracked and previous states reverted if needed (as shown in **Figure 4.46**).



Item (8/138 selected)	Status	Date modified
✓ Assets\FPS Terrain\Materials\plant ttexture.mat	Checked-out (changed)	3 hours ago (6/21/2024 10:55:36 AM)
✓ Assets\FPS Terrain\Prefabs\flower.prefab +meta	Checked-out (unchanged)	11/8/2023 (11/8/2023 3:15:43 PM)
✓ Assets\FPS Terrain\Terrain\Terrain 02.asset	Checked-out (changed)	2 hours ago (6/21/2024 12:23:59 PM)

Figure 4.46 Unity Version Control

The advantage of using Unity Version Control is that it seamlessly integrates with the Unity Editor, simplifying the project's version control process. It also keeps your development organized, allowing you to track changes, experiment with new features, and ensure your project remains stable.

4.5 Implementation Status

The implementation status section provides an overview of the project's progress in various aspects. It outlines what has been completed, what is in progress, and what remains to be done. This section helps track the project's development and ensures that all components progress as planned.

The user authentication module has been implemented using a secure protocol. Basic functionalities such as user registration and login are operational. The real-time database has been set up using Firebase. Data structures for usernames, kill scores, and death scores are in place and are being updated regularly. The host-migration feature is designed to allow seamless transfer of the game state if the host disconnects.

Body parts of 3D models are still in progress to enhance visual appeal and realism and provide more choices to players. Audio assets still have room for improvement, refinement, and enhancement to match the game’s tone and atmosphere. Design and implementation of the user interface, including main menus and in-game HUD, are completed. A version control system (Unity Version Control) was integrated into the development workflow. Regular commits and backups are being performed.

4.6 Conclusion

In conclusion, the project's development is on schedule, all essential landmarks have been defined and, to a large extent, fulfilled, as well as the work in different important spheres is ongoing. Utilizing a user authentication module makes it possible to grant only the players the right of access, and the real-time setting of the database helps track and update some critical items for playing the game, including usernames, kill scores, and death scores.

The host-migration feature’s design serves as a continuous game playing from the user end even though the host is disconnected from the server. At the same time, work is underway on the 3D models and audio items, where further refinement of the looks, sense of detail, and the mood of the levels is the main goal of making the game as enjoyable for the player as possible.

The finalization of the user interface, which comprises principal menus and the heads-up display for gaming, makes for a responsive and smooth stage for player engagement. One of the improvements to the current project management is chalking down the changes at regular intervals. With the use of Unity Version Control, the process has been made much easier and more efficient.

CHAPTER 5: DATA ANALYSIS AND DISCUSSION

5.1 Introduction

This chapter discusses the analysis results gathered from user feedback based on the instruments used which are the Google Forms distributed to the users. The first analysis conducted in this study is to determine how familiar the users are with multiplayer FPS games in the pre-test questionairre. This analysis is undertaken before the users start to test the prototype. The rest of the data will be gathered after the users evaluate the completely developed prototype. The data gathered is related to the prototype's performance and usability. Results are compiled and analyzed in tables, charts, and figures and will be further discussed in this chapter.

5.2 Testing Setup and Procedure

The testing of the prototype took place in the same room to ensure both users are using the same Wifi router so both users were experiencing the same network bandwidth. The laptop and the PC have the exact hardware specifications to ensure consistent output. Figure 5.1 illustrates the experimental setup of testing the prototype in a room with the placement of the laptop and the desktop computer.

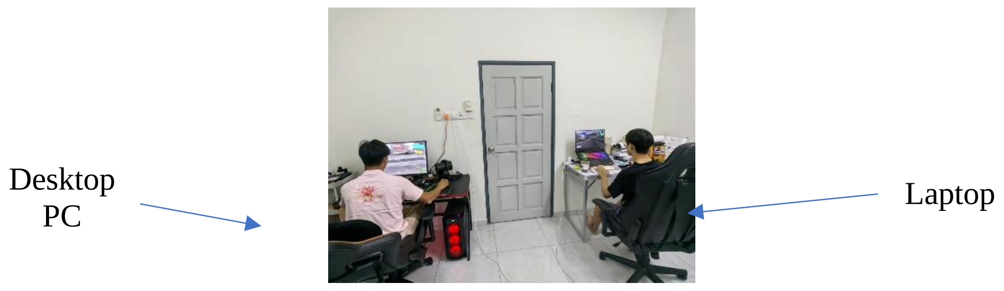


Figure 5.1 The testing setup of multiplayer FPS

5.3 Analysis of data

The result of the analysis of users' familiarity with multiplayer FPS games is taken from the instrument of a questionnaire prepared. The questionnaire is seperated into two main parts. The first part is to determine how familiar users are with multiplayer FPS games, while the second part is whether they can acknowledge the networking issues in multiplayer FPS games. The questionnaire gathers the users' demography data used for the analysis and how frequently the user plays multiplayer FPS games in the first part. The second part of the questionnaire is the users' feedback on the prototype, which includes its usability and performance.

5.3.1 Experience Testing

a. Demography data

The design questionnaire distributed in the study included five questions aimed at collecting demographic data. These questions aimed to collect information about users from different backgrounds and determine how much time they spend on multiplayer FPS games before moving on to the next part of the questionnaire. Table 5.1 shows the demography data of the users who answer the questionnaires based on gender.

Table 5.1 Analysis of demography questionnaire (Gender)

Gender	**Respondents**
Male	13
Female	7
Total	20

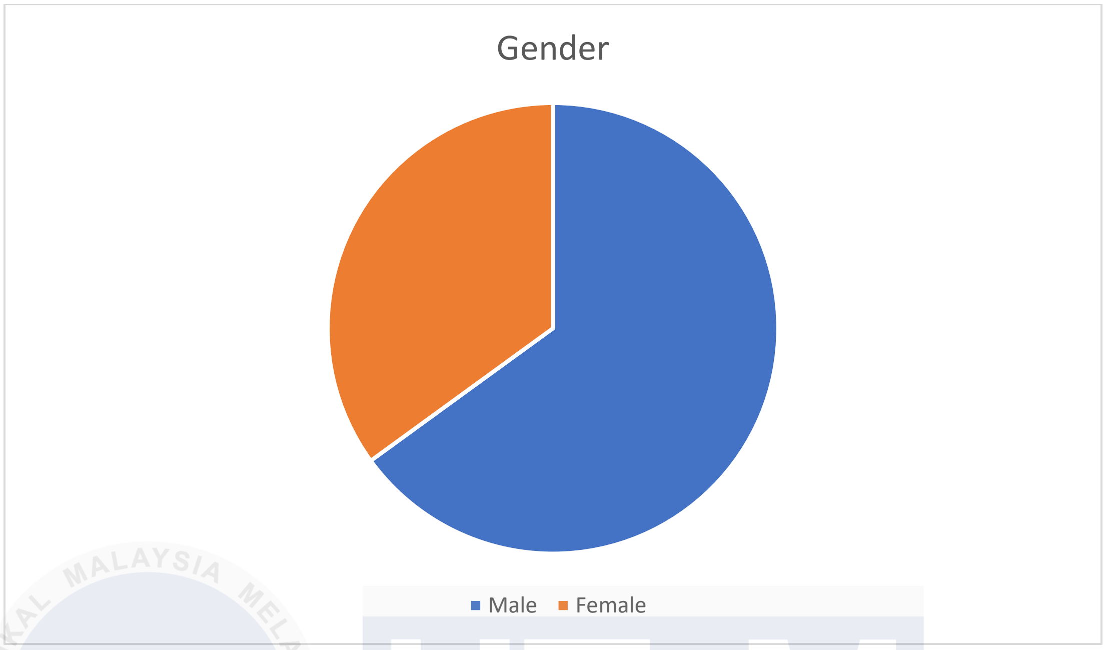


Figure 5.2 Chart of analysis on demography questionnaire (Gender)

According to the data from Table 5.1, the total number of respondents in this study is 20. Most respondents are male, with 13 participants, while the remaining 7 are female. Table 5.2 presents the distribution of respondents by age range.

Table 5.2 Analysis of demography questionnaire (Age)

Age	**Respondents**
18 years old and below	1
19-25 years old	18
26-40 years old	1
41 years old and above	0

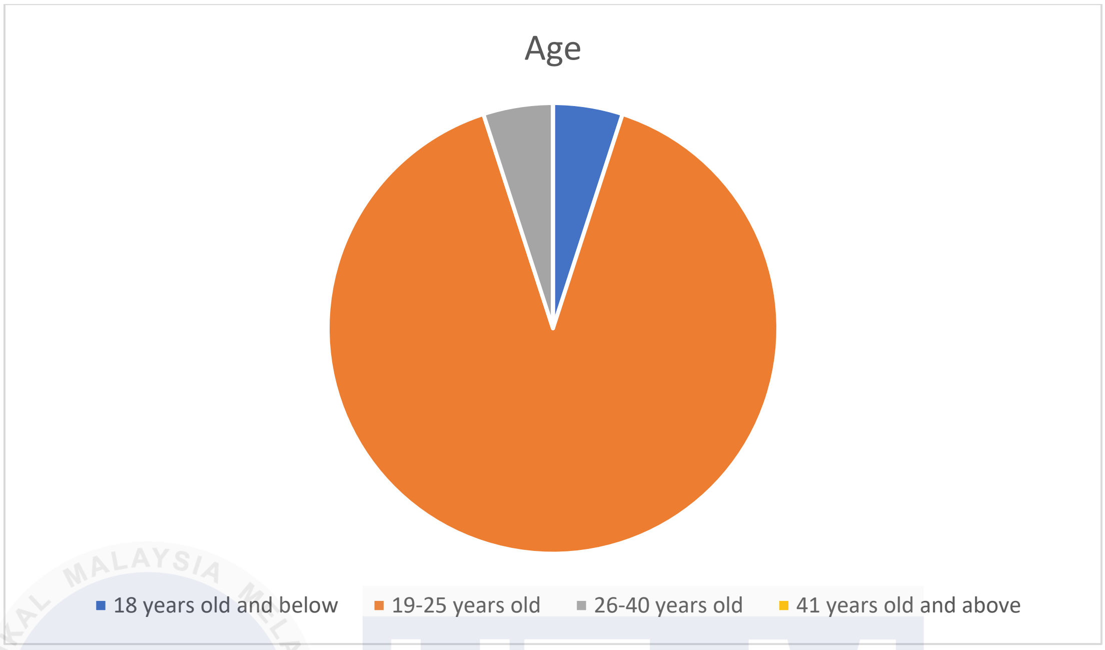


Figure 5.3 Chart of analysis on demography questionnaire (Age)

Table 5.2 shows the respondents' age range. Most of the respondents are 19 to 25 years old, which is 18 people. There is one person who is 18 years old and below and 26-40 years old each. However, there are no respondents who are 41 years old and above. Table 5.3 shows how frequently the respondents play multiplayer FPS games per week.

b. Experience analysis

Experience analysis is where users rate their experience playing multiplayer FPS games. The data analysis for experience analysis has been gathered using a descriptive method: the Likert Scale. The Likert Scale uses a 5-point scale with 1 (Disagree) to 5 (Agree) indicators. The experience analysis questions are listed in Table 5.6 with their aspects. There are 3 questions for experience analysis to determine how familiar users are with multiplayer FPS games.

Table 5.3 Analysis of experience questionnaire (frequency per week)

Frequency per week	**Respondents**
0-1 time	11
2-4 times	4
5-8 times	4
More than 8 times	1

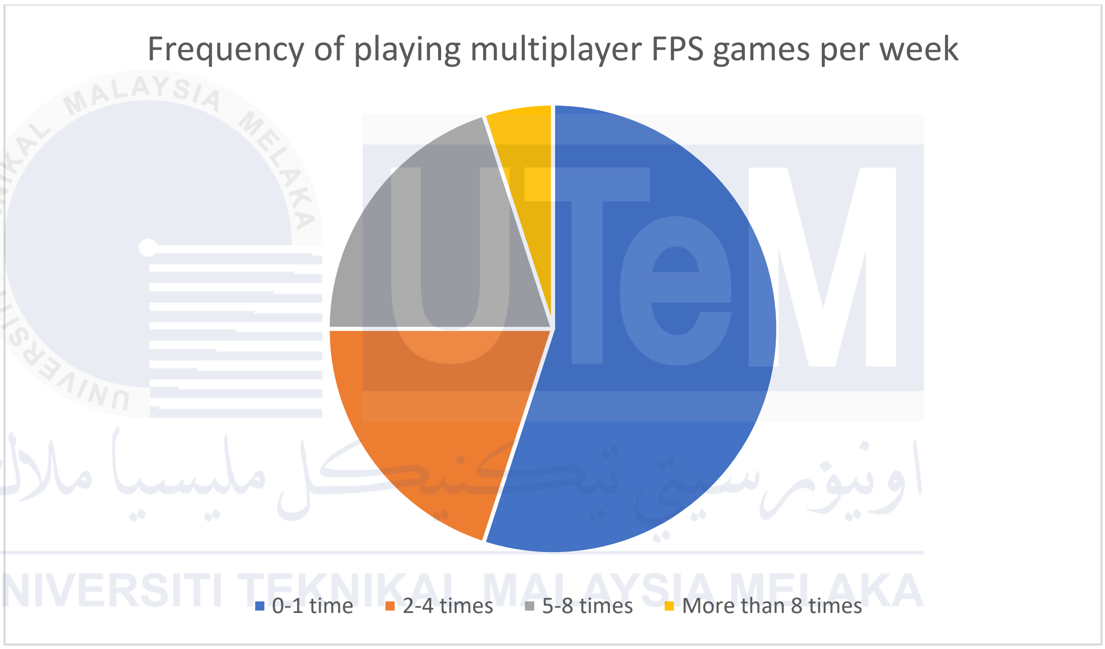


Figure 5.4 Chart of analysis on experience questionnaire (frequency per week)

Based on Table 5.3, 11 respondents only play 0-1 time multiplayer FPS games per week, while only 4 respondents play 2-4 times and 5-8 times multiplayer FPS games per week each. There is one respondent who plays multiplayer FPS games more than 8 times a week. Table 5.4 shows the distribution of respondents based on when they last played multiplayer FPS games.

Table 5.4 Analysis of experience questionnaire (last time play multiplayer FPS game)

When is the last time playing multiplayer FPS game	**Respondents**
Within the last three days	4
Within the last week	5
Within the last month	2
Within the last 3 months	3
None of the above	6

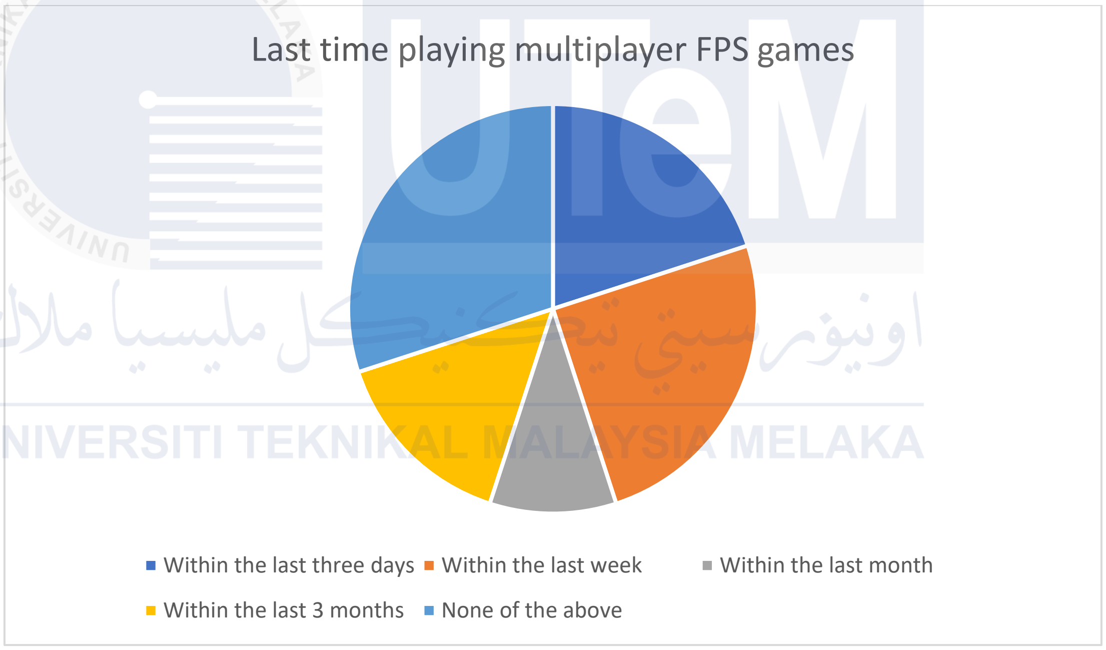


Figure 5.5 Chart of analysis on experience questionnaire (Last time play multiplayer FPS game)

Table 5.4 shows that the last time most respondents played multiplayer FPS games was more than three months, which is six people. Four respondents played multiplayer FPS games within the last three days, while five respondents played multiplayer FPS games within the last week while this study was conducted. However, two respondents played multiplayer FPS games within the last month, and three respondents played the game within the last three months. Table 5.5 shows the respondents' experience with multiplayer FPS games.

Table 5.5 Analysis of experience questionnaire (Rating)

Rating of experience in multiplayer FPS games (from 1 to 5)	**Respondents**
1	1
2	2
3	3
4	8
5	6
Mean	**3.85**

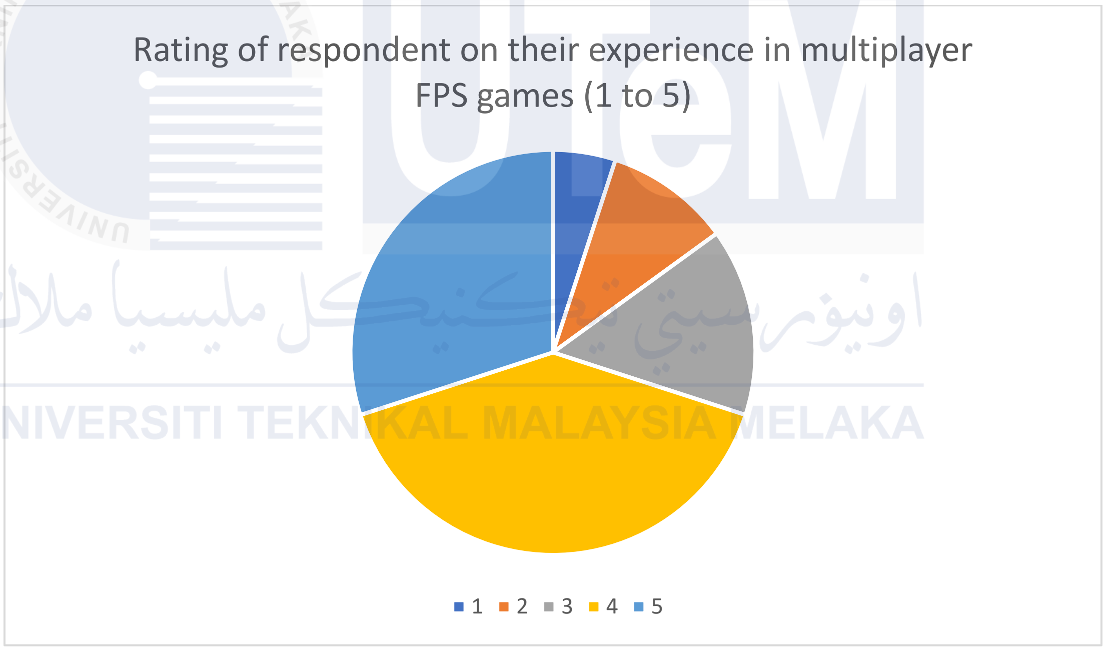


Figure 5.6 Chart of analysis on experience questionnaire (Rating)

Table 5.5 displays the respondents' ratings on their experience of playing multiplayer FPS games. The mean rating is 3.85, which suggests that most users acknowledge their experience with multiplayer FPS games.

5.3.2 Usability Testing

Usability testing consists of post-test questionnaires in which users give feedback after testing the developed prototype. The data analysis for usability testing on the developed prototype was conducted using a descriptive method, specifically employing the Likert Scale. The Likert Scale uses a 5-point scale with 1 (Disagree) to 5 (Agree) indicators. The usability questions are listed in Table 5.6 with their aspects. There are 6 questions for usability, each with different aspects, such as preference and functionality.

Table 5.6 Usability questionnaires

No	**Statements**	**Aspects**
1	I enjoy playing the Echoes of Valor	Preference
2	The multiplayer experience is smooth and enjoyable	Preference
3	The network performance meets my expectations for a competitive FPS game	Preference
4	The ping display is very important to you	Functionality
5	You notice lag or delay in multiplayer FPS games very often	Functionality
6	I am satisfied with the overall network performance of the game	Functionality

Table 5.7 will display the mean and standard deviation (SD) values for the questions in the construct of preference expectancy. Three questions will be asked based on preference expectancy.

Table 5.7 Analysis value of usability (Preference)

	Preference Expectancy	**Mean**	**SD**
1	I enjoy playing the Echoes of Valor	3.84	1.34
2	The multiplayer experience is smooth and enjoyable	4.15	0.75
3	The network performance meets my expectations for a competitive FPS game	4.05	0.83

According to the findings analyzed in Table 5.7, all items in this aspect show a mean value ranging from 3.8 to 4.2, indicating that most respondents agree with all the items related to preference expectancy. The highest mean value is 4.15 for item 2, while the lowest mean value is 3.84 for item 1, which is still close to the "agree" level. Table 5.8 presents the second aspect of user usability, which is functionality expectancy.

Table 5.8 Analysis value of usability (Functionality)

	Functionality Expectancy	**Mean**	**SD**
1	The ping display is very important to you	4.15	1.14
2	You notice lag or delay in multiplayer FPS games very often	4.05	1.15
3	I am satisfied with the overall network performance of the game	4.20	0.89

Based on Table 5.8, all items in the functionality expectancy aspect have achieved a mean value of over four. This indicates that all respondents agree with the items in this aspect. The highest mean value is 4.20 for the third item, while the lowest is 4.05 for the second item.

Table 5.9 Analysis value of usability questionnaire (User Testing)

Factor	**Mean**	**SD**
Preference	4.01	0.97
Functionality	4.13	1.06

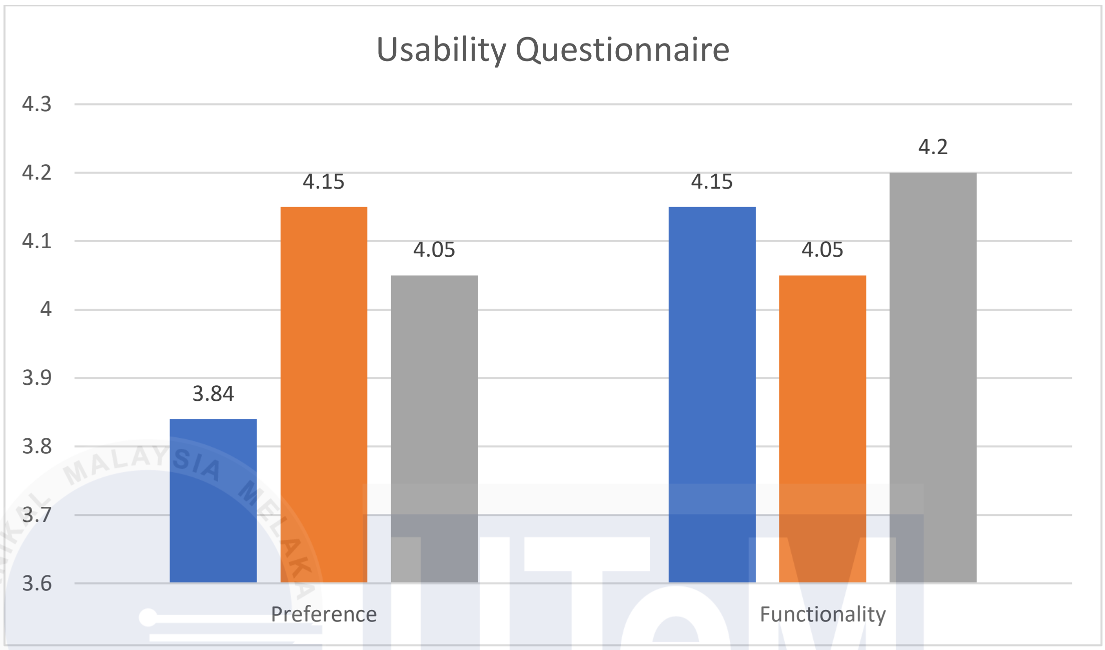


Figure 5.7 Chart of analysis value of usability questionnaire (User Testing)

A s presented in Table 5.9, all aspects of user usability have been analyzed, and the results show that respondents agreed with all aspects, as the mean values are all above 4.0. Based on Figure 5.7, all of the questions related to usability obtained a mean value of more than 4.0 except for the first item in preference expectancy, which means respondents acknowledge the usability of the developed prototype. However, the mean value of 3.84, while slightly below 4.0, is still close to it. Additionally, the overall mean value of the usability questionnaire is 4.07, indicating that respondents generally agreed relatively well with the items in the questionnaire.

5.3.3 Performance Testing

Performance testing consists of post-test questionnaires in which users give feedback after testing the developed prototype which is the same with usability testing. The data analysis of performance testing also uses a 5-point Likert scale with 1 (Disagree) to 5 (Agree) indicators. The performance questions are listed in Table 5.10 with their aspects. There are 9 questions for performance, each with different aspects, such as stability, reliability and efficiency.

Table 5.10 Performance questionnaires

No	Statement	Aspect
1	The game maintains a stable connection without frequent disconnects	Stability
2	I experience minimal lag or jitter during gameplay	Stability
3	The game performs consistently across different network conditions	Stability
4	I experience low ping times during gameplay	Reliability
5	There are minimal disruptions (e.g., freezes, crashes) during gameplay	Reliability
6	The latency does not affect my overall gaming experience	Reliability
7	The game responds quickly to my inputs without noticeable delays	Efficiency
8	Dynamic host migration can reduce network issues in multiplayer games	Efficiency
9	Players becoming the host can improve network performance in-game	Efficiency

Table 5.11 presents the mean and standard deviation values for the questions within the stability expectancy construct. Stability expectancy includes three items for users to evaluate the network stability of the developed prototype.

Table 5.11 Analysis value of performance (Stability)

	Stability Expectancy	**Mean**	**SD**
1	The game maintains a stable connection without frequent disconnects	4.20	0.95
2	I experience minimal lag or jitter during gameplay	4.05	0.94
3	The game performs consistently across different network conditions	4.15	1.04

This aspect consists of three items. The results for all three items indicate that respondents agreed with this aspect, as the mean values for each statement are above 4.0. Specifically, the mean values for the items in Table 5.11 are 4.20, 4.05, and 4.15, respectively. Table 5.12 below presents the mean and standard deviation values for the performance construct of reliability.

Table 5.12 Analysis value of performance (Reliability)

	Reliability Expectancy	**Mean**	**SD**
1	I experience low ping times during gameplay	4.00	1.12
2	There are minimal disruptions (e.g., freezes, crashes) during gameplay	3.90	1.12
3	The latency does not affect my overall gaming experience	4.05	0.89

Through the analysis in Table 5.12, the mean value for all items was 4.0 or higher, except for the second item. The highest mean value was for the third item, at 4.05. The second item had the lowest mean value at 3.90, which is still close to 4.0, indicating that respondents generally agreed with the stability of the developed prototype. Table 5.12 exhibits the findings of efficiency aspect.

Table 5.13 Analysis value of performance (Efficiency)

	Efficiency Expectancy	**Mean**	**SD**
1	The game responds quickly to my inputs without noticeable delays	3.75	1.16
2	Dynamic host migration can reduce network issues in multiplayer games	4.10	0.97
3	Players becoming the host can improve network performance in-game	4.20	0.83

This analysis result, Table 5.13, shows how competent the developed prototype was when the respondents tested it. The first item indicates only a mean value of 3.75, which is the lowest in efficiency. The second and third items have a mean value of 4.10 and 4.20, respectively. Table 5.14 displays the overall mean values for all aspects of the user performance instrument.

Table 5.14 Analysis value of performance questionnaire (User Testing)

Factor	**Mean**	**SD**
Stability	4.13	0.97
Reliability	3.98	1.04
Efficiency	4.02	0.99

Based on the overall mean values presented in Table 5.14, it can be concluded that respondents generally perceive the performance of the developed prototype as the average mean value of the user performance questionnaire is 4.04. The stability factor has the greatest mean value, 4.13, while reliability has the least mean value, 3.98. Efficiency has the second highest mean value, 4.02, slightly greater than reliability.

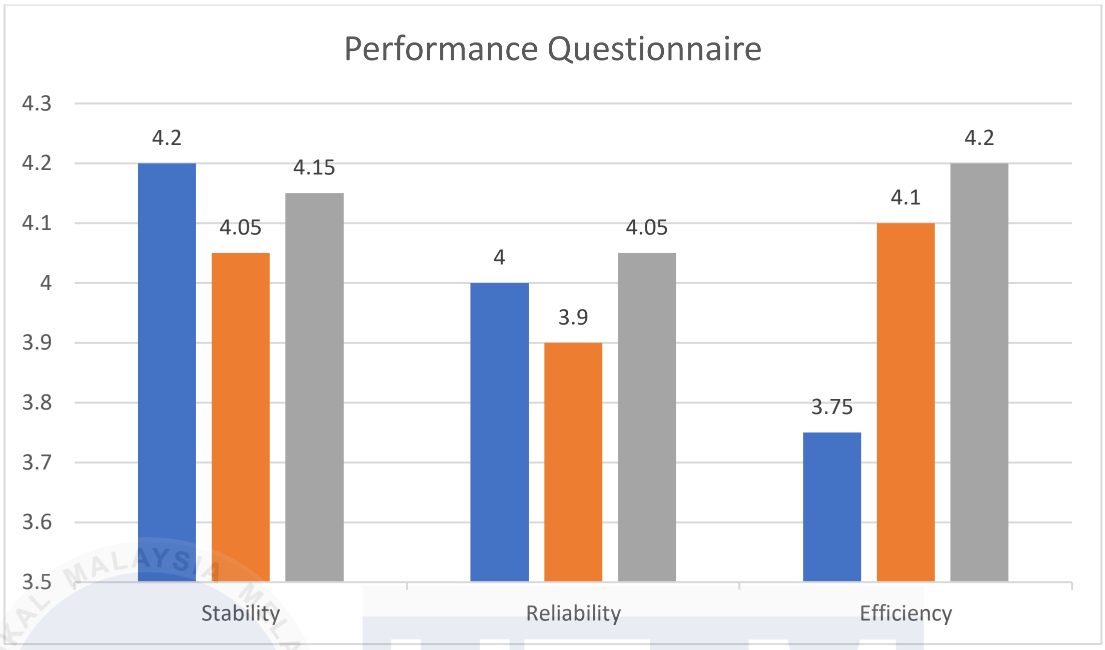


Figure 5.8 Chart of analysis value of performance questionnaire (User Testing)

Based on the graph in Figure 5.8, the finding show that most of the users agree that the performance in networking is stable, reliable and efficient for the developed prototype as most of the items have a mean value more than 4.0. Although there is a mean value of 3.9 in reliability and 3.75 in efficiency, but the overall perception is still positive.

5.3.4 Evaluation on prototype by expert

Two experts further evaluated the prototype of a multiplayer FPS game. The experts are comprised of Agmo Studio seniors who are also from Game Technology at UTeM. The qualitative data were collected from the experts' evaluations. Table 5.15 summarizes the overall comments from the experts on the prototype. Most of the feedback provided by the experts was more critical of the interface design of the prototype. Table 5.15 below presents the remarks collected from the experts.

Table 5.15 Recommendations for Improvement

Expert	Response
E1	The prototype has a good environment design and has a good graphic
E2	More weapon classes should be added to the game to make it more interesting
E1	There is a little bit of lagging noticed with the rocket.
E2	The scoreboard should be optimized to be more user-friendly.
E1	The game is smooth overall.
E2	There is no severe lag experienced while testing the game.

From Table 5.15, the qualitative analysis of the feedback from experts from the industry gave constructive comments on what can be improved for this prototype. Overall, the experts are optimistic that the host migration technique can help to reduce network issues after testing the prototype, although they face small latency issues during the test of the prototype. They also gave constructive feedback on how to improve the aesthetic and interface design of the prototype to make the game more attractive and increase the users' retention.

5.3.5 Discussion

Based on Figure 5.1, the usability analysis of the prototype revealed a general positive feedback from the suers as most of the items obtain a mean score above 4 on a 5-point scale. The high mean score indicates that the prototype is enjoyable and satisfied how the game operates in real-time. However, the enjoyment of playing "Echoes of Valor" received a slightly lower mean score suggest that some areas in the prototype such as gameplay or networking or other factor should be enhanced to obtain higher level of user satisfaction.

Based on Figure 5.2, the performance analysis shows a promising result for the host migration technique implemented in the prototype. The stability aspect in performance has the highest mean score among the three aspects. This indicates that host migration technique had a great impact on reducing the networking issues in multiplayer FPS games.

Overall, the positive feedback on both usability and performance shows that host migration technique can help to reduce network issues in multiplayer FPS games but there are still further refinement needed in the future work.

5.4 Summary

This chapter demonstrates the analysis of the research findings and is followed by a discussion of the findings. In this research, experiments were carried out to collect qualitative data for statistical analysis. The qualitative data were collected based on the analysis of the users' responses, starting from users' familiarity with multiplayer FPS games and the prototype's usability and performance.

The data for all these instruments are collected through questionnaires. The qualitative data collected is essential to further the research and determine the prototype's limitations and the effectiveness of the developed prototype.

CHAPTER 6: CONCLUSION

6.1 Introduction

This chapter provides a comprehensive conclusion to the study conducted, highlighting all the key findings obtained in this study and contributions of the research. The summary consists of the multiplayer FPS game evaluation, the usability, and the performance of the prototype. Additionally, this chapter discusses the implications of the findings across various perspectives, including those of users, experts, game developers, network engineers, and others. These insights provide a comprehensive understanding of how the prototype can be improved and refined based on feedback from different stakeholders. The limitations encountered during the project will also be discussed and the suggestion on the further studies that can be conducted is interpreted at the end of this chapter.

6.2 Conclusion for study

The conclusion for this study is described based on the data analyzed in Chapter 5, which is based on the usability and performance of the prototype. The following conclusions can be drawn from the evaluations provided by users and experts for each research question. Overall, the results and analysis indicate that this multiplayer FPS game demonstrated positive usability and performance for the users.

6.2.1 Usability of prototype

The usability of the prototype was investigated based on two key constructs: preference and functionality. The result can be concluded that the prototype's usability is positive on all of the factors based on the table in Figure 5.7. However, the mean value for the statement “I enjoy playing Echoes of Valor” is lower than 4.0. This result needs to be reconsidered, whether it's because of the network aspect, as it can be the game mechanics or the game aesthetic that affects the statement’s mean score. Overall, the usability is getting positive feedback from users, which also further supports the effectiveness of the prototype, and certain aspects of the game’s enjoyment and playability could be enhanced.

6.2.2 Performance of prototype

The performance of the prototype has been analysed based on three factors, which are stability, reliability and efficiency. The analysis of user feedback on the performance of the developed prototype shows that the system is largely successful in maintaining stability and other factors of networking in the prototype with an overall eman performance score of 4.04. Users consistently agreed that the game maintained a stable connection without frequent disconnects with the implementation of host migration technique in Photon Fusion. However, the reliability and efficiency aspects were also rated positively, they scored slightly lower, with mean values of 3.98 and 4.02 respectively. This suggests that while the prototype is effective, there is room for improvement, particularly in minimizing disruptions during gameplay and ensuring quick response times to user inputs.

6.3 Contribution

Table 6.1 Summary on contribution from study

Objective	Research Question	Contribution
1(a)	RQ 1	Provide practical framework for integrating Photon Fusion into multiplayer games
1(b)	RQ 2	Develop a dynamic host migration to transfer host responsibilities without disrupting gameplay Address key technical challenges in implementing host migration
1(c)	RQ 3	Demonstrate dynamic host migration reduces in-game disruptions

Table 6.1 presents a summary of the contributions made by this research. This study has contributed on contributions related to network factors in game.

6.4 Implication of research

The finding of this research can have a positive impact on those involved in designing or developing multiplayer games using Photon Fusio.

i. Multiplayer Game Developer

The successful implementation of Photon Fusion and the host migration feature in this project provides valuable blueprint for the developers to seek chances to enhance the performance of their multiplayer games. The techniques can be applied to the development of other multiplayer games, not only limited to multiplayer FPS games. Developers can potentially reduce the frequency of in-game disruption, leading to a more seamless and enjoyable player experience.

ii. Academic Researchers

The implications of this research extend to the academic community as well, providing a solid foundation for further studies in multiplayer game technology. Researchers can build on this work to explore new approaches to reducing latency, improving server performance, and enhancing the overall multiplayer experience. Additionally, the research underscores the potential for collaborative efforts between academia and the gaming industry to drive technological advancements that benefit both sectors.

6.5 Limitation of research

While conducting this study, several limitations had to be overcome for the research to progress. The primary limitation was the availability of devices and personnel needed for testing the prototype. This is because this research is a multiplayer game which required at least two devices with the similar hardware specifications to debug and test the prototype to ensure everything works smoothly.

The second limitation was the scope of the game environment and the complexity of the networking scenarios. The game was developed with few players and relatively simple mechanics, which may not fully represent the challenges faced in larger-scale or more complex multiplayer games.

6.6 Recommendation of future study

Based on the study's findings and the feedback from experts and users, several recommendations for further research have been identified. One recommendation is to conduct research that is focused not only on the networking aspects but also on the overall game interface design, including the audio and game flow. Further studies could also investigate the scalability of the network solution, particularly in handling a significant increase in the number of simultaneous players.

6.7 Summary

This research has developed a prototype of multiplayer FPS games, providing an overview of the key conclusions from the multiplayer FPS games, emphasizing its contributions to game technology and the implications for future development. While the research had certain limitations, the findings offer valuable insights and recommendations that can guide future studies and practical applications in the field of multiplayer game development.

REFERENCES

Buchwalow, I. B., and Böcker, W. (2010). Immunohistochemistry: basics and methods. Berlin: Springer Verlag.

Caamaño-Tubío, R. I., Pérez, J., Ferreiro, S., and Aldegunde, M. (2007). Peripheral serotonin dynamics in the rainbow trout (Oncorhynchus mykiss) Comparative Biochemistry and Physiology Part C: Toxicology & Pharmacology. 145(2): 245-255.

Cameron, A. A., Plenderleith, M. B. and Snow, P. J. (1990). Organization of the spinal cord in four species of elasmobranch fishes: cytoarchitecture and distribution of serotonin and selected neuropeptides. The Journal of Comparative Neurology. 297: 201-218.

Desa, M.I (1995). Bus fleet maintenance modeling in a developing country. Ph.D Thesis, University of Salford.

Improve indigenous housing now, government told, 2007. accessed 8 February 2009, <http://www.architecture.com.au/i-cms?page=10220>.

International Narcotics Control Board 1999, United Nations, accessed 1 October 1999, <http://www.incb.org>.

Theusen, G. J. and Fabrycky, W. J. (1984). Engineering Economy. 6th. Ed. Englewood Cliffs, N. J.: Prentice Hall. 150-178.

Abidi, F. and V. Singh (2013). Cloud servers vs. dedicated servers—A survey. 2013 IEEE International Conference in MOOC, Innovation and Technology in Education (MITE), IEEE.

Adams, E. and J. Dormans (2012). Game mechanics: advanced game design, New Riders.

Armitage, G. (2008). Over-estimation of game server RTT during FPS server discovery, CAIA Technical Report.

Armitage, G., et al. (2006). Networking and online games: understanding and engineering multiplayer Internet games, John Wiley & Sons.

Arsenault, D. (2009). "Video game genre, evolution and innovation." Eludamos: Journal for computer game culture **3**(2): 149-176.

Bartle, R. A. (2004). Designing virtual worlds, New Riders.

Beigbeder, T., et al. (2004). The effects of loss and latency on user performance in unreal tournament 2003®. Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games.

Carroll, M., et al. (2019). Effects of VR gaming and game genre on player experience. 2019 IEEE Games, Entertainment, Media Conference (GEM), IEEE.

Chen, L., et al. (2016). Investigating the security and digital forensics of video games and gaming systems: A study of PC games and PS4 console. 2016 International Conference on Computing, Networking and Communications (ICNC), IEEE.

Claypool, K. T. and M. Claypool (2007). "On frame rate and player performance in first person shooter games." Multimedia systems **13**(1): 3-17.

De La Cruz, A. and J. Ryan (2015). "Tennis for Two."

Dick, M., et al. (2005). Analysis of factors affecting players' performance and perception in multiplayer games. Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games.

Dueck, P. (2021). Network Factors Influencing Packet Loss in Online Games, University of Saskatchewan.

Fernandes, L. V., et al. (2018). A survey on game analytics in massive multiplayer online games. 2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), IEEE.

Finn, M. (2002). Console Games in the Age of Convergence. CGDC Conf.

Ito, M. (1994). Cybernetic fantasies: extensions of selfhood in a Multi-User Dungeon. American Anthropological Association annual meeting, Atlanta.

Jansz, J. and M. Tanis (2007). "Appeal of playing online first person shooter games." Cyberpsychology & behavior **10**(1): 133-136.

Jiang, X., et al. (2005). Latency and scalability: a survey of issues and techniques for supporting networked games. 2005 13th IEEE International Conference on Networks Jointly held with the 2005 IEEE 7th Malaysia International Conf on Communic, IEEE.

Kangiwa, B. I. (2014). "CROSS PLATFORM ONLINE GAMING AS A SERVICE: A CLOUD COMPUTING PERSPECTIVE." WATARI Multidisciplinary Journal of Science & Technology Education **2**(1): 163-171.

Lee, W.-K. and R. K. Chang (2015). Evaluation of lag-related configurations in first-person shooter games. 2015 international workshop on network and systems support for games (NetGames), IEEE.

Li, F. W. (2008). Computer Games, Citeseer.

Liu, Y., et al. (2006). FPS Game Performance in Wi-Fi Networks. 4th International Game Design and Technology Workshop and Conference (GDTW2006), Citeseer.

Molyneux, L., et al. (2015). "Gaming social capital: Exploring civic value in multiplayer video games." Journal of Computer-Mediated Communication **20**(4): 381-399.

Monnens, D. and M. Goldberg (2015). "Space Odyssey: The long journey of Spacewar! from MIT to computer labs around the world." Kinephanos: Journal of Media Studies and Popular Culture: 124-147.

Oluwatosin, H. S. (2014). "Client-server model." IOSR Journal of Computer Engineering **16**(1): 67-71.

Rettberg, S., et al. (2008). "Corporate ideology in World of Warcraft." Digital culture, play, and identity: A World of Warcraft reader: 19-38.

Rideout, V. J., et al. (2010). "Generation M 2: Media in the Lives of 8-to 18-Year-Olds." Henry J. Kaiser Family Foundation.

Saldana, J. and M. Suznjevic (2015). "QoE and latency issues in networked games." Handbook of digital games and entertainment technologies: 1-36.

Süselbeck, R., et al. (2009). Peer-to-peer support for low-latency massively multiplayer online games in the cloud. 2009 8th Annual Workshop on Network and Systems Support for Games (NetGames), IEEE.

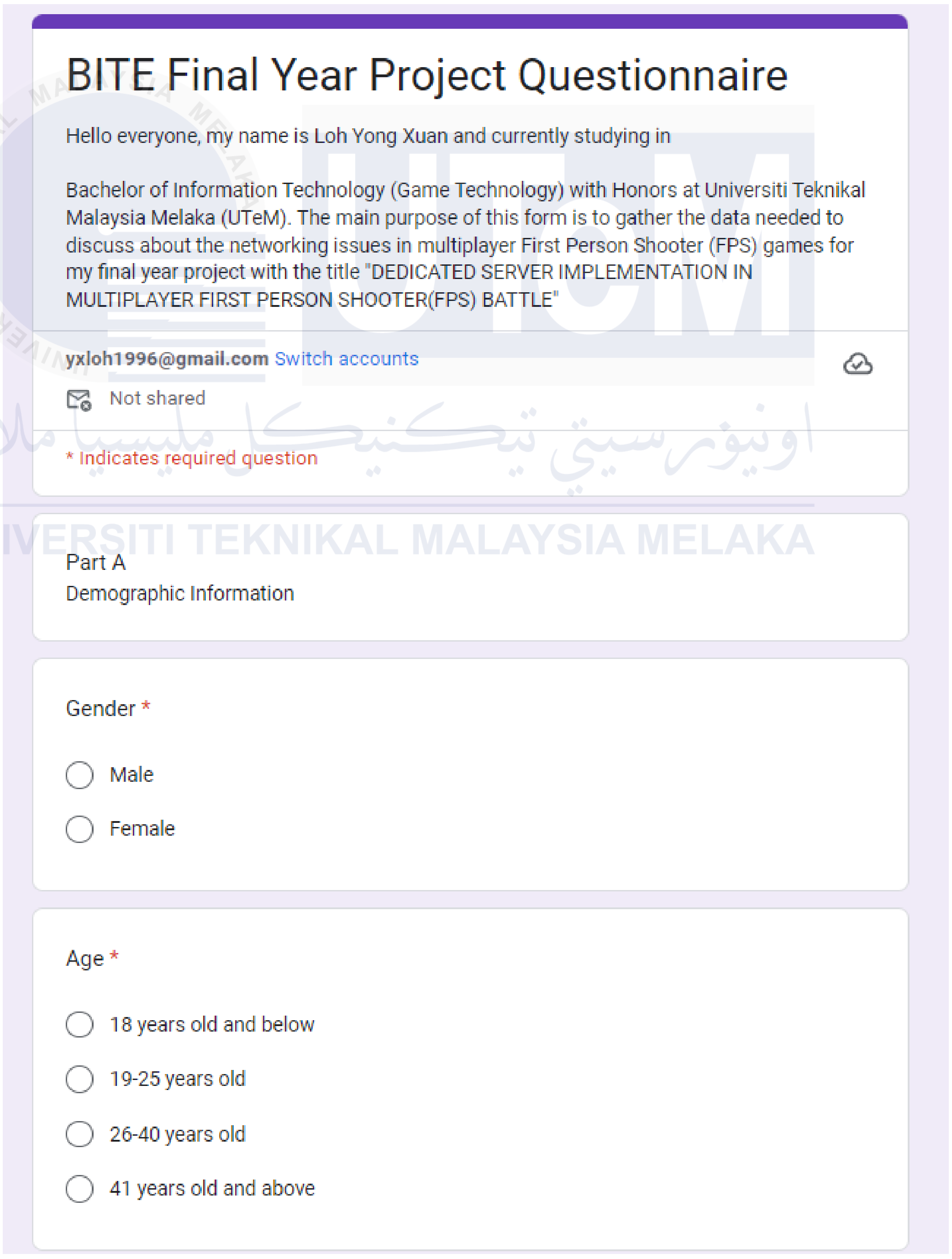
Therrien, C. (2015). "Inspecting video game historiography through critical lens: Etymology of the first-person shooter genre." Game Studies **15**(2).

Wang, Y. (2021). The Evolving Game Mechanics in Esports First-Person Shooter Games, Northeastern University.

Woolley, D. R. (1994). "PLATO: The emergence of online community." Social Media Archeology and Poetics, MIT Press, Cambridge.

Wydmuch, M., et al. (2018). "Vizdoom competitions: Playing doom from pixels." IEEE Transactions on Games **11**(3): 248-259.

APPENDIX A



BITE Final Year Project Questionnaire

Hello everyone, my name is Loh Yong Xuan and currently studying in

Bachelor of Information Technology (Game Technology) with Honors at Universiti Teknikal Malaysia Melaka (UTeM). The main purpose of this form is to gather the data needed to discuss about the networking issues in multiplayer First Person Shooter (FPS) games for my final year project with the title "DEDICATED SERVER IMPLEMENTATION IN MULTIPLAYER FIRST PERSON SHOOTER(FPS) BATTLE"

yxloh1996@gmail.com Switch accounts

Not shared

* Indicates required question

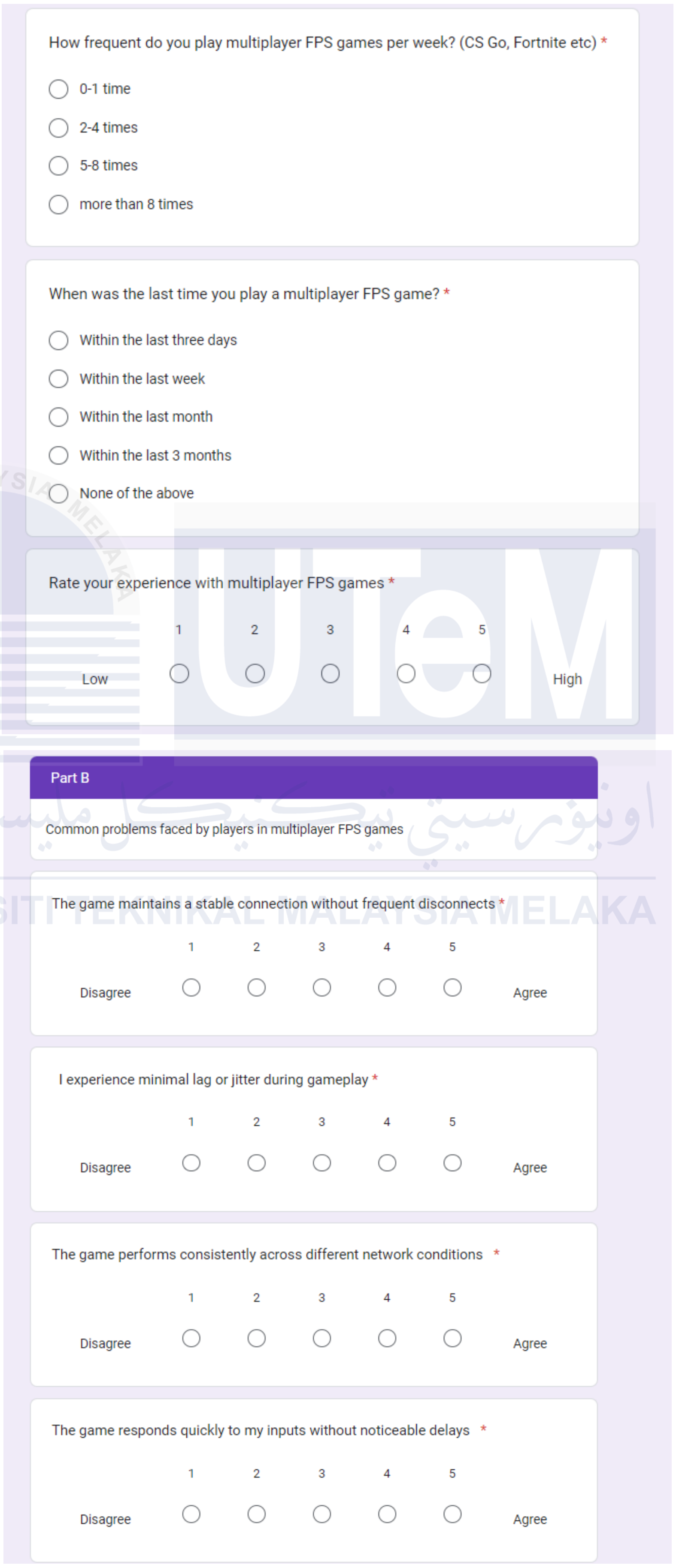
Part A
Demographic Information

Gender *

- ○ Male
- ○ Female

Age *

- ○ 18 years old and below
- ○ 19-25 years old
- ○ 26-40 years old
- ○ 41 years old and above



How frequent do you play multiplayer FPS games per week? (CS Go, Fortnite etc) *

- ◯ 0-1 time
- ◯ 2-4 times
- ◯ 5-8 times
- ◯ more than 8 times

When was the last time you play a multiplayer FPS game? *

- ◯ Within the last three days
- ◯ Within the last week
- ◯ Within the last month
- ◯ Within the last 3 months
- ◯ None of the above

Rate your experience with multiplayer FPS games *

	1	2	3	4	5	
Low	◯	◯	◯	◯	◯	High

Part B

Common problems faced by players in multiplayer FPS games

The game maintains a stable connection without frequent disconnects *

	1	2	3	4	5	
Disagree	◯	◯	◯	◯	◯	Agree

I experience minimal lag or jitter during gameplay *

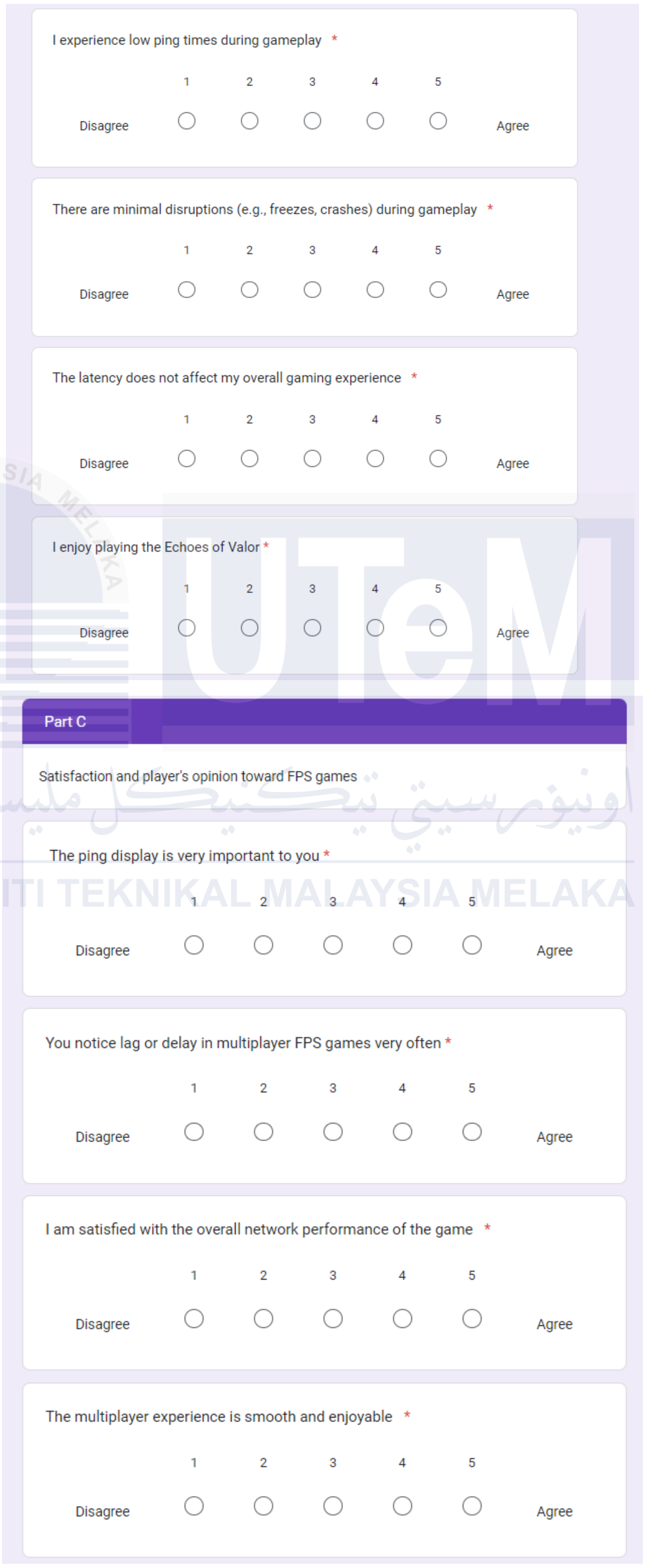
	1	2	3	4	5	
Disagree	◯	◯	◯	◯	◯	Agree

The game performs consistently across different network conditions *

	1	2	3	4	5	
Disagree	◯	◯	◯	◯	◯	Agree

The game responds quickly to my inputs without noticeable delays *

	1	2	3	4	5	
Disagree	◯	◯	◯	◯	◯	Agree



I experience low ping times during gameplay *

	1	2	3	4	5	
Disagree	○	○	○	○	○	Agree

There are minimal disruptions (e.g., freezes, crashes) during gameplay *

	1	2	3	4	5	
Disagree	○	○	○	○	○	Agree

The latency does not affect my overall gaming experience *

	1	2	3	4	5	
Disagree	○	○	○	○	○	Agree

I enjoy playing the Echoes of Valor *

	1	2	3	4	5	
Disagree	○	○	○	○	○	Agree

Part C

Satisfaction and player's opinion toward FPS games

The ping display is very important to you *

	1	2	3	4	5	
Disagree	○	○	○	○	○	Agree

You notice lag or delay in multiplayer FPS games very often *

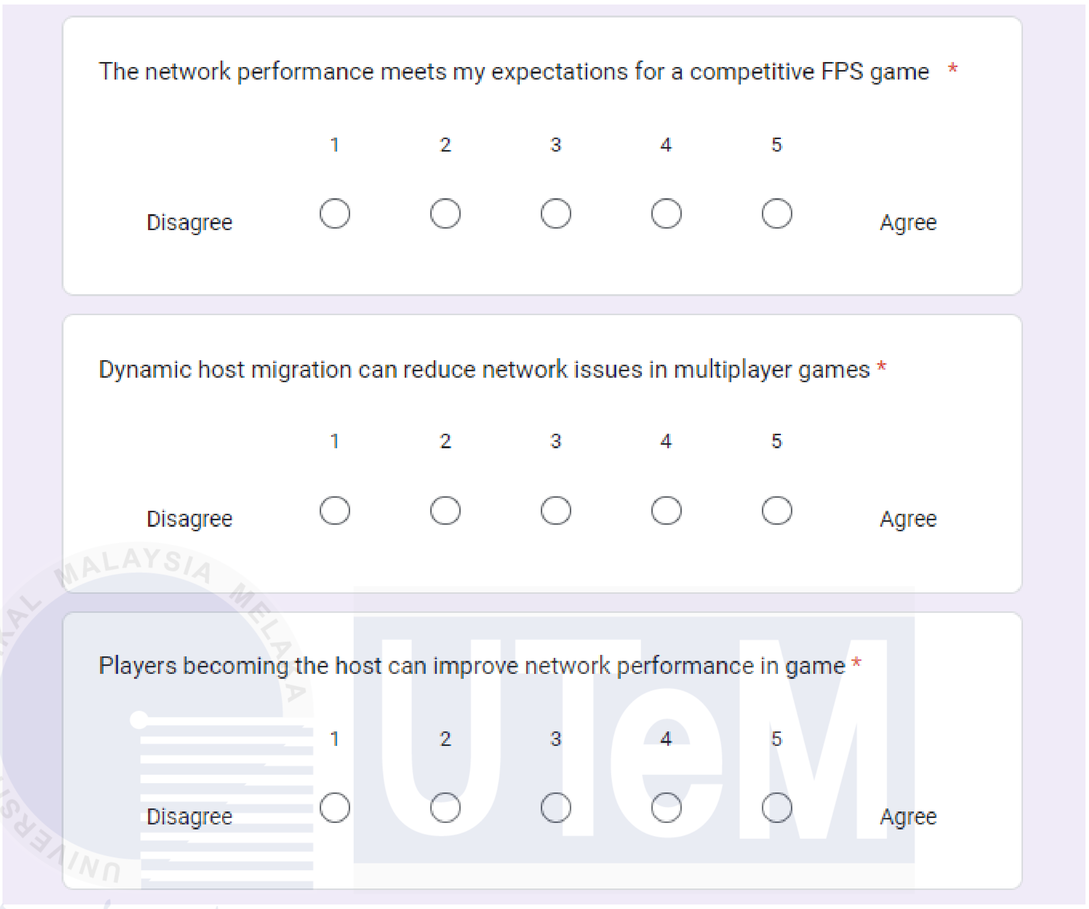
	1	2	3	4	5	
Disagree	○	○	○	○	○	Agree

I am satisfied with the overall network performance of the game *

	1	2	3	4	5	
Disagree	○	○	○	○	○	Agree

The multiplayer experience is smooth and enjoyable *

	1	2	3	4	5	
Disagree	○	○	○	○	○	Agree



The network performance meets my expectations for a competitive FPS game *

	1	2	3	4	5	
Disagree	○	○	○	○	○	Agree

Dynamic host migration can reduce network issues in multiplayer games *

	1	2	3	4	5	
Disagree	○	○	○	○	○	Agree

Players becoming the host can improve network performance in game *

	1	2	3	4	5	
Disagree	○	○	○	○	○	Agree