Multiplayer Game Using Photon Unity Networking and Unity Engine

A Practice School Report submitted to
Manipal Academy of Higher Education
in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

in

Computer Science & Engineering

Submitted by
UJJAWAL VIVEK
150905256
Under the guidance of

Dr. Mamatha Balachandra
Associate Professor, Senior Scale
Computer Science and Engineering
Manipal Institute of Technology

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

May 2021

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Manipal
25 May 2021

CERTIFICATE

This is to certify that the project titled **Multiplayer Game Using Photon Unity Networking and Unity Engine** is a record of the bonafide work done by **Ujjawal Vivek [150905256]** submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in **COMPUTER SCIENCE & ENGINEERING** of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2021.

Dr. Mamatha Balachandra
Associate Professor, Senior Scale
M.I.T, MANIPAL

Prof. Dr. Ashalatha Nayak
HOD, CSE Dept.
M.I.T, MANIPAL

ACKNOWLEDGMENTS

I am highly thankful to **Dr. Mamatha Balachandra** for guiding me throughout the past 4 months to undergo the internal project held under the guidance of **Manipal institute of Technology** in the field of **Computer Science.** She guided me from time to time and gave me dynamic ideas and suggestions by which I was able to complete my project successfully.

I am also thankful to the college for providing me this opportunity to carry out the project in the field of networking, which I want to make a career out of, in the future.

I am also thankful to all the teachers on my midterm viva panel who clarified my doubts & confusion and gave me firm feedbacks. It was a great and satisfying experience on having to work on a project under the guidance of the college. I also want to thank all the visible and non-visible hands, which helped me to complete the internal project with great success.

I would also like to thank my university, **Manipal University**, for giving me the chance to pursue this project.

Ujjawal Vivek

ABSTRACT

The project named **Multiplayer Game Using Photon Unity Networking and Unity Engine** is where the objective was to build a working networked multiplayer game. The concept of multiplayer game is not new, it has been there for quite a while now. This project aims at learning how the networking aspects work in a multiplayer game, how to reduce latency issues in a basic peer to peer based connections and utilize computational resources efficiently. The objective of this game is be the last one standing. The game was typically intended to be an FPS style game but needed a unique aspect to set the game aside from all the FPS's already on the market. [1]

Unity Engine along with Photon Unity Networking using the Photon 4 api was used to achieve the final result, that was a working game with very little networking issues. The coding language used to build the game was C# using the .net libraries and IL2CPP to build the project.

There are a couple of drawbacks linked with the development of an FPS game. It was most prominent during development that a multiplayer game is particularly large and complicated to develop thus requiring a considerable amount of time to even get it to a playable state. Especially for a multiplayer game, where I had to account for latency and lag issues due to heavy data being sent over the network.

Started with Unity 3D game engine and Photon Unity Networking solution to develop my game. Though PUN does simplify the process, creating any multiplayer game can still be a daunting task. Its comparatively easier as both has a large community which provides learning support from the Unity website and forums. [2] The game can be coded in C# which made it ideal, as this language has been covered in our curriculum. The game will run on PC which gives me the option of publishing my game to Steam, thus making it available to multiple users online.

Table of Contents

List of Figures

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

The games industry has grown significantly in recent years. It is a great place to get engaged in as it allows resourcefulness, innovation, and freedom for developers. As the new 'indie' market for games exploded, the gaming industry now allows smaller scale games to be developed and released more often than before. They get a opportunity to experiment with all types of media. Indie games have more innovation and developers are willing to take risks on their games. 'Indie' games also seem to have a huge market base and many games are being released on PC, Android etc.

Started with Unity 3D game engine and Photon Unity Networking solution to develop my game. Though PUN does simplify the process, creating any multiplayer game can still be a daunting task. Its comparatively easier as both has a large community which provides learning support from the Unity website and forums. [2] The game can be coded in C# which made it ideal, as this language has been covered in our curriculum. The game will run on PC which gives me the option of publishing my game to Steam, thus making it available to multiple users online.

1.2 MOTIVATION

The motivation for the project of choice was a multiplayer game using Unity Engine, which is my career choice moving forward in life. I wanted to experiment with game development and decided to up it one level by adding networking aspect to it. I have read through many research papers regarding game development and networking, and most of them do not tackle the difficulties encountered while making a multiplayer game, which is why I decided to do the same and making a note of all the difficulties I encountered during the development phase and how I solved them using various strategies. Importance of this project comes in the form of my career choice ahead in life, and I am very much grateful to the college for giving me such opportunity to pursue the same.

I used Unity engine with the Photon Unity Networking library to achieve the results as laid out in the later parts of this report. The coding language of choice was C# along with Visual Studio IDE for development purpose.

1.3 OBJECTIVES

The objective of this internal project was to create a fully functional FPS game. The game scene has no light and there is no possible way to find players unless they come close. This adds a level of scare in the game as you will not know how close the next player is or when they will attack you. I included sound effects to boost the ambience of the game. The rationale of this project is to build a fps game with good quality graphics, audio and animations made with essential technologies and a minimal budget.

Unity's scripting is based primarily on Visual Studio which is an implementation within the .Net framework. When working with Unity, C# is the language you have to code in. Unity makes use of Visual Studio. Unity supports cross platform deployment. When a project is completed, developers can choose which platform they would like their project to run on, e.g., PlayStation, Xbox, Window, Linux, Android, IOS and many others. I am currently using Unity version 2019 LTS in the development of my game.[3]

This report talks about the literature used for the project, the shortcomings, the objectives, methodology, results, and conclusions in the later parts of this report. Then I talk about the future scope of the project and finally I have included project details and references.

CHAPTER 2
BACKGROUND THEORY/ LITERATURE REVIEW

2.1 BACKGROUND

The games industry has grown significantly in recent years. It is a great place to get engaged in as it allows resourcefulness, innovation and freedom for developers. As the new 'indie' market for games exploded, the gaming industry now allows smaller scale games to be developed and released more often than before. They get a opportunity to experiment with all types of media. Indie games have more innovation and developers are willing to take risks on their games. 'Indie' games also seem to have a huge market base and many games are being released on PC, Android etc.[4]

Indie games shows innovation and that the developers are prepared to take risks on their games. Indie games have a fairly large market base with many of the games being released on steam and play store. Currently there is a massive surge in the popularity of particularly innovative horror games. I put a lot of research into trying to find out what users want from a game. I had to decide between a first person shooter or a third person shooter, and what type of environment would be best to build my game. I finally decided to go with an FPS in a horror style environment.

I decided to go with a 3D game over a 2D game as I think it makes the game that extra bit challenging and interesting. There are also benefits to the style of game I chose.

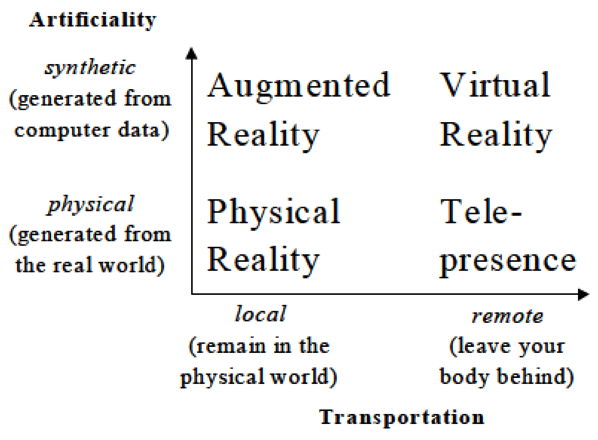

Artificiality
synthetic (generated from computer data)
physical (generated from the real world)
Augmented Reality
Virtual Reality
Physical Reality
Tele-presence
local (remain in the physical world)
remote (leave your body behind)
Transportation

Fig 2.1.1: Classification of shared-space technologies according to transportation and artificiality.

"While one widely held view maintains that playing video games is intellectually lazy, such play actually may strengthen a range of cognitive skills such as spatial navigation, reasoning, memory, and perception, according to several studies reviewed in the article. This is particularly true for shooter video games, which are often violent, the authors found. A 2013 meta-analysis found that playing shooter video games improved a player's capacity to think about objects in three dimensions just as well as academic courses designed to enhance these same skills, according to the study." ***(Apa.org, 2016)***[1]

2.2 *LITERATURE* REVIEW

The concept of multiplayer game is not new, it has been there for quite a while now. This project aims at learning how the networking aspects work in a multiplayer game, how to reduce latency issues in a basic peer to peer based connections and utilize computational resources efficiently. Creating games is already a complex process, and even more for multiplayer games. Good thing that there is a helpful plugin that makes the multiplayer game creation a much easier process. That plugin is PUN (Photon Unity Networking) created by Exit Games. This plug-in specializes in easing the multiplayer game creation process by handling aspects such as matchmaking, client to server architecture, and cracking down on issues such as latency. Though PUN does simplify the process, creating any multiplayer game is still be a daunting task.

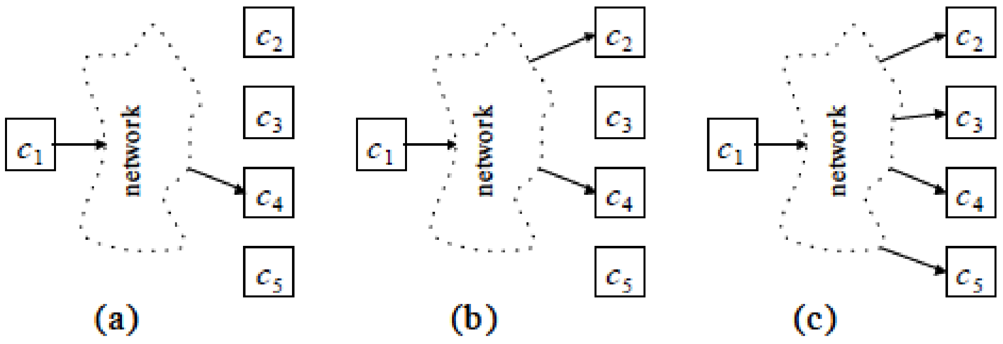


Fig 2.2.1: Transmission Technique: [5]

(a) In unicasting, the message is sent to a single receiver.

(b) In multicasting, the message is sent to one or more receivers that have joined a multicast group.

(c) In broadcasting, the message is sent to all nodes in the network.

In my view there are many advantages and disadvantages to creating this type of a game. I liked that I had complete liberty with the ingenuity of the game, and I could execute it with my vision in mind. Another gain was the familiarity of coding in C# in previous college modules. I was pleasingly surprised with how much coding experience I picked up. I also got to work around with fresh technologies. There are a few disadvantages associated with the development of an FPS game. It was apparent during development that this type of game is extremely large and complex to develop thus requiring a significant amount of time to even get it to a playable state. Especially for a multiplayer game, where I had to account for latency and lag issues due to heavy data being sent over the network.

Distributed, real-time multiplayer computer games (MCGs) are in the need of utilizing the networking possibilities. Even though similar research has been done in various simulations, and computer supported cooperative working, the suggested solutions deviate from the problems posed by Multiplayer Games. With this in mind, this project aims to provide a concise overview of some aspects affecting networking in MCGs. Firstly, networking resources (bandwidth, latency, and computational power) set the technical boundaries within which the MCG must operate. Secondly, distribution concepts encompass communication architectures (peer-to-peer, client/server, server network), and both data and control architectures (centralized, distributed, replicated). Thirdly, scalability allows the MCG to adapt to the resource changes by parametrization. Finally, security aims at fighting back against cheating and hacking, which are common in online gaming.

NETWORKING RESOURCES – LATENCY [6]

Networking latency indicates the length of time (or delay) that incurs when a message gets from one designated node to another. In addition, the variance of latency over time (i.e., jitter) is another feature that affects interactive applications. Latency cannot be totally eliminated. For example, speed-of-light propagation delays and the slowdown of electrical signal in a cable alone yield a latency of 25– 30ms for crossing the Atlantic. Moreover, routing, queuing and packet processing delays add dozens of milliseconds to the overall latency. For interactive real-time systems such as MCGs, the rule of thumb is that latency between 0.1 and 1.0 seconds is acceptable. For instance, the DIS standard specifies that the network latency should be less than 100ms. Latency affects the user’s performance nonlinearly: Continuous and fluid control is possible when the latency does not exceed 200 ms, after which the interaction becomes more observational and cognizant. Consequently, the threshold of when latency become inconvenient for the user depends on the type of application. In a real-time strategy (RTS) game, a higher latency (even up to 500 ms) may be acceptable as long as it remains static (i.e., jitter is low). Interestingly, experiments on CVEs have yielded similar results. On the other hand, games requiring a tight hand eye motor control such as first person shooters (FPSs) demand that the latency runs closer at 100 ms.

NETWORKING RESOURCES – COMPUTATIONAL POWER [6]

Network traffic handling puts additional computational strain on the computer running a distributed system. This limited resource can be easily overlooked and usually the network alone is deemed as the source of problems. By using quit conservative estimates, Singhal [24] demonstrates how a simulation handling the network traffic of 100,000 entities (i.e., participating objects) can require up to 80 percent of the total CPU time on a 500 MHz processor. Furthermore, packet delivery demands are unlikely to be satisfied in the future due to increasing processing requirements in distributed systems (e.g., more participants, more interaction) and the slow increase in hardware memory speeds.

DISTRIBUTION CONCEPT

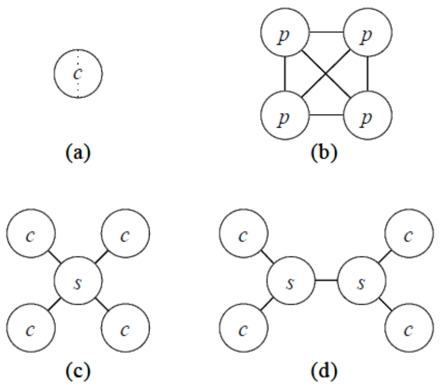


Fig 2.2.2: Distribution Techniques -

(a) A split screen on a single computer

(b) Peer – Peer architecture

(c) A client – server architecture

(d) A server – network architecture

CHAPTER 3
METHODOLOGY

3.1 UNITY ENGINE

Unity [3] is a cross platform game engine with its own built in IDE which allows for development of either 2D or 3D games. Unity is primarily used to create games for a variety of platforms such as web, desktop, consoles and most recently, mobile platforms. Unity is the primary tool that was used to create this project. It was chosen primarily because of the fact that it is free to use and has a vast amount of support available from the community including an asset store from which some free assets can be added to a project in order to enhance it as well as saving time and money.

Unity features a main scene view which displays the visual elements of the project such as the terrains, models, etc. Any object that is used in Unity, whether it is a terrain, a shape, a model, is referred to as a GameObject. These game objects can be modified using the Inspector view. With the Inspector view you can see the transform, rotation, and scale information of the selected GameObject. You can also see the scripts that the object is using, and the components attached to it. Unity offers support for assets that are created in a wide variety of file formats, such as fbx. Assets created in other programs can simply be added to a game project in Unity by choosing them from the saved location on your computer.

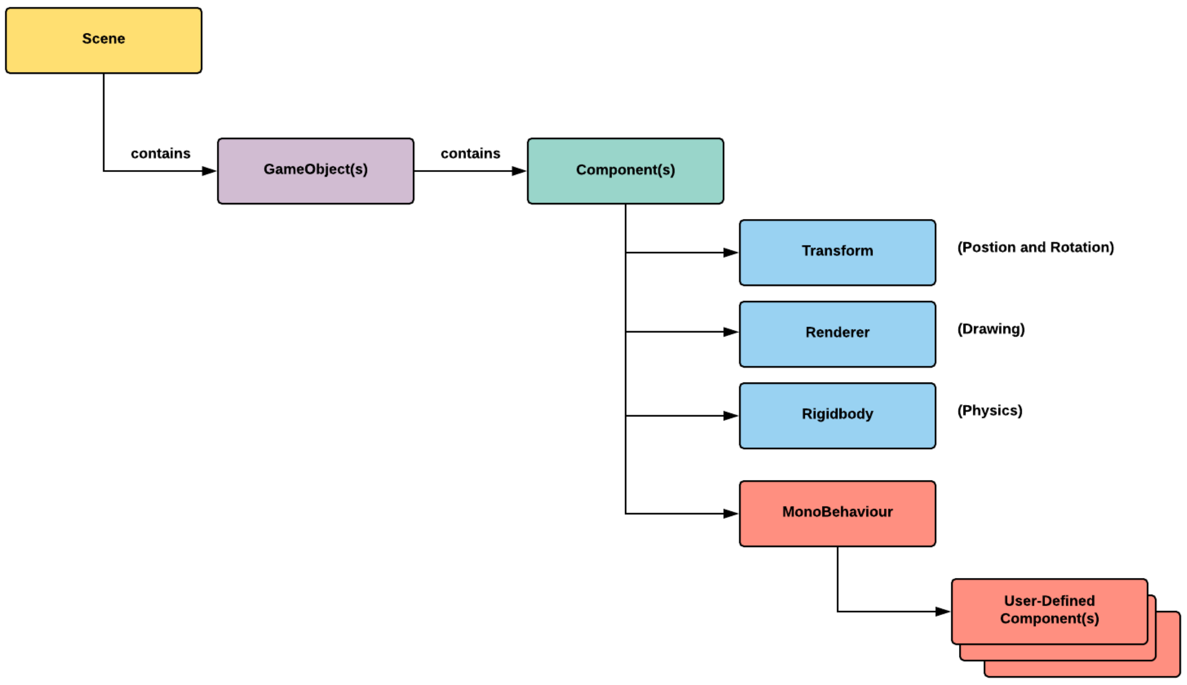


Fig 3.1.1: Unity Engine Workflow

Unity makes adding imported art assets to the game scene as simple as drag and drop in the desired location. Unity's scripting is based primarily on Visual Studio which is an implementation within the .Net framework. When working with Unity, C# is the language you have to code in. Unity makes use of Visual Studio.

Unity supports project deployment on multiple platforms. When a project is finished, developers can choose which platform they would like their project to run on, e.g., PlayStation, Xbox, Window, Linux, Android, IOS and many others. I am currently using Unity version 2019 LTS in the development of my game.

3.2 VISUAL STUDIO

Visual Studio [7] is an IDE which is installed with Unity. It allows you to create and edit C# files used in the game. It differs from the standard IDE because it has in-built functions and auto-completion with coding specifically relating to Unity. Visual Studio is able to interact with the GameObjects used in Unity. It can access their co-ordinates, scripts attached to them, their tags, etc. These are very useful in creating powerful scripts.

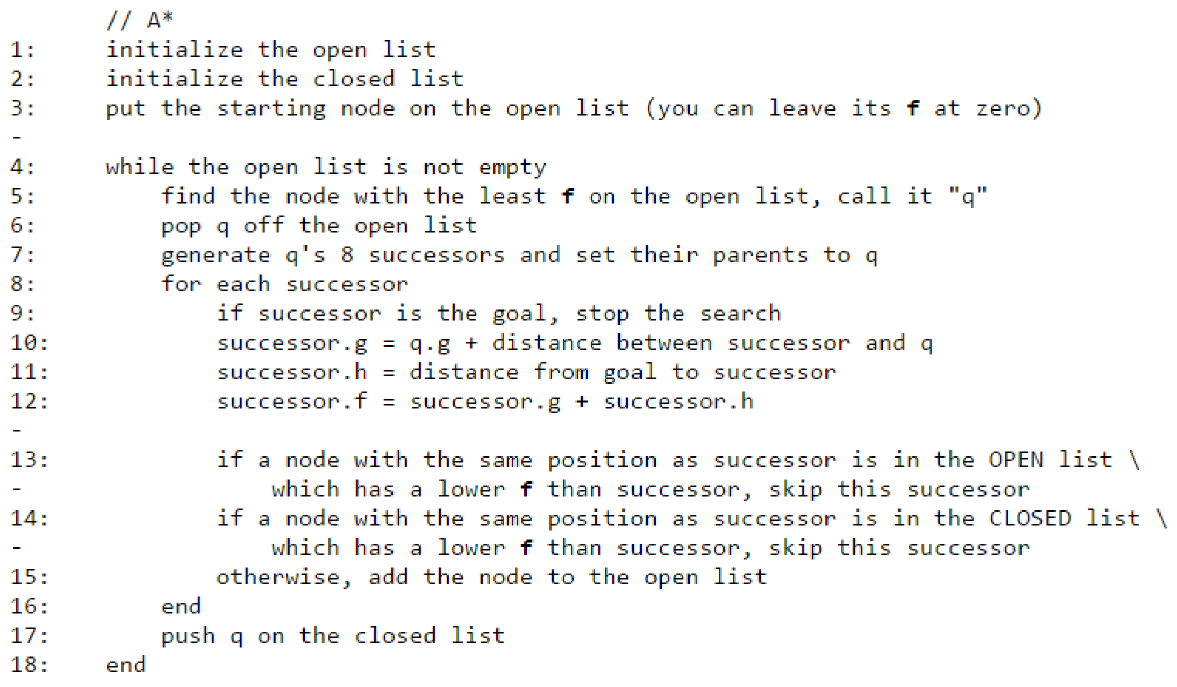
It also allows variables to be edited from Unity, so changes do not need to be made in the scripts every time. This made it really easy to test different things in my game as I could easily change variables such as distance or try different images or sound clips. Errors that appear within the IDE are also shown in the Unity Debugging Window which recompiles regularly. Double clicking the error will bring you to the line of code in the script that is causing the problem. This was very helpful during the development phase.

3.3 C#

C# [8] is a modern programming language which allows developers to create applications which will then run on the.NET framework. I had been introduced to C# in one of my modules in college which is a reason Unity was chosen. As I had been introduced to C# in college, I was familiar with it and it made it that bit easier when trying to fix problems in scripts. Each script that unity makes derives from MonoBehaviour which enables scripts to be put on GameObjects. We can derive such scripts from ScriptableObjects if we want the scripts to reside as an asset in the project folder. This makes it such that we can make it an abstract class and make multiple scripts deriving from the same for different items in a game which share the same code.

3.4 A PATH FINDING ALGORITHM*

This is a computer algorithm that is widely used in pathfinding [9] and graph traversal. A* tries to look for a better path by using heuristic functions to optimize the search which gives priority to nodes.



```
// A*
initialize the open list
initialize the closed list
put the starting node on the open list (you can leave its f at zero)

while the open list is not empty
    find the node with the least f on the open list, call it "q"
    pop q off the open list
    generate q's 8 successors and set their parents to q
    for each successor
        if successor is the goal, stop the search
        successor.g = q.g + distance between successor and q
        successor.h = distance from goal to successor
        successor.f = successor.g + successor.h

        if a node with the same position as successor is in the OPEN list \
            which has a lower f than successor, skip this successor
        if a node with the same position as successor is in the CLOSED list \
            which has a lower f than successor, skip this successor
        otherwise, add the node to the open list
    end
    push q on the closed list
end
```

Fig 3.4.1: A* Pathfinding algorithm pseudocode.

A* is typically used in games as it is useful when you are searching for a path on a map where you can guess the distance to the target from a given graph node. Grid graphs were used which are useful for ease of use and fast updates. These are built by arranging nodes in a grid like pattern around the world. They allow for fast updates due to their regular structure and are useful in games that require frequent updates to the graphs.

A* was beneficial in my game as I could easily set up a grid graph that showed the ground layer and the obstacles layer. I could assign objects in my game to a specific layer and decide if the enemy could walk on them or not. This made it so the enemy could not walk through walls or objects and they would have to go around them. The pathfinding also computed the best path to take to reach the specified destination while avoiding the obstacles and doing it in the least amount of time possible.

3.5 MULTIPLAYER GAME ARCHITECTURE

Multiplayer Game Architecture using unity engine

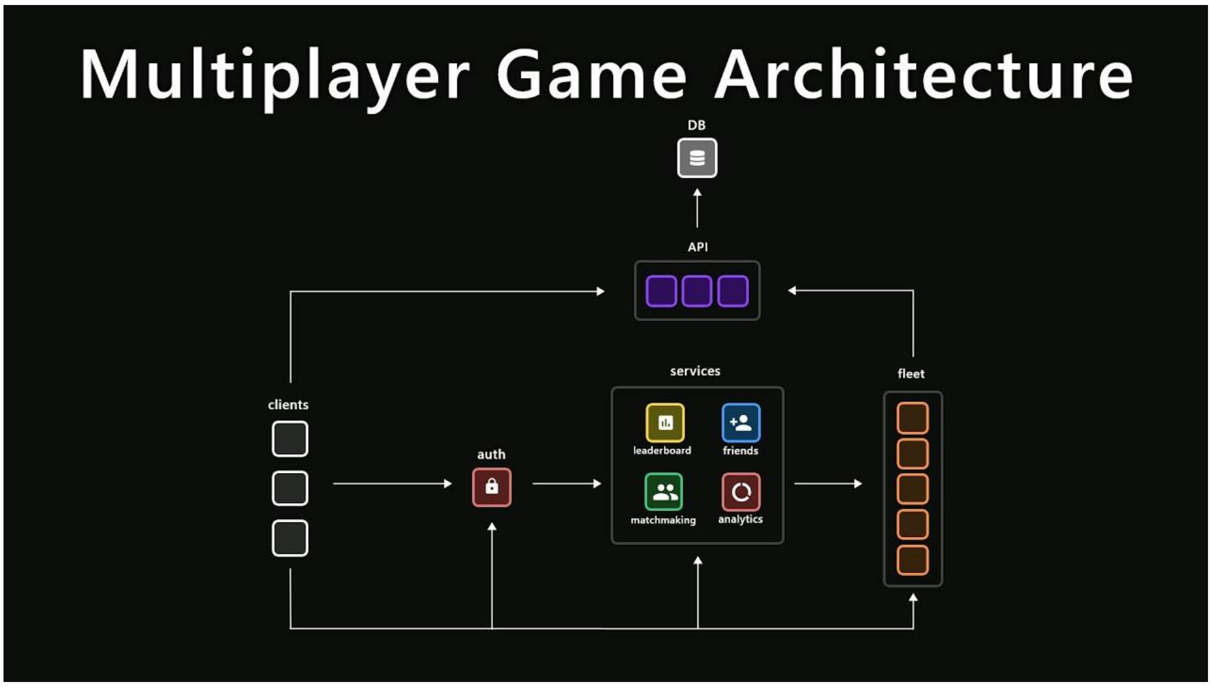


Fig 3.5.1: Multiplayer game architecture for a client/server architecture.

3.6 DATA REQUIREMENTS

All data is passed in the background. These are variables that monitor the positions of both the player and the other players and keep track of the players health. Information is only saved during a game. No information is saved when the application is closed. [10]

The system will constantly render textures and models. Graphics will need to be realistic to provide the user with a believable world. The game should also have suitable animations to compliment the realism of the graphics including walking, running, attacking etc. I feel that audio will help to complete this 3D environment. Background music to set the scene, footsteps, and gun shots should be incorporated.

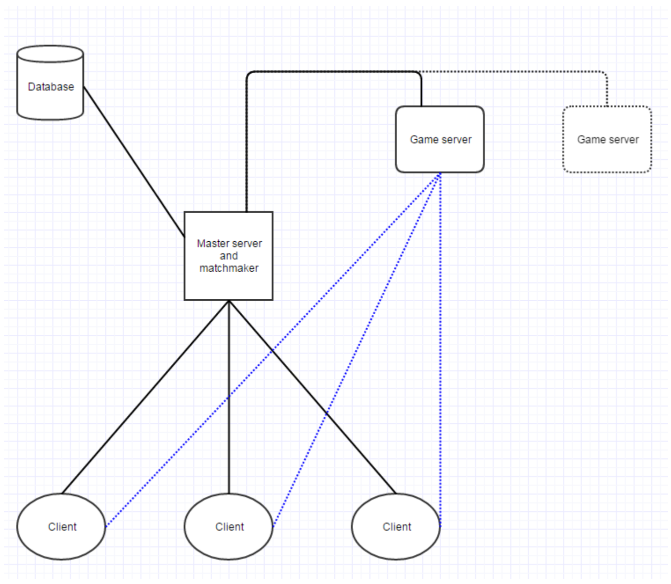


Fig 3.6.1: Data requirements for the client/server architecture.

The user requirements are used to describe what the user will need in order to run and play the game. The user must have a computer or laptop capable of running a modern version of the Windows operating system (must be Windows 8 or higher) and the graphics card requirements will vary depending on the type of game. As this project consists of simple graphics, the standard graphics cards in the users' PC should be able to run the game. Internet access will also be required for the user to be able to download the game and play it online. When the game is compiled, it is built into an executable file which is separate from Unity which will benefit the user as they will not be required to download and install Unity if they want to play the game.

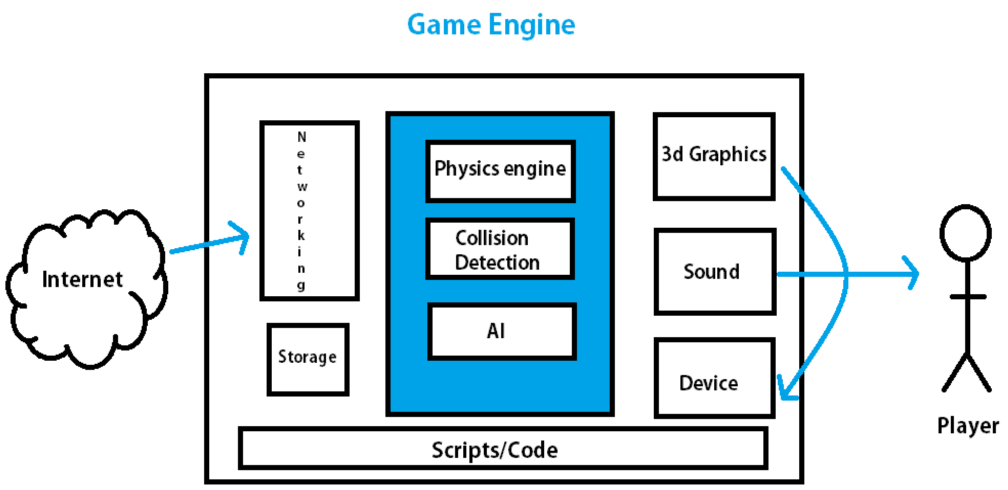


Fig 3.6.2: System architecture

When the player starts up the game the main menu will be displayed. The player will be presented with a couple options to choose from. The ‘Play Game’ option will bring the player to play the game. The ‘Controls' option will bring the player to a screen that will show the controls for the game and the ‘Exit Game’ option will allow the player to quit the game and exit the application. A pop-up menu will appear asking the player are they sure. If the user selects yes, the application will close.

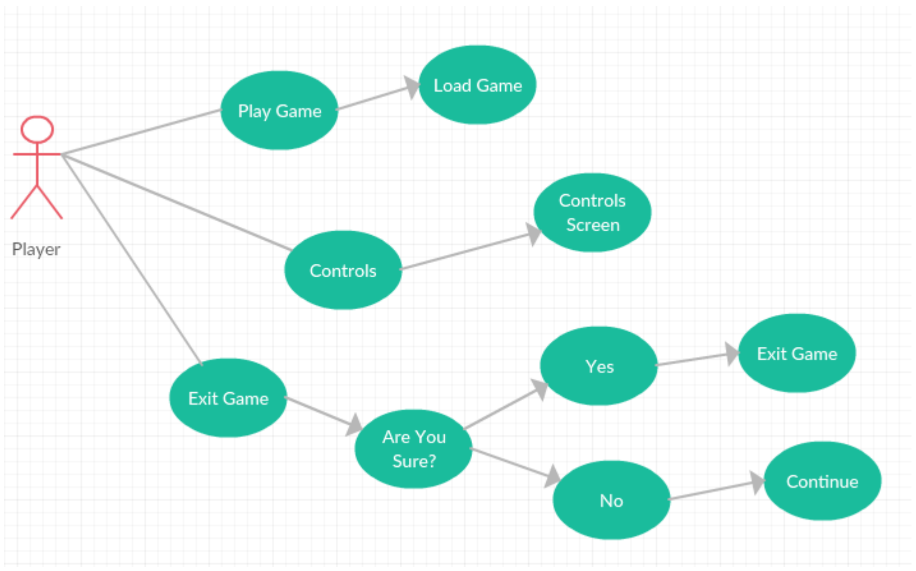


Fig 3.6.3: Use case diagram

3.7 TESTING [11]

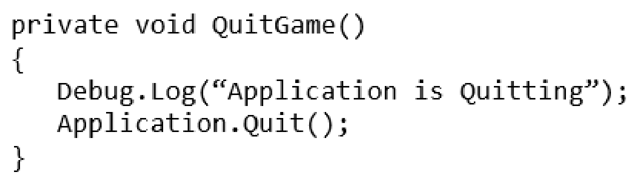
Game testing is a software testing process for quality control of games. Once code is generated, software must be tested to uncover as many errors as possible before publishing. Testing was very important for this project and many different forms of testing have been carried out in order to ensure that the software acts as expected in all situations as well as ensuring that the requirements have all been successfully met.

Usability Testing was used to identify how players would interact with the overall look of the game. I gave the game to a few of my friends and asked them what they thought of my game menu and scenes.

I used **White Box Testing** in order to test my logical paths through my game using specific sets of conditions and loops. I started my game from the very first scene and played it through to end to test if scenes connected the way they should. Scenes are connected by conditions that are met such as selecting a button on a menu, to waiting a certain number of seconds or to winning the game. The goal of the white box testing was to ensure that every possible condition that could have been met was.

Integration Testing was next after Unit testing. It was unrealistic just to test individual aspects of the game. Multiple processes within the game needed to be tested working in conjunction with each other. I tested to see if all my different code and game objects worked seamlessly together and did not overlap.

Unity has a debugger and console which can be used to monitor what the game is doing with information and functions. The console shows errors and warnings in scripts and double clicking will bring the user to the line of code that has the error. The Debug.Log() function and Debug.Print() function were also used to check if code was actually working. An example of this would be when the exit game button is pressed. The application will not close when you are developing the game so you can use Debug.Log to send text to the console to check if the code is executing.



```
private void QuitGame()
{
   Debug.Log("Application is Quitting");
   Application.Quit();
}
```

Fig 3.7.1: Quit Game Debugger

This is a form of **Black Box Testing** as it allows you to input information into the game and monitor the output to see if its returning what is expected. The system is a ‘black-box’ whose behaviour can only be determined by studying the inputs and outputs. Black box testing identifies errors, shows incorrect or missing functions and behaviour and performance errors.

CHAPTER 4
RESULT ANALYSIS

4.1 SCALABILITY ANALYSIS

Scalability [12] is the ability to adapt to the resource changes. In MCGs this concern, for example, how to construct an online server that dynamically adapts to varying number of players, or how to allocate the computation of non-player character among the nodes. To achieve this kind of scalability there must be physical (i.e., hardware-based) parallelism that enables logical (i.e., software) concurrency of computation.

The potential speedup obtained by applying multiple nodes is bounded by the system's inherently sequential computations. The time required by the serially executed parts cannot be reduced by parallel computation. Thus, the theoretical speedup gained (S) is:

$$S(n) = \frac{T(1)}{T(n)} \leq \frac{1}{n}$$

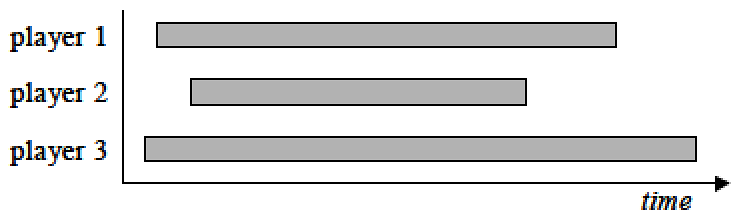
Fig 4.1.1: Scalability Analysis

where $T_{(1)}$ the time to execute with one node and $T_{(n)}$ with n nodes. The execution time can be divided into a serializable part T_S and parallel part T_P. Let $T_S+T_P = 1$ and $\alpha = T_S/(T_S+T_P)$. If the system is serialized optimally, the equation can be rewritten as:

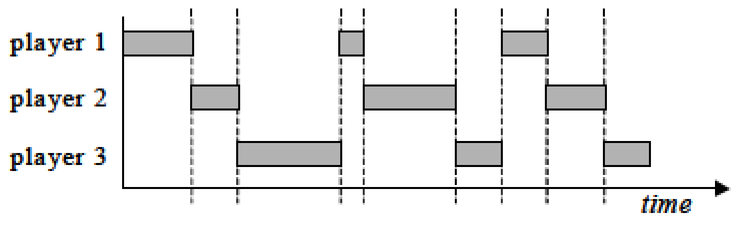
$$S(n) = \frac{T_s + T_p}{T_s + T_p/n} = \frac{1}{\alpha + (1-\alpha)/n} \leq \frac{1}{\alpha}.$$

Fig 4.1.2: Serialized Scalability

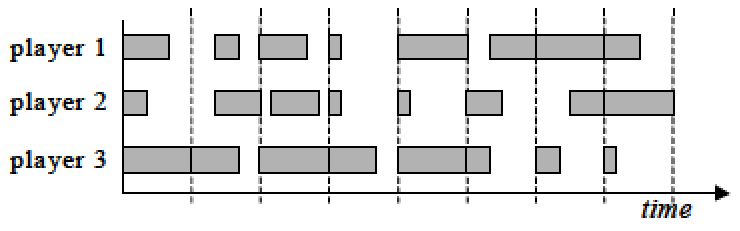
Ideally, the serializable part should be non-existent and, thus, everything would be computed in parallel. However, in that case there cannot exist any coordination between the nodes. The only example of such MCG is that each player is playing their own game regardless of the others. The other extreme is that there is no parallel part with respect to the game state, which is the case in a round robin or a turn-based game. Between these extremes are the MCGs which provide real-time interaction and which, consequently, comprise both parallel and serial computation.



(a) Separate real-time games can run in parallel but without interaction.



(b) A turn-based game is serialized and interactive but not real-time, unless the turns are very short.



(c) An interactive real-time game runs both in serial and in parallel.

Fig 4.1.3: Serial and Parallel executions in multiplayer games.

Deployment architecture	Capacity requirement
Single node	0
Peer-to-peer	$\sim n \ldots n^2$
Client/server	$\sim n$
Peer-to-peer server-network	$\sim \frac{n}{m} + m \ldots \frac{n}{m} + m^2$
Hierarchical server-network	$\sim n$

Fig 4.1.4: Communication capacity requirements for different deployment architectures.

The above table shows the Communication capacity requirements for different deployment architectures. Where:

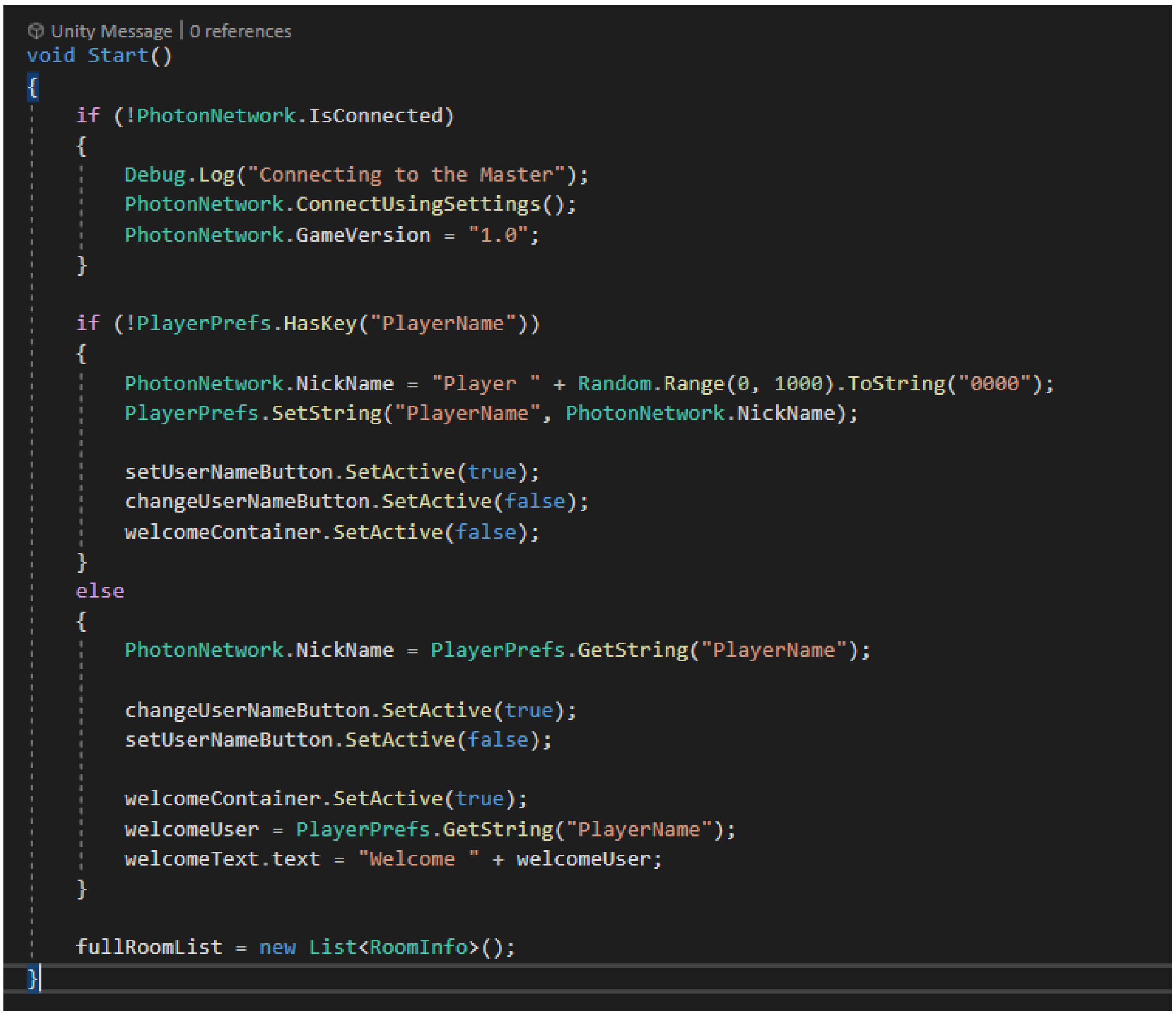
n is the number of clients active on the server.

m is the number of servers available.

4.2 GAME ANALYSIS

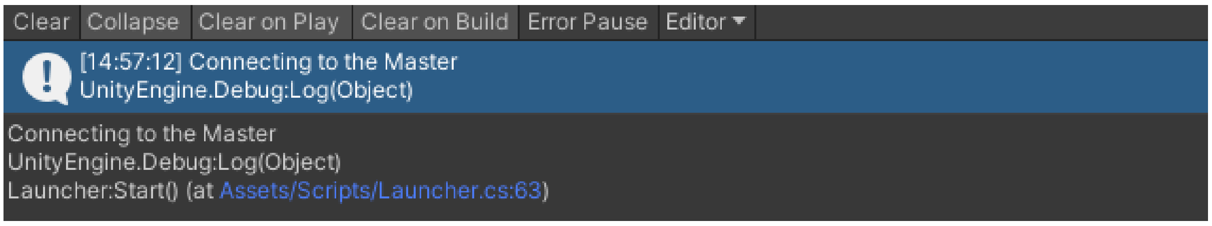
- Initial connection to the server

Fig 4.2.1: Initial connection to the server – c# code



```
Unity Message | 0 references
void Start()
{
    if (!PhotonNetwork.IsConnected)
    {
        Debug.Log("Connecting to the Master");
        PhotonNetwork.ConnectUsingSettings();
        PhotonNetwork.GameVersion = "1.0";
    }

    if (!PlayerPrefs.HasKey("PlayerName"))
    {
        PhotonNetwork.NickName = "Player " + Random.Range(0, 1000).ToString("0000");
        PlayerPrefs.SetString("PlayerName", PhotonNetwork.NickName);

        setUserNameButton.SetActive(true);
        changeUserNameButton.SetActive(false);
        welcomeContainer.SetActive(false);
    }
    else
    {
        PhotonNetwork.NickName = PlayerPrefs.GetString("PlayerName");

        changeUserNameButton.SetActive(true);
        setUserNameButton.SetActive(false);

        welcomeContainer.SetActive(true);
        welcomeUser = PlayerPrefs.GetString("PlayerName");
        welcomeText.text = "Welcome " + welcomeUser;
    }

    fullRoomList = new List<RoomInfo>();
}
```

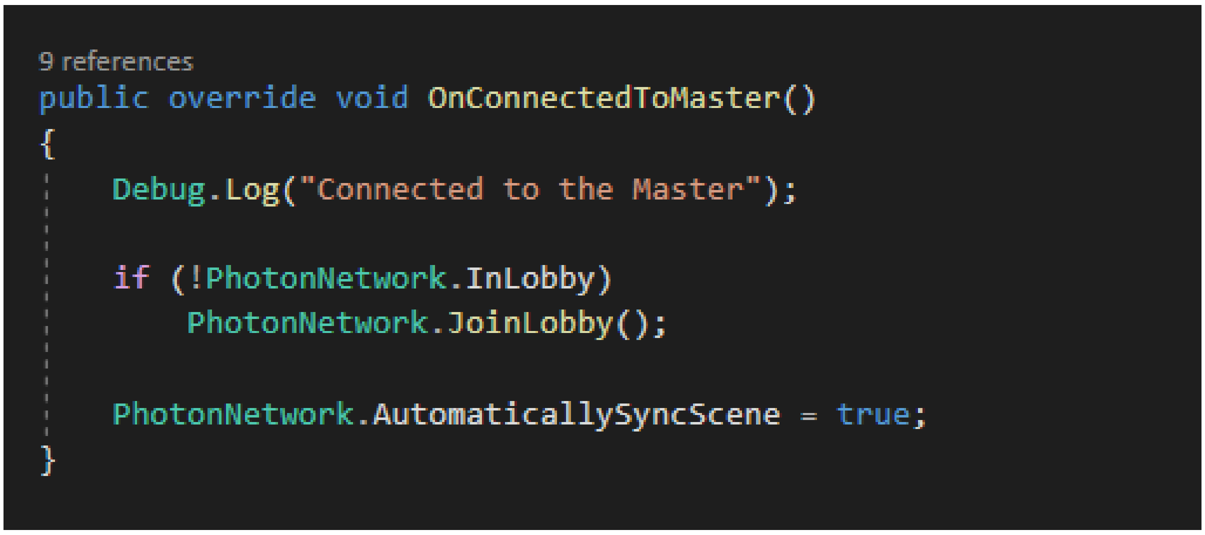
Fig 4.2.2: Initial connection to the server – Debug statements



```
[14:57:12] Connecting to the Master
UnityEngine.Debug:Log(Object)

Connecting to the Master
UnityEngine.Debug:Log(Object)
Launcher:Start() (at Assets/Scripts/Launcher.cs:63)
```

- Connection call-back from the server

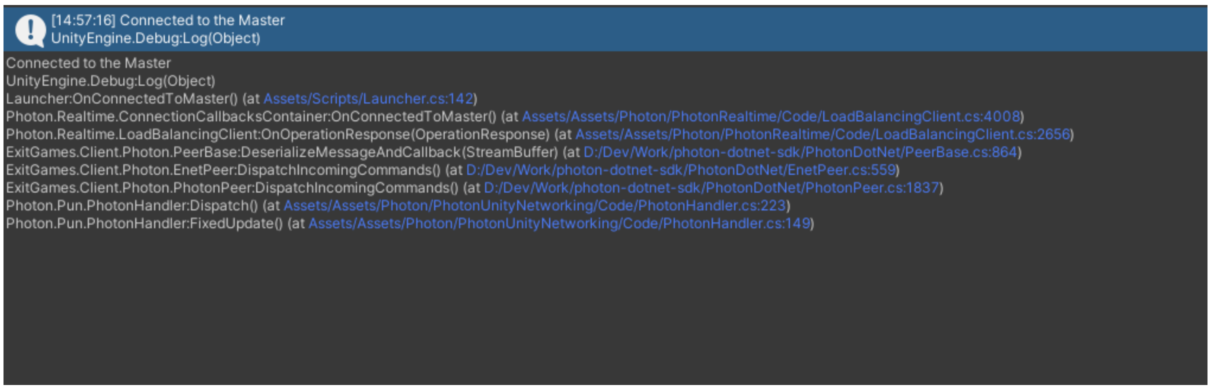
Fig 4.2.3: Connection call-back from the server – c# code



```
9 references
public override void OnConnectedToMaster()
{
    Debug.Log("Connected to the Master");

    if (!PhotonNetwork.InLobby)
        PhotonNetwork.JoinLobby();

    PhotonNetwork.AutomaticallySyncScene = true;
}
```

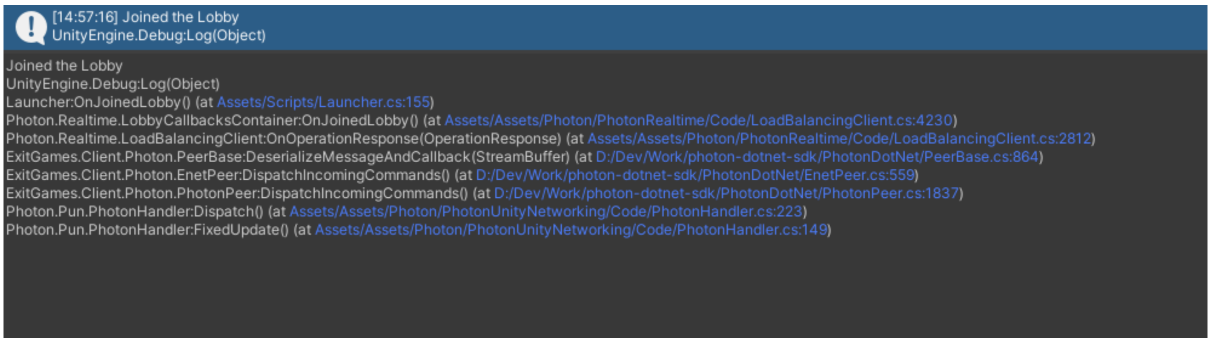
Debug Statement:

Fig 4.2.4: Connection call-back from the server – Debug Statement 1



```
[14:57:16] Connected to the Master
UnityEngine.Debug:Log(Object)
Connected to the Master
UnityEngine.Debug:Log(Object)
Launcher:OnConnectedToMaster() (at Assets/Scripts/Launcher.cs:142)
Photon.Realtime.ConnectionCallbacksContainer:OnConnectedToMaster() (at Assets/Assets/Photon/PhotonRealtime/Code/LoadBalancingClient.cs:4008)
Photon.Realtime.LoadBalancingClient:OnOperationResponse(OperationResponse) (at Assets/Assets/Photon/PhotonRealtime/Code/LoadBalancingClient.cs:2656)
ExitGames.Client.Photon.PeerBase:DeserializeMessageAndCallback(StreamBuffer) (at D:/Dev/Work/photon-dotnet-sdk/PhotonDotNet/PeerBase.cs:864)
ExitGames.Client.Photon.EnetPeer:DispatchIncomingCommands() (at D:/Dev/Work/photon-dotnet-sdk/PhotonDotNet/EnetPeer.cs:559)
ExitGames.Client.Photon.PhotonPeer:DispatchIncomingCommands() (at D:/Dev/Work/photon-dotnet-sdk/PhotonDotNet/PhotonPeer.cs:1837)
Photon.Pun.PhotonHandler:Dispatch() (at Assets/Assets/Photon/PhotonUnityNetworking/Code/PhotonHandler.cs:223)
Photon.Pun.PhotonHandler:FixedUpdate() (at Assets/Assets/Photon/PhotonUnityNetworking/Code/PhotonHandler.cs:149)
```

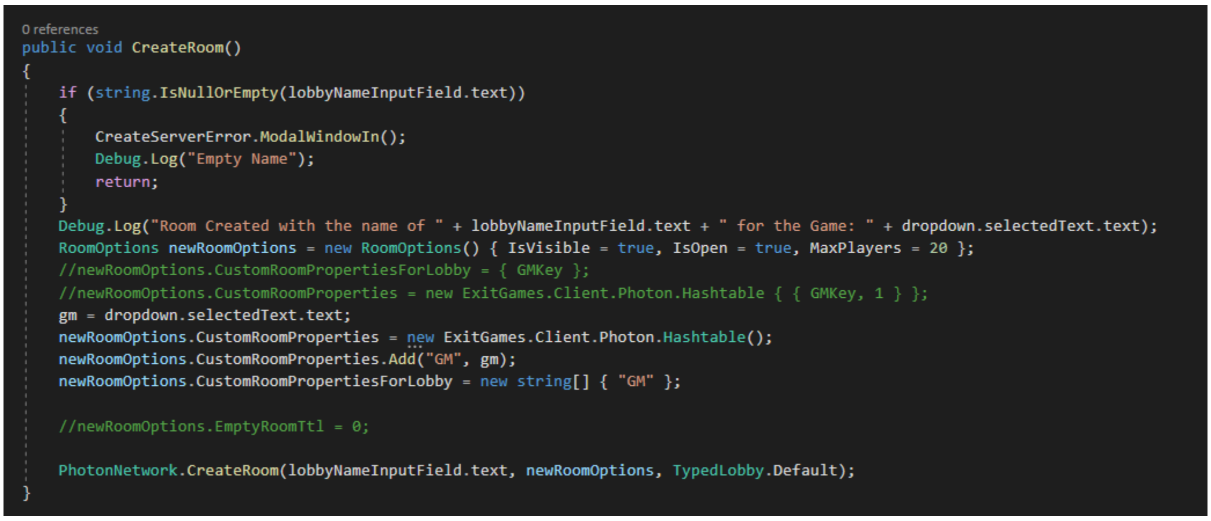
Fig 4.2.5: Connection call-back from the server – Debug Statements 2



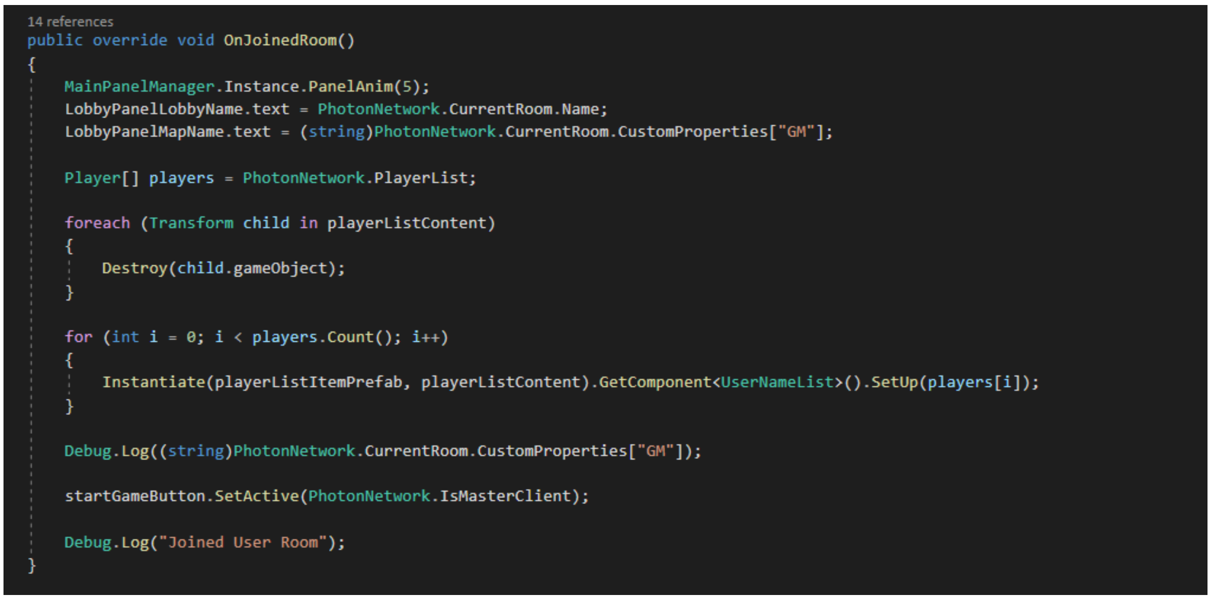
```
[14:57:16] Joined the Lobby
UnityEngine.Debug:Log(Object)
Joined the Lobby
UnityEngine.Debug:Log(Object)
Launcher:OnJoinedLobby() (at Assets/Scripts/Launcher.cs:155)
Photon.Realtime.LobbyCallbacksContainer:OnJoinedLobby() (at Assets/Assets/Photon/PhotonRealtime/Code/LoadBalancingClient.cs:4230)
Photon.Realtime.LoadBalancingClient:OnOperationResponse(OperationResponse) (at Assets/Assets/Photon/PhotonRealtime/Code/LoadBalancingClient.cs:2812)
ExitGames.Client.Photon.PeerBase:DeserializeMessageAndCallback(StreamBuffer) (at D:/Dev/Work/photon-dotnet-sdk/PhotonDotNet/PeerBase.cs:864)
ExitGames.Client.Photon.EnetPeer:DispatchIncomingCommands() (at D:/Dev/Work/photon-dotnet-sdk/PhotonDotNet/EnetPeer.cs:559)
ExitGames.Client.Photon.PhotonPeer:DispatchIncomingCommands() (at D:/Dev/Work/photon-dotnet-sdk/PhotonDotNet/PhotonPeer.cs:1837)
Photon.Pun.PhotonHandler:Dispatch() (at Assets/Assets/Photon/PhotonUnityNetworking/Code/PhotonHandler.cs:223)
Photon.Pun.PhotonHandler:FixedUpdate() (at Assets/Assets/Photon/PhotonUnityNetworking/Code/PhotonHandler.cs:149)
```

- Initial creation/joining of a room

Fig 4.2.6: Initial creation/joining of a room – c# code

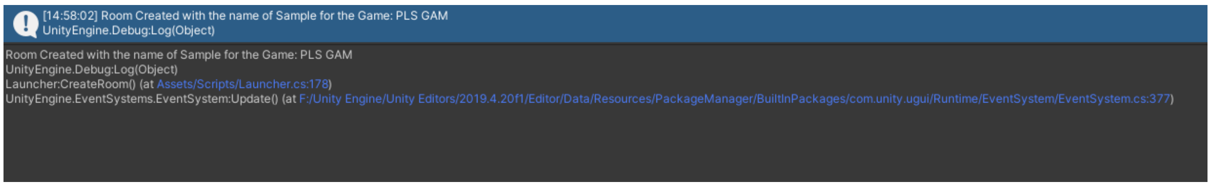


```
0 references
public void CreateRoom()
{
    if (string.IsNullOrEmpty(lobbyNameInputField.text))
    {
        CreateServerError.ModalWindowIn();
        Debug.Log("Empty Name");
        return;
    }
    Debug.Log("Room Created with the name of " + lobbyNameInputField.text + " for the Game: " + dropdown.selectedText.text);
    RoomOptions newRoomOptions = new RoomOptions() { IsVisible = true, IsOpen = true, MaxPlayers = 20 };
    //newRoomOptions.CustomRoomPropertiesForLobby = { GMKey };
    //newRoomOptions.CustomRoomProperties = new ExitGames.Client.Photon.Hashtable { { GMKey, 1 } };
    gm = dropdown.selectedText.text;
    newRoomOptions.CustomRoomProperties = new ExitGames.Client.Photon.Hashtable();
    newRoomOptions.CustomRoomProperties.Add("GM", gm);
    newRoomOptions.CustomRoomPropertiesForLobby = new string[] { "GM" };

    //newRoomOptions.EmptyRoomTtl = 0;

    PhotonNetwork.CreateRoom(lobbyNameInputField.text, newRoomOptions, TypedLobby.Default);
}
```

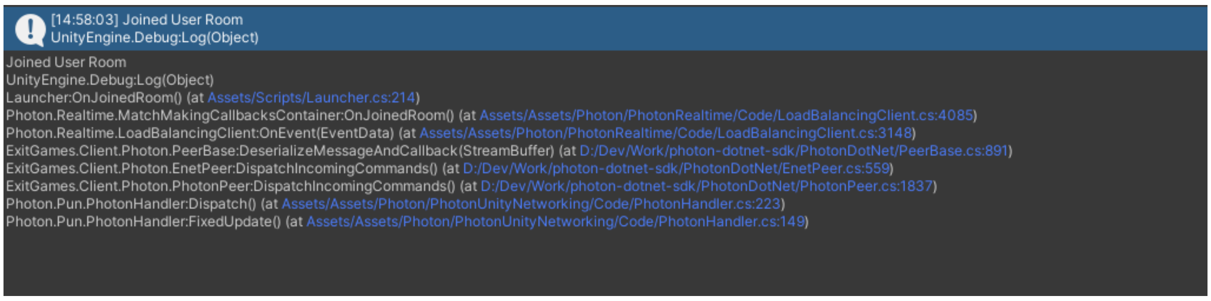


```
14 references
public override void OnJoinedRoom()
{
    MainPanelManager.Instance.PanelAnim(5);
    LobbyPanelLobbyName.text = PhotonNetwork.CurrentRoom.Name;
    LobbyPanelMapName.text = (string)PhotonNetwork.CurrentRoom.CustomProperties["GM"];

    Player[] players = PhotonNetwork.PlayerList;

    foreach (Transform child in playerListContent)
    {
        Destroy(child.gameObject);
    }

    for (int i = 0; i < players.Count(); i++)
    {
        Instantiate(playerListItemPrefab, playerListContent).GetComponent<UserNameList>().SetUp(players[i]);
    }

    Debug.Log((string)PhotonNetwork.CurrentRoom.CustomProperties["GM"]);

    startGameButton.SetActive(PhotonNetwork.IsMasterClient);

    Debug.Log("Joined User Room");
}
```

Fig 4.2.7: Initial creation/joining of a room – Debug Statements



```
[14:58:02] Room Created with the name of Sample for the Game: PLS GAM
UnityEngine.Debug:Log(Object)

Room Created with the name of Sample for the Game: PLS GAM
UnityEngine.Debug:Log(Object)
Launcher:CreateRoom() (at Assets/Scripts/Launcher.cs:178)
UnityEngine.EventSystems.EventSystem:Update() (at F:/Unity Engine/Unity Editors/2019.4.20f1/Editor/Data/Resources/PackageManager/BuiltInPackages/com.unity.ugui/Runtime/EventSystem/EventSystem.cs:377)
```

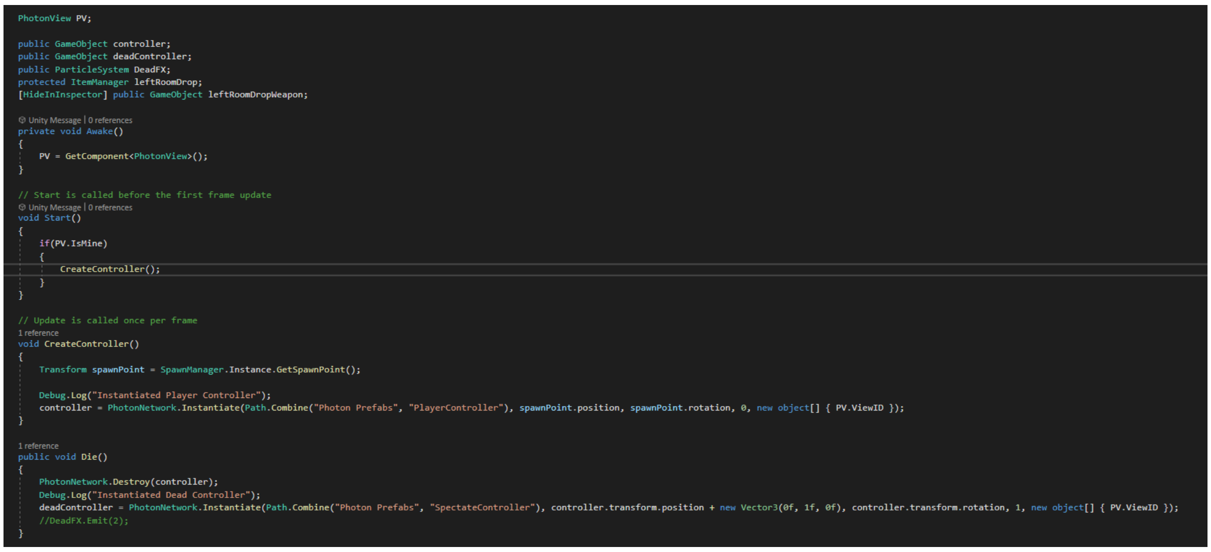


```
[14:58:03] Joined User Room
UnityEngine.Debug:Log(Object)

Joined User Room
UnityEngine.Debug:Log(Object)
Launcher:OnJoinedRoom() (at Assets/Scripts/Launcher.cs:214)
Photon.Realtime.MatchMakingCallbacksContainer:OnJoinedRoom() (at Assets/Assets/Photon/PhotonRealtime/Code/LoadBalancingClient.cs:4085)
Photon.Realtime.LoadBalancingClient:OnEvent(EventData) (at Assets/Assets/Photon/PhotonRealtime/Code/LoadBalancingClient.cs:3148)
ExitGames.Client.Photon.PeerBase:DeserializeMessageAndCallback(StreamBuffer) (at D:/Dev/Work/photon-dotnet-sdk/PhotonDotNet/PeerBase.cs:891)
ExitGames.Client.Photon.EnetPeer:DispatchIncomingCommands() (at D:/Dev/Work/photon-dotnet-sdk/PhotonDotNet/EnetPeer.cs:559)
ExitGames.Client.Photon.PhotonPeer:DispatchIncomingCommands() (at D:/Dev/Work/photon-dotnet-sdk/PhotonDotNet/PhotonPeer.cs:1837)
Photon.Pun.PhotonHandler:Dispatch() (at Assets/Assets/Photon/PhotonUnityNetworking/Code/PhotonHandler.cs:223)
Photon.Pun.PhotonHandler:FixedUpdate() (at Assets/Assets/Photon/PhotonUnityNetworking/Code/PhotonHandler.cs:149)
```

- Player Controller and weapons Instantiation

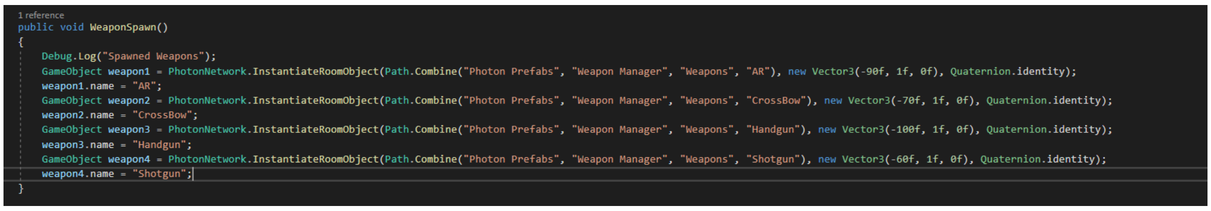
C# Code:

Fig 4.2.8: Player Controller



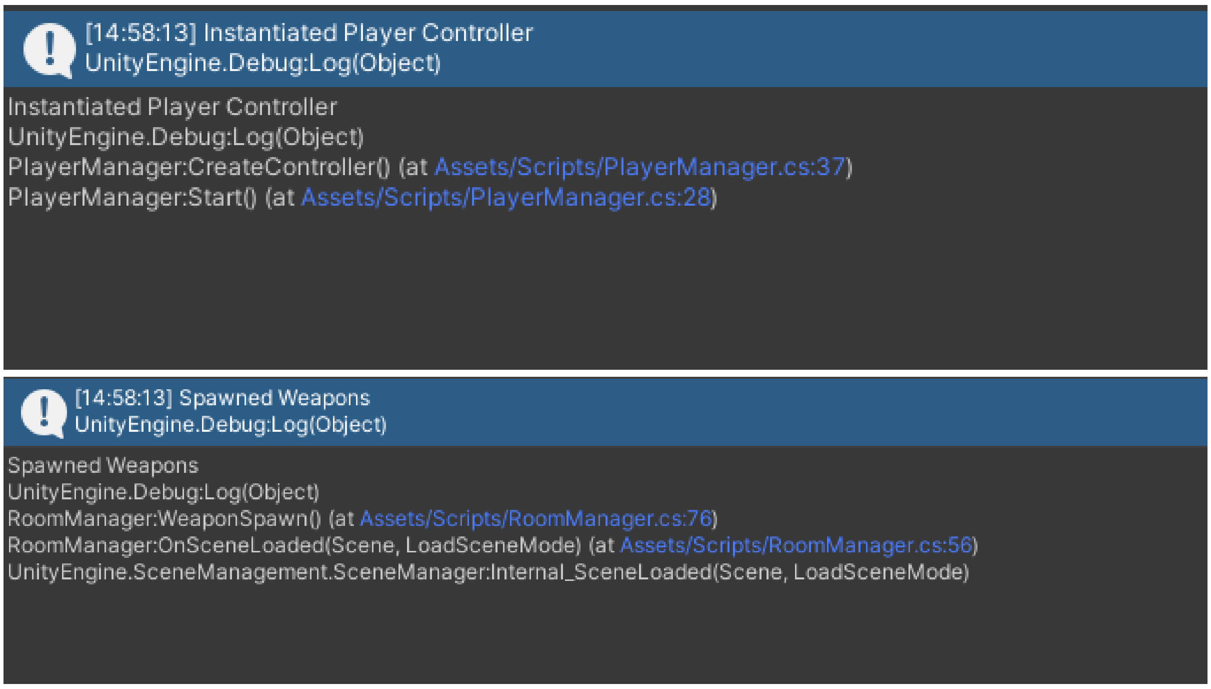
```
PhotonView PV;

public GameObject controller;
public GameObject deadController;
public ParticleSystem DeadFX;
protected ItemManager leftRoomDrop;
[HideInInspector] public GameObject leftRoomDropWeapon;

Unity Message | 0 references
private void Awake()
{
    PV = GetComponent<PhotonView>();
}

// Start is called before the first frame update
Unity Message | 0 references
void Start()
{
    if(PV.IsMine)
    {
        CreateController();
    }
}

// Update is called once per frame
1 reference
void CreateController()
{
    Transform spawnPoint = SpawnManager.Instance.GetSpawnPoint();

    Debug.Log("Instantiated Player Controller");
    controller = PhotonNetwork.Instantiate(Path.Combine("Photon Prefabs", "PlayerController"), spawnPoint.position, spawnPoint.rotation, 0, new object[] { PV.ViewID });
}

1 reference
public void Die()
{
    PhotonNetwork.Destroy(controller);
    Debug.Log("Instantiated Dead Controller");
    deadController = PhotonNetwork.Instantiate(Path.Combine("Photon Prefabs", "SpectateController"), controller.transform.position + new Vector3(0f, 1f, 0f), controller.transform.rotation, 1, new object[] { PV.ViewID });
    //DeadFX.Emit(2);
}
```

Fig 4.2.9: Weapon Spawns



```
1 reference
public void WeaponSpawn()
{
    Debug.Log("Spawned Weapons");
    GameObject weapon1 = PhotonNetwork.InstantiateRoomObject(Path.Combine("Photon Prefabs", "Weapon Manager", "Weapons", "AR"), new Vector3(-90f, 1f, 0f), Quaternion.identity);
    weapon1.name = "AR";
    GameObject weapon2 = PhotonNetwork.InstantiateRoomObject(Path.Combine("Photon Prefabs", "Weapon Manager", "Weapons", "CrossBow"), new Vector3(-70f, 1f, 0f), Quaternion.identity);
    weapon2.name = "CrossBow";
    GameObject weapon3 = PhotonNetwork.InstantiateRoomObject(Path.Combine("Photon Prefabs", "Weapon Manager", "Weapons", "Handgun"), new Vector3(-100f, 1f, 0f), Quaternion.identity);
    weapon3.name = "Handgun";
    GameObject weapon4 = PhotonNetwork.InstantiateRoomObject(Path.Combine("Photon Prefabs", "Weapon Manager", "Weapons", "Shotgun"), new Vector3(-60f, 1f, 0f), Quaternion.identity);
    weapon4.name = "Shotgun";
}
```

Fig 4.2.10: Debug Statements for Player Controller and Weapon Spawns

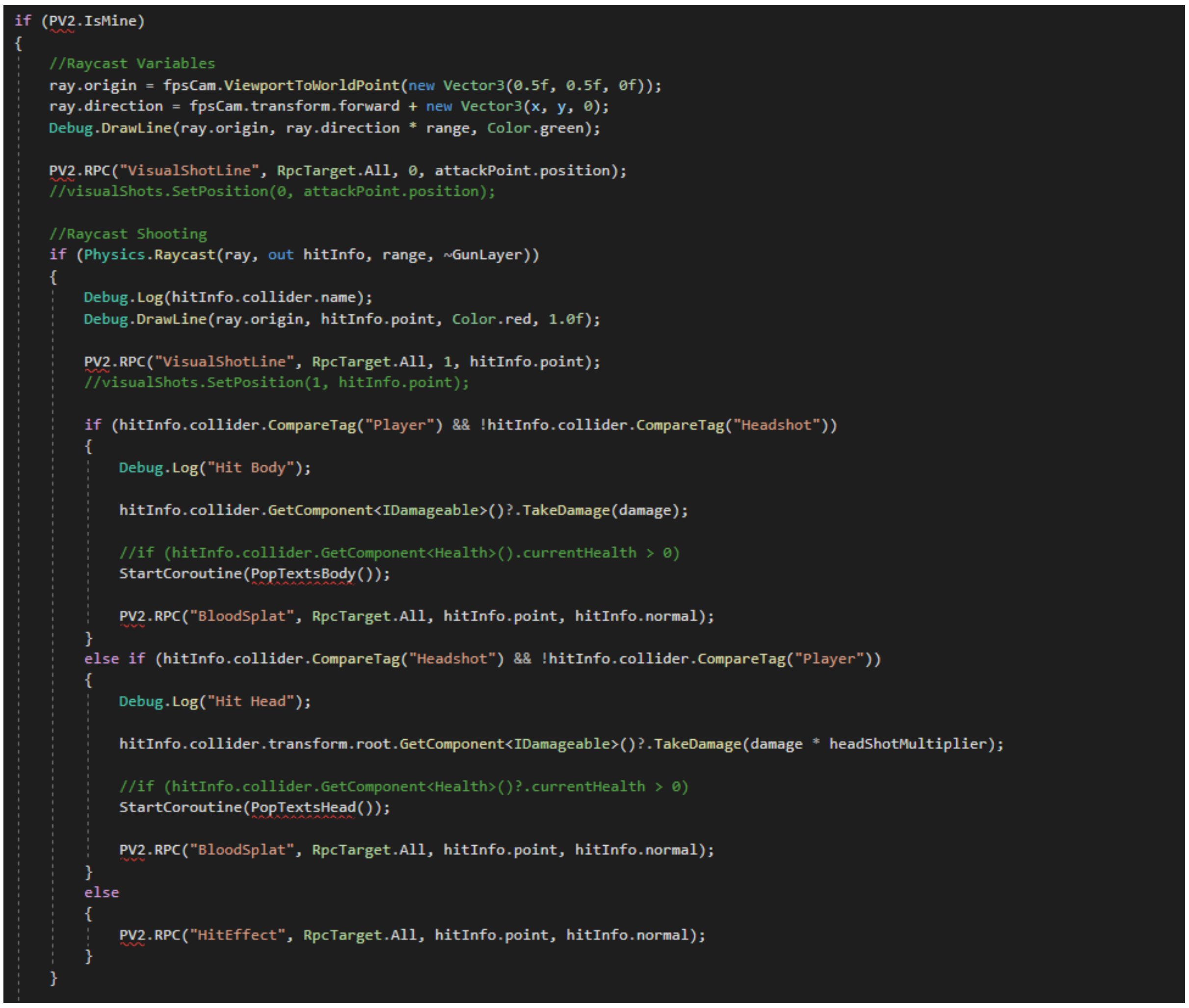


```
[14:58:13] Instantiated Player Controller
UnityEngine.Debug:Log(Object)

Instantiated Player Controller
UnityEngine.Debug:Log(Object)
PlayerManager:CreateController() (at Assets/Scripts/PlayerManager.cs:37)
PlayerManager:Start() (at Assets/Scripts/PlayerManager.cs:28)
```

```
[14:58:13] Spawned Weapons
UnityEngine.Debug:Log(Object)

Spawned Weapons
UnityEngine.Debug:Log(Object)
RoomManager:WeaponSpawn() (at Assets/Scripts/RoomManager.cs:76)
RoomManager:OnSceneLoaded(Scene, LoadSceneMode) (at Assets/Scripts/RoomManager.cs:56)
UnityEngine.SceneManagement.SceneManager:Internal_SceneLoaded(Scene, LoadSceneMode)
```

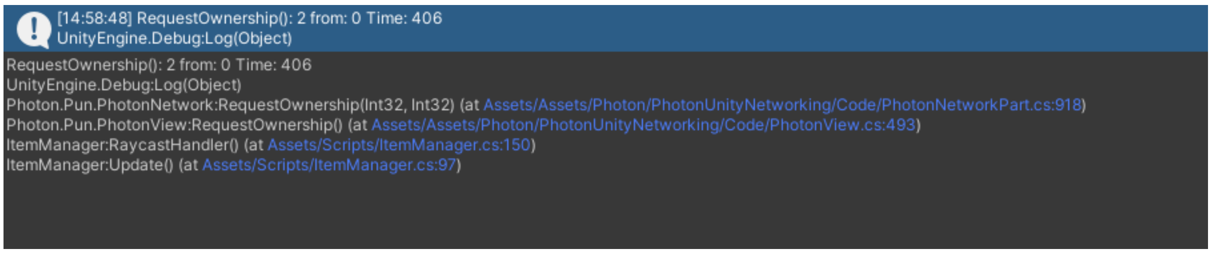
- Shooting mechanism using Raycast

Fig 4.2.11: Shooting mechanism using Raycast – c# code



```
if (PV2.IsMine)
{
    //Raycast Variables
    ray.origin = fpsCam.ViewportToWorldPoint(new Vector3(0.5f, 0.5f, 0f));
    ray.direction = fpsCam.transform.forward + new Vector3(x, y, 0);
    Debug.DrawLine(ray.origin, ray.direction * range, Color.green);

    PV2.RPC("VisualShotLine", RpcTarget.All, 0, attackPoint.position);
    //visualShots.SetPosition(0, attackPoint.position);

    //Raycast Shooting
    if (Physics.Raycast(ray, out hitInfo, range, ~GunLayer))
    {
        Debug.Log(hitInfo.collider.name);
        Debug.DrawLine(ray.origin, hitInfo.point, Color.red, 1.0f);

        PV2.RPC("VisualShotLine", RpcTarget.All, 1, hitInfo.point);
        //visualShots.SetPosition(1, hitInfo.point);

        if (hitInfo.collider.CompareTag("Player") && !hitInfo.collider.CompareTag("Headshot"))
        {
            Debug.Log("Hit Body");

            hitInfo.collider.GetComponent<IDamageable>()?.TakeDamage(damage);

            //if (hitInfo.collider.GetComponent<Health>().currentHealth > 0)
            StartCoroutine(PopTextsBody());

            PV2.RPC("BloodSplat", RpcTarget.All, hitInfo.point, hitInfo.normal);
        }
        else if (hitInfo.collider.CompareTag("Headshot") && !hitInfo.collider.CompareTag("Player"))
        {
            Debug.Log("Hit Head");

            hitInfo.collider.transform.root.GetComponent<IDamageable>()?.TakeDamage(damage * headShotMultiplier);

            //if (hitInfo.collider.GetComponent<Health>()?.currentHealth > 0)
            StartCoroutine(PopTextsHead());

            PV2.RPC("BloodSplat", RpcTarget.All, hitInfo.point, hitInfo.normal);
        }
        else
        {
            PV2.RPC("HitEffect", RpcTarget.All, hitInfo.point, hitInfo.normal);
        }
    }
```

Fig 4.2.12: Shooting mechanism using Raycast – Debug Statement

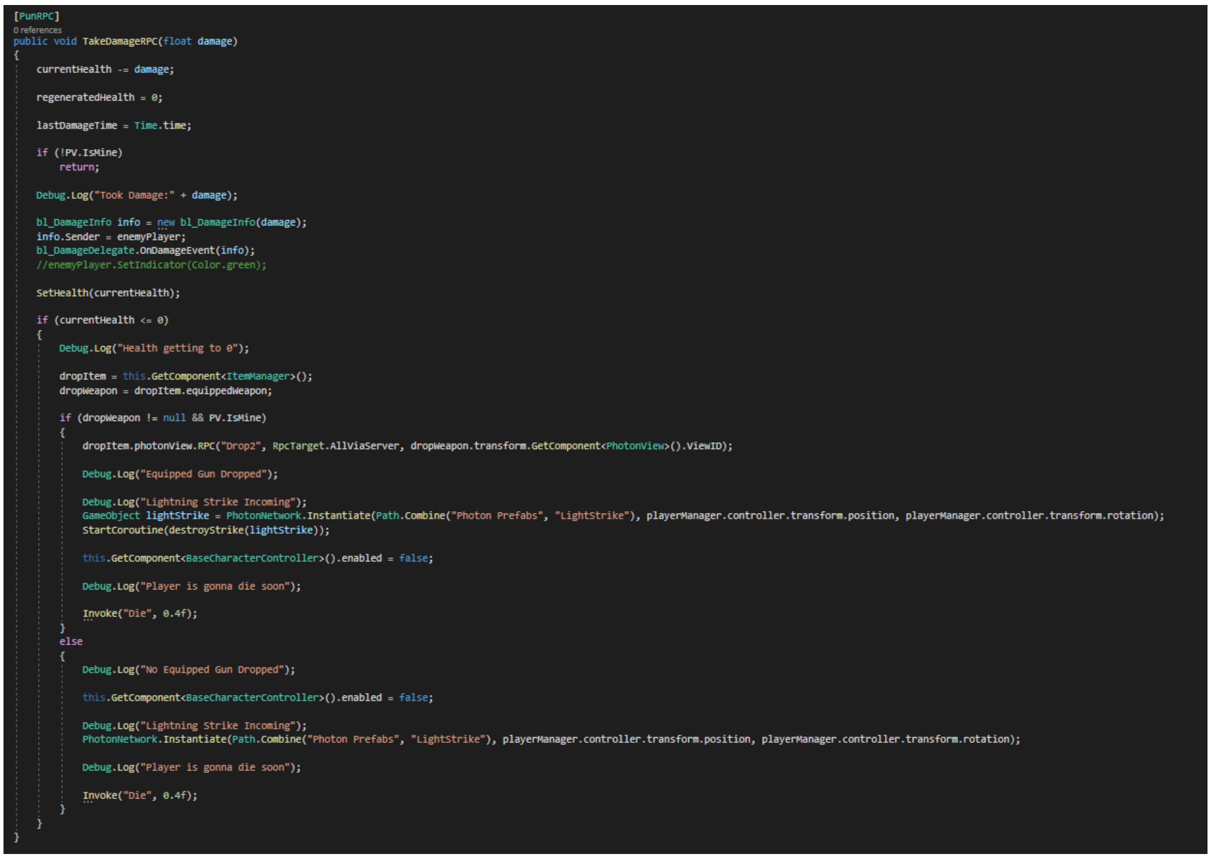


```
[14:58:48] RequestOwnership(): 2 from: 0 Time: 406
UnityEngine.Debug:Log(Object)
RequestOwnership(): 2 from: 0 Time: 406
UnityEngine.Debug:Log(Object)
Photon.Pun.PhotonNetwork:RequestOwnership(Int32, Int32) (at Assets/Assets/Photon/PhotonUnityNetworking/Code/PhotonNetworkPart.cs:918)
Photon.Pun.PhotonView:RequestOwnership() (at Assets/Assets/Photon/PhotonUnityNetworking/Code/PhotonView.cs:493)
ItemManager:RaycastHandler() (at Assets/Scripts/ItemManager.cs:150)
ItemManager:Update() (at Assets/Scripts/ItemManager.cs:97)
```

- Health System using RPC and Network Connectivity

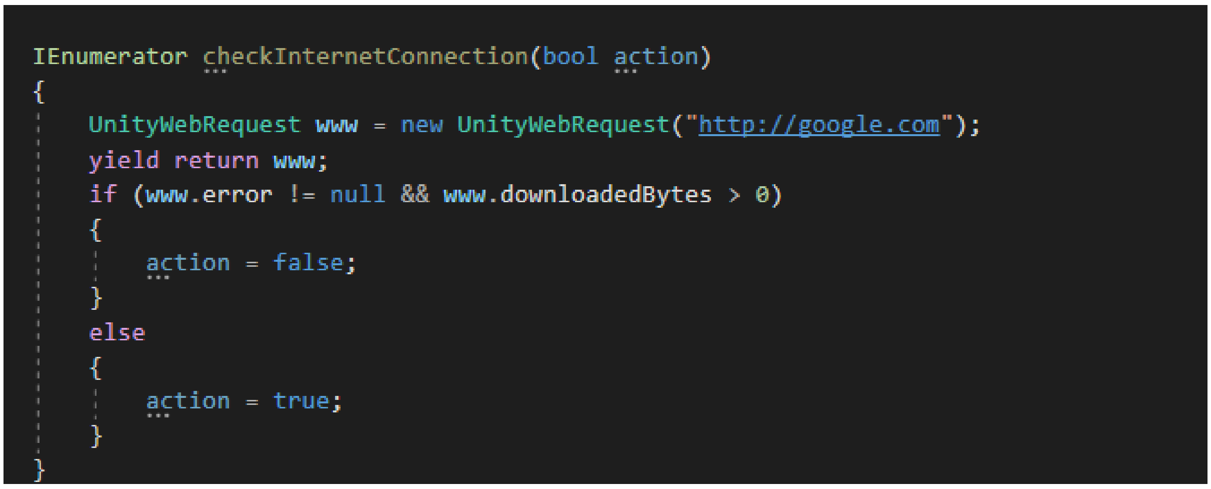
C# Code:

Fig 4.2.13: Health System – c# code



```
[PunRPC]
0 references
public void TakeDamageRPC(float damage)
{
    currentHealth -= damage;

    regeneratedHealth = 0;

    lastDamageTime = Time.time;

    if (!PV.IsMine)
        return;

    Debug.Log("Took Damage:" + damage);

    bl_DamageInfo info = new bl_DamageInfo(damage);
    info.Sender = enemyPlayer;
    bl_DamageDelegate.OnDamageEvent(info);
    //enemyPlayer.SetIndicator(Color.green);

    SetHealth(currentHealth);

    if (currentHealth <= 0)
    {
        Debug.Log("Health getting to 0");

        dropItem = this.GetComponent<ItemManager>();
        dropWeapon = dropItem.equippedWeapon;

        if (dropWeapon != null && PV.IsMine)
        {
            dropItem.photonView.RPC("Drop2", RpcTarget.AllViaServer, dropWeapon.transform.GetComponent<PhotonView>().ViewID);

            Debug.Log("Equipped Gun Dropped");

            Debug.Log("Lightning Strike Incoming");
            GameObject lightStrike = PhotonNetwork.Instantiate(Path.Combine("Photon Prefabs", "LightStrike"), playerManager.controller.transform.position, playerManager.controller.transform.rotation);
            StartCoroutine(destroyStrike(lightStrike));

            this.GetComponent<BaseCharacterController>().enabled = false;

            Debug.Log("Player is gonna die soon");

            Invoke("Die", 0.4f);
        }
        else
        {
            Debug.Log("No Equipped Gun Dropped");

            this.GetComponent<BaseCharacterController>().enabled = false;

            Debug.Log("Lightning Strike Incoming");
            PhotonNetwork.Instantiate(Path.Combine("Photon Prefabs", "LightStrike"), playerManager.controller.transform.position, playerManager.controller.transform.rotation);

            Debug.Log("Player is gonna die soon");

            Invoke("Die", 0.4f);
        }
    }
}
```

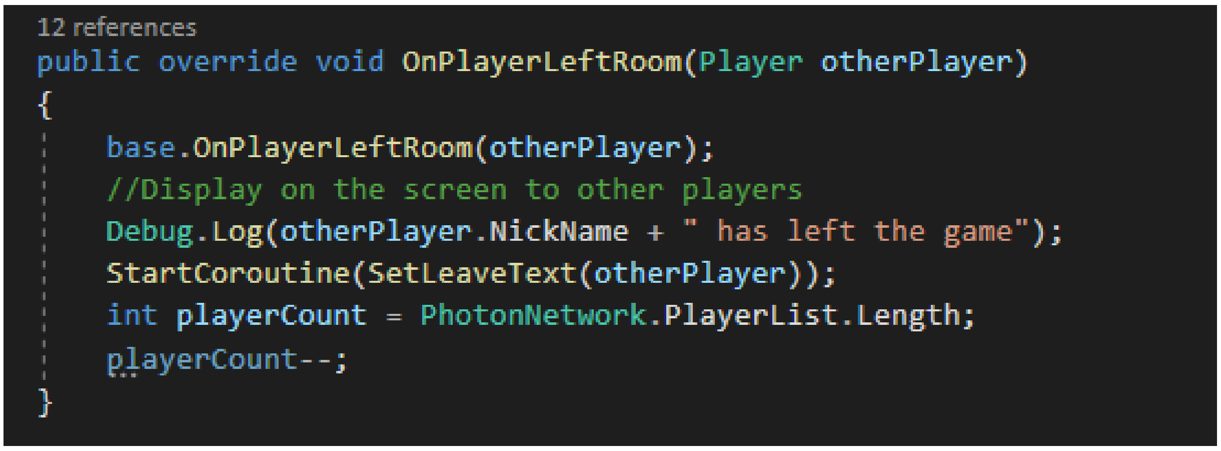
Fig 4.2.14: Network Connectivity – c# code



```
IEnumerator checkInternetConnection(bool action)
{
    UnityWebRequest www = new UnityWebRequest("http://google.com");
    yield return www;
    if (www.error != null && www.downloadedBytes > 0)
    {
        action = false;
    }
    else
    {
        action = true;
    }
}
```

- Leaving the hosted game and quitting the game

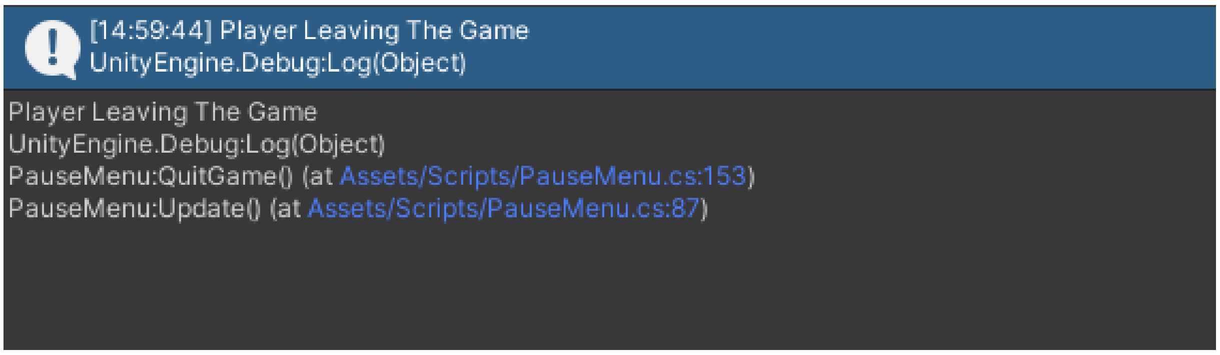
Fig 4.2.15: Leaving the hosted game and quitting the game – c# code



```
12 references
public override void OnPlayerLeftRoom(Player otherPlayer)
{
    base.OnPlayerLeftRoom(otherPlayer);
    //Display on the screen to other players
    Debug.Log(otherPlayer.NickName + " has left the game");
    StartCoroutine(SetLeaveText(otherPlayer));
    int playerCount = PhotonNetwork.PlayerList.Length;
    playerCount--;
}
```

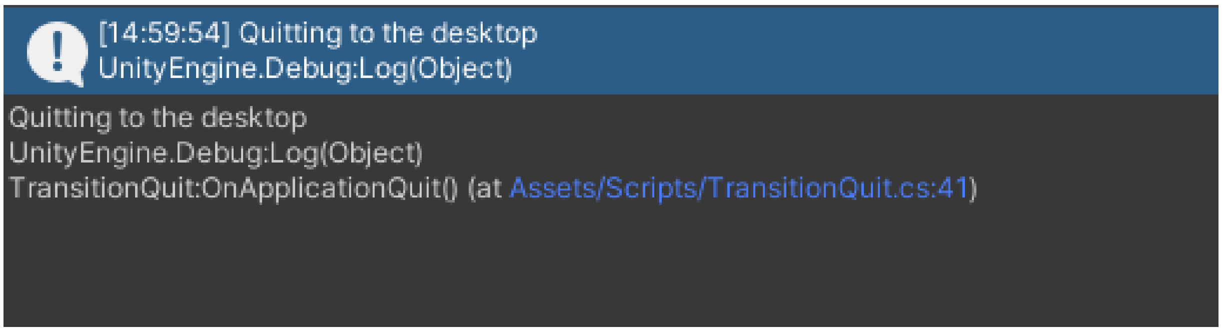
Debug Statement:

Fig 4.2.16: Leaving the game – Debug Statement



```
[14:59:44] Player Leaving The Game
UnityEngine.Debug:Log(Object)
Player Leaving The Game
UnityEngine.Debug:Log(Object)
PauseMenu:QuitGame() (at Assets/Scripts/PauseMenu.cs:153)
PauseMenu:Update() (at Assets/Scripts/PauseMenu.cs:87)
```

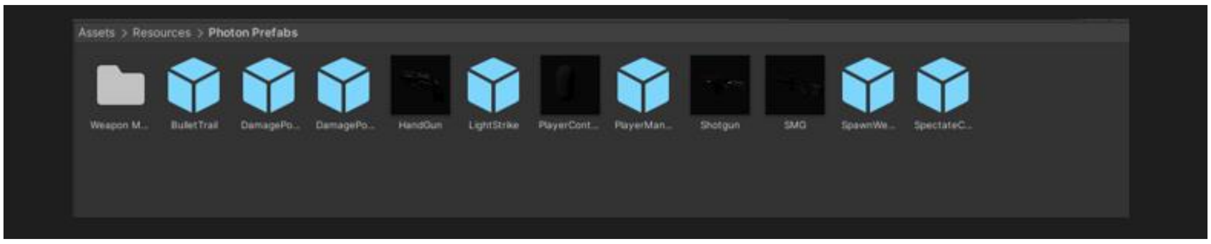
Fig 4.2.17: Quitting the game – Debug Statement



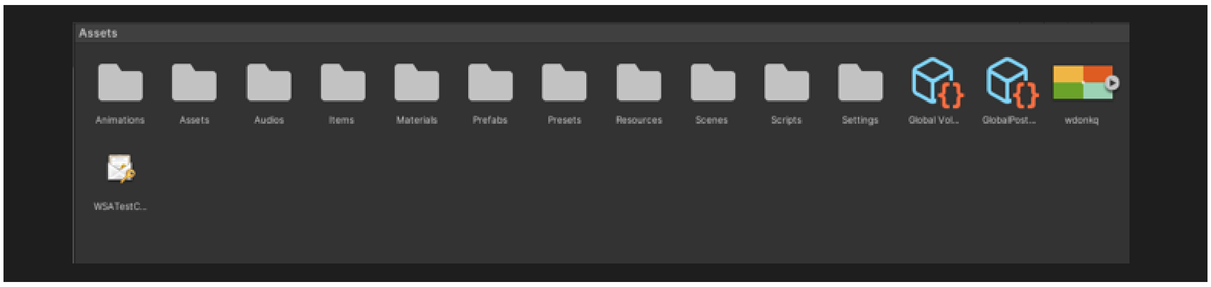
```
[14:59:54] Quitting to the desktop
UnityEngine.Debug:Log(Object)
Quitting to the desktop
UnityEngine.Debug:Log(Object)
TransitionQuit:OnApplicationQuit() (at Assets/Scripts/TransitionQuit.cs:41)
```

4.3 FOLDER STRUCTURE ANALYSIS

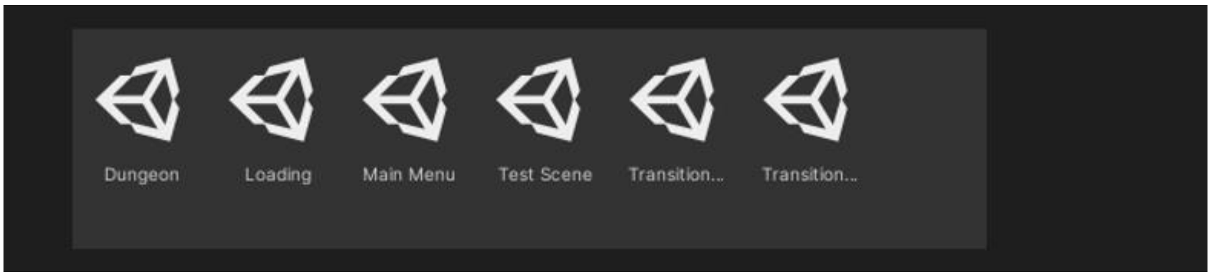
- Networked Folder, where all objects to be spawned on network is included here.
  Fig 4.3.1: Networked Folder



- Game Asset Folder
  Fig 4.3.2: Game Assets



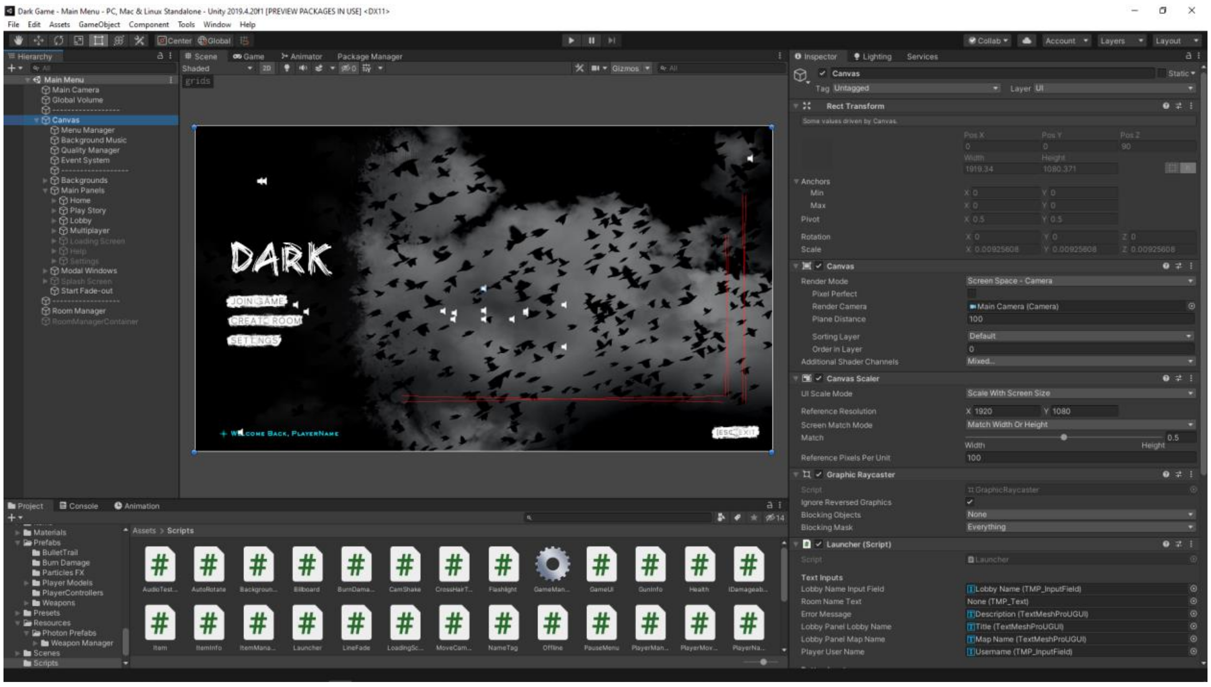
- Game Scenes
  Fig 4.3.3: Scenes used in the game.



4.4 IMPLEMENTATION DETAILS

a. Main Menu: The game should have a main menu where the player can decide what option they would like to invoke.

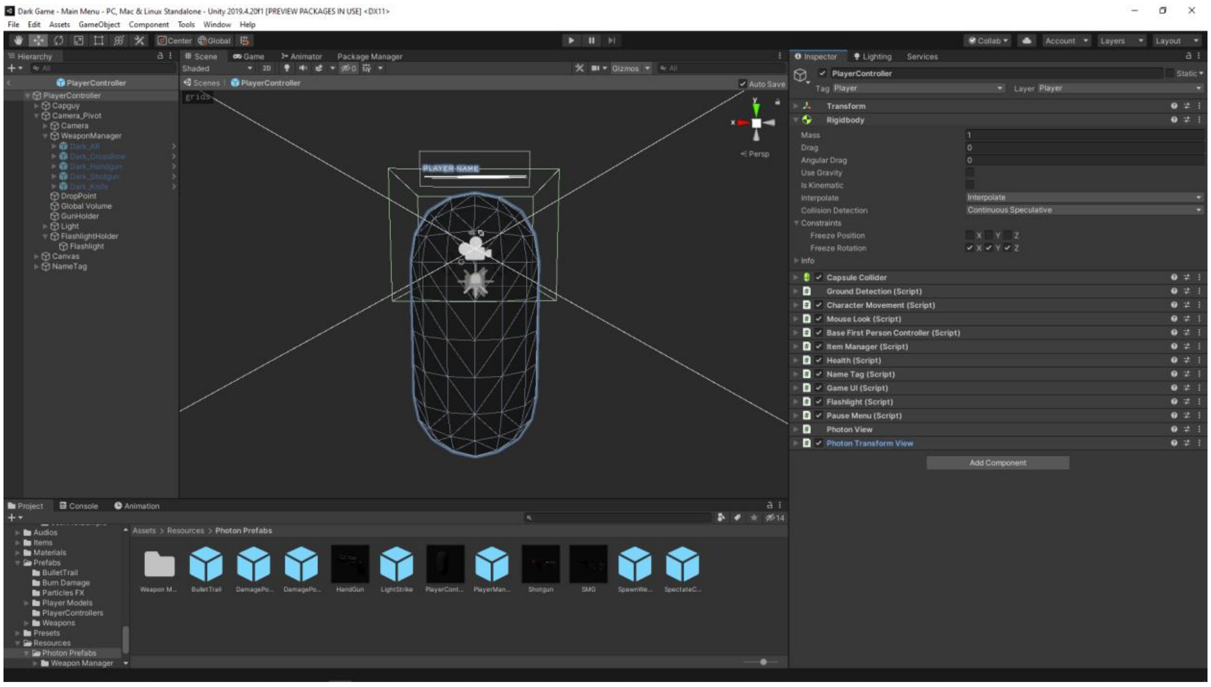
Fig 4.4.a.1: Main Menu



```
Dark Game - Main Menu - PC, Mac & Linux Standalone - Unity 2019.4.20f1 [PREVIEW PACKAGES IN USE] <DX11>

Inspector
Canvas                         Tag Untagged   Layer UI
Rect Transform
  Some values driven by Canvas.
  Pos X 0   Pos Y 0   Pos Z 90
  Width 1919.34   Height 1080.371
  Anchors  Min X 0 Y 0   Max X 0 Y 0
  Pivot X 0.5 Y 0.5
  Rotation X 0 Y 0 Z 0
  Scale X 0.00925608 Y 0.00925608 Z 0.00925608
Canvas
  Render Mode                 Screen Space - Camera
  Pixel Perfect
  Render Camera               Main Camera (Camera)
  Plane Distance              100
  Sorting Layer               Default
  Order in Layer              0
  Additional Shader Channels  Mixed...
Canvas Scaler
  UI Scale Mode               Scale With Screen Size
  Reference Resolution        X 1920  Y 1080
  Screen Match Mode           Match Width Or Height
  Match                       0.5 (Width – Height)
  Reference Pixels Per Unit   100
Graphic Raycaster
  Script                      GraphicRaycaster
  Ignore Reversed Graphics    ✓
  Blocking Objects            None
  Blocking Mask               Everything
Launcher (Script)
  Script                      Launcher
  Text Inputs
  Lobby Name Input Field      Lobby Name (TMP_InputField)
  Room Name Text              None (TMP_Text)
  Error Message               Description (TextMeshProUGUI)
  Lobby Panel Lobby Name      Title (TextMeshProUGUI)
  Lobby Panel Map Name        Map Name (TextMeshProUGUI)
  Player User Name            Username (TMP_InputField)
```

b. Characters: The game must have a set of characters who will either act as the Artificial intelligence or the player.

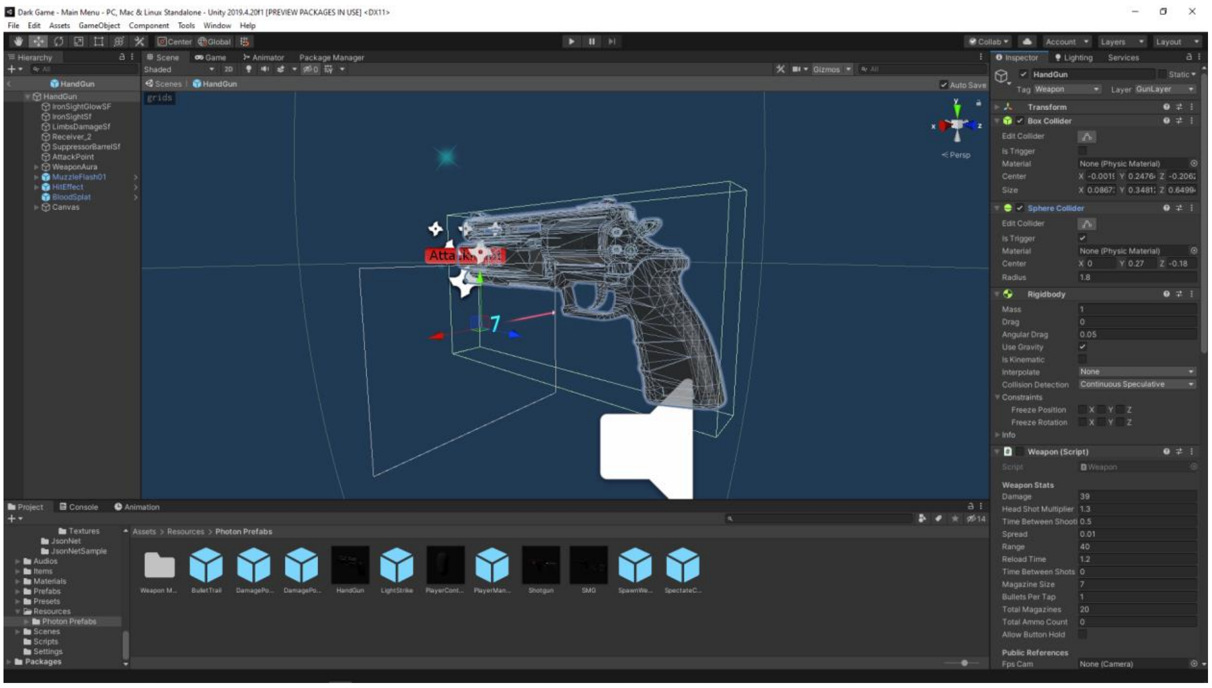
Fig 4.4.b.1: Character Screen



```
Dark Game - Main Menu - PC, Mac & Linux Standalone - Unity 2019.4.20f1 [PREVIEW PACKAGES IN USE] <DX11>

Inspector
PlayerController               Tag Player   Layer Player
Transform
Rigidbody
  Mass                  1
  Drag                  0
  Angular Drag          0
  Use Gravity
  Is Kinematic
  Interpolate           Interpolate
  Collision Detection   Continuous Speculative
  Constraints
    Freeze Position     X Y Z
    Freeze Rotation     ✓X ✓Y ✓Z
  Info
✓ Capsule Collider
  Ground Detection (Script)
✓ Character Movement (Script)
✓ Mouse Look (Script)
✓ Base First Person Controller (Script)
✓ Item Manager (Script)
✓ Health (Script)
✓ Name Tag (Script)
✓ Game UI (Script)
✓ Flashlight (Script)
✓ Pause Menu (Script)
  Photon View
✓ Photon Transform View
Add Component
```

c. Character Weapons: The player must be provided/Pickup Drop System with a weapon in order to kill and damage an enemy or other player.

Fig 4.4.c.1: Character Weapon Screen


Dark Game - Main Menu - PC, Mac & Linux Standalone - Unity 2019.4.20f1 [PREVIEW PACKAGES IN USE] <DX11>
File Edit Assets GameObject Component Tools Window Help
Hierarchy
Scene Game Animator Package Manager
Inspector Lighting Services
HandGun
IronSightGlowSF
IronSightSf
LimbsDamageSf
Receiver_2
SuppressorBarrelSf
AttackPoint
WeaponAura
MuzzleFlash01
HitEffect
BloodSplat
Canvas
Project Console Animation
Assets > Resources > Photon Prefabs
Weapon M... BulletTrail DamagePo... DamagePo... HandGun LightStrike PlayerCont... PlayerMan... Shotgun SMG SpawnWe... SpectateC...

HandGun
Tag Weapon Layer GunLayer

Box Collider
Material None (Physic Material)
Center X -0.0015 Y 0.2476 Z -0.2062
Size X 0.0867 Y 0.3481 Z 0.6499

Sphere Collider
Is Trigger ✓
Material None (Physic Material)
Center X 0 Y 0.27 Z -0.18
Radius 1.8

Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity ✓
Is Kinematic
Interpolate None
Collision Detection Continuous Speculative

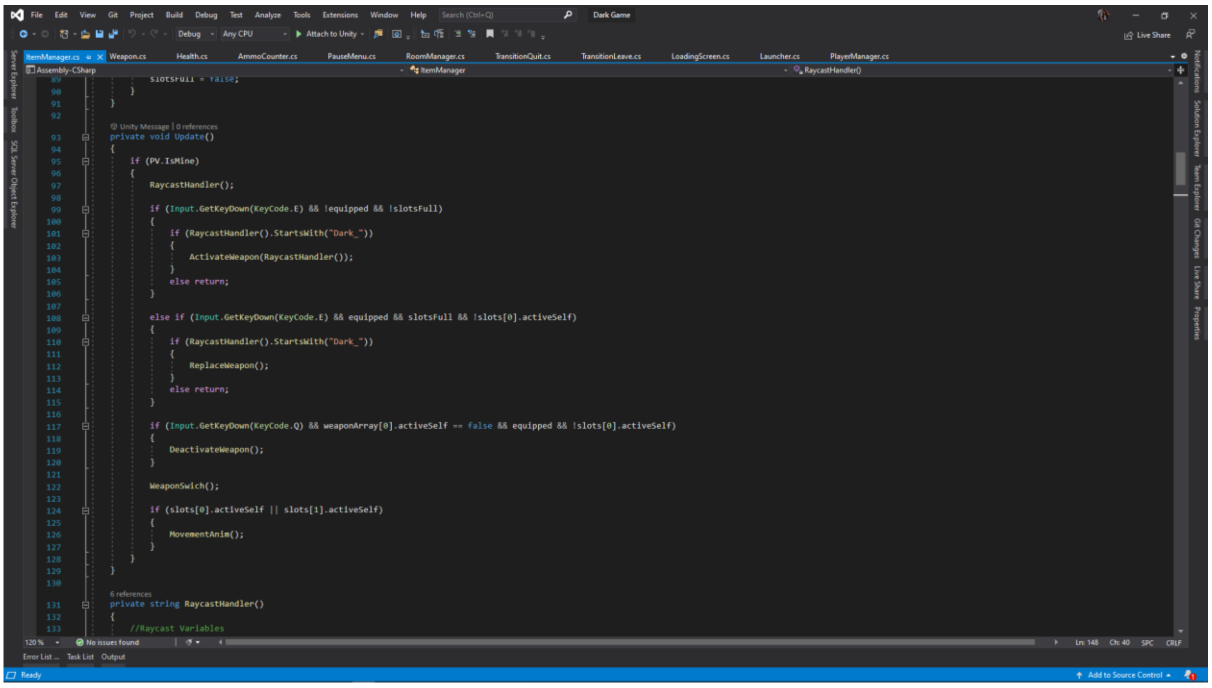
Weapon (Script)
Script Weapon

Weapon Stats
Damage 39
Head Shot Multiplier 1.3
Time Between Shooti 0.5
Spread 0.01
Range 40
Reload Time 1.2
Time Between Shots 0
Magazine Size 7
Bullets Per Tap 1
Total Magazines 20
Total Ammo Count 0
Allow Button Hold

Public References
Fps Cam None (Camera)

d. Interactable Objects: The player should be able interact with a few objects in the game. The player can interact with an object by using the E key. Code below.

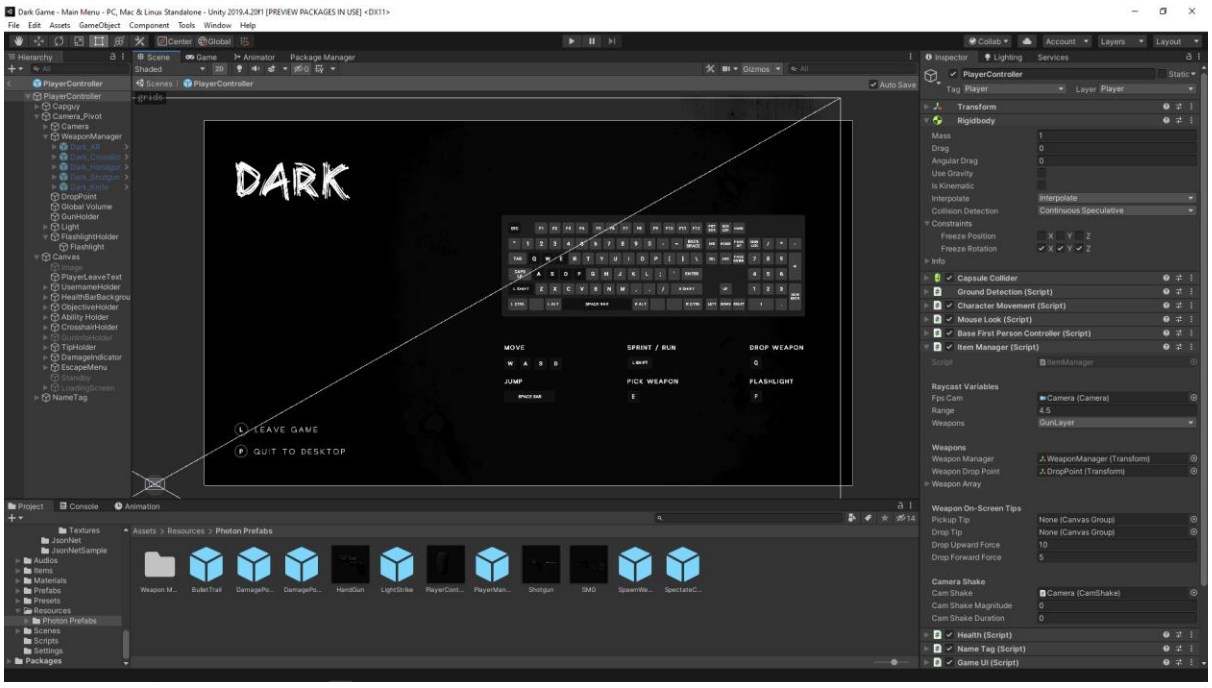
Fig 4.4.d.1: Interactable Objects


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Dark Game
ItemManager.cs Weapon.cs Health.cs AmmoCounter.cs PauseMenu.cs RoomManager.cs TransitionQuit.cs TransitionLeave.cs LoadingScreen.cs Launcher.cs PlayerManager.cs
Assembly-CSharp ItemManager RaycastHandler()

```csharp
                slotsFull = false;
            }
        }

        private void Update()
        {
            if (PV.IsMine)
            {
                RaycastHandler();

                if (Input.GetKeyDown(KeyCode.E) && !equipped && !slotsFull)
                {
                    if (RaycastHandler().StartsWith("Dark_"))
                    {
                        ActivateWeapon(RaycastHandler());
                    }
                    else return;
                }

                else if (Input.GetKeyDown(KeyCode.E) && equipped && slotsFull && !slots[0].activeSelf)
                {
                    if (RaycastHandler().StartsWith("Dark_"))
                    {
                        ReplaceWeapon();
                    }
                    else return;
                }

                if (Input.GetKeyDown(KeyCode.Q) && weaponArray[0].activeSelf == false && equipped && !slots[0].activeSelf)
                {
                    DeactivateWeapon();
                }

                WeaponSwich();

                if (slots[0].activeSelf || slots[1].activeSelf)
                {
                    MovementAnim();
                }
            }
        }

        private string RaycastHandler()
        {
            //Raycast Variables
```

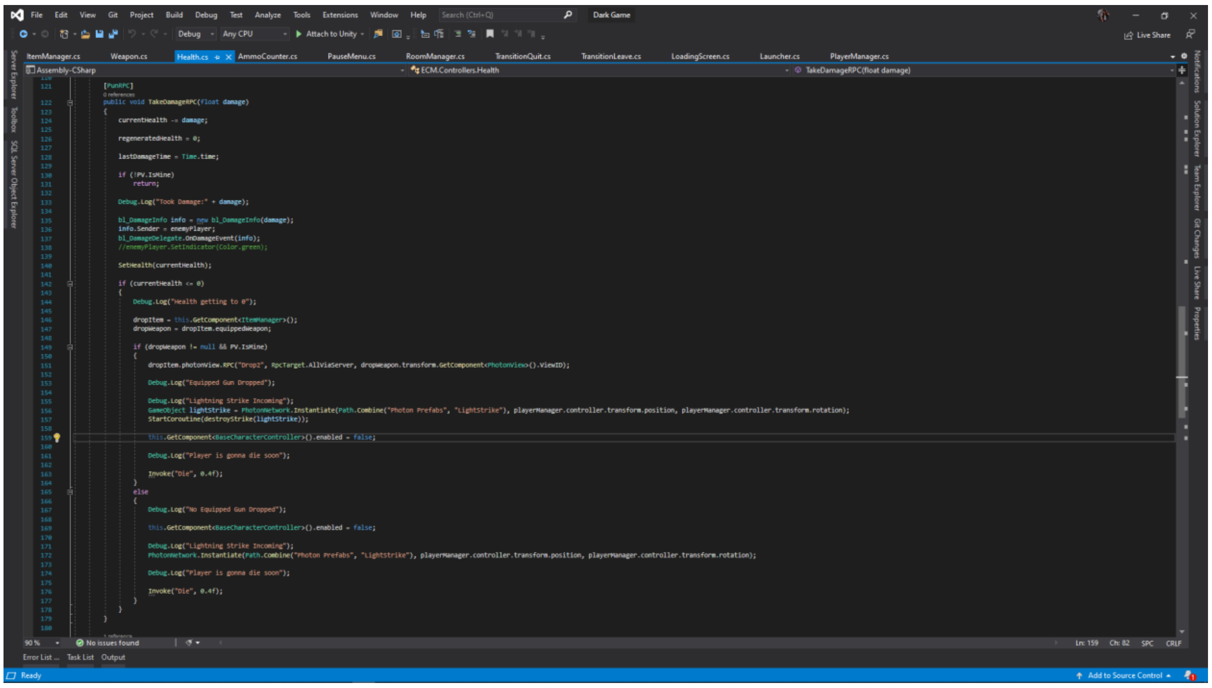
e. Pause Game/Quit Game: The player should be able to pause/Quit the game. The player will press the ESC button or press the pause button in the corner of the screen which will pause the game and give the player the option to continue, return to main menu or quit the game.

Fig 4.4.e.1: Pause/Quit Game


Dark Game - Main Menu - PC, Mac & Linux Standalone - Unity 2019.4.20f1 [PREVIEW PACKAGES IN USE] <DX11>
DARK
MOVE
SPRINT / RUN
DROP WEAPON
JUMP
PICK WEAPON
FLASHLIGHT
LEAVE GAME
QUIT TO DESKTOP

f. Health System: The player must be able to lose health when attacked by other players. When the player has run out of health, they will die.

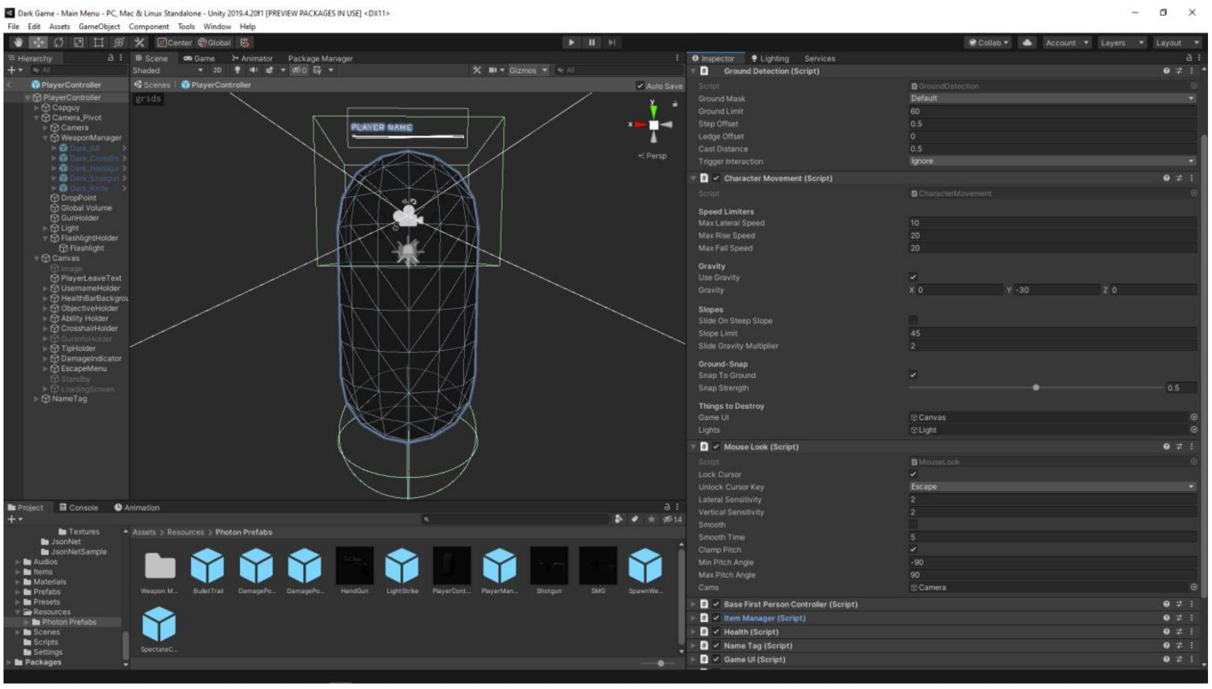
Fig 4.4.f.1: Health System



```
        [PunRPC]
        0 references
        public void TakeDamageRPC(float damage)
        {
            currentHealth -= damage;

            regeneratedHealth = 0;

            lastDamageTime = Time.time;

            if (!PV.IsMine)
                return;

            Debug.Log("Took Damage:" + damage);

            bl_DamageInfo info = new bl_DamageInfo(damage);
            info.Sender = enemyPlayer;
            bl_DamageDelegate.OnDamageEvent(info);
            //enemyPlayer.SetIndicator(Color.green);

            SetHealth(currentHealth);

            if (currentHealth <= 0)
            {
                Debug.Log("Health getting to 0");

                dropItem = this.GetComponent<ItemManager>();
                dropWeapon = dropItem.equippedWeapon;

                if (dropWeapon != null && PV.IsMine)
                {
                    dropItem.photonView.RPC("Drop2", RpcTarget.AllViaServer, dropWeapon.transform.GetComponent<PhotonView>().ViewID);

                    Debug.Log("Equipped Gun Dropped");

                    Debug.Log("Lightning Strike Incoming");
                    GameObject lightStrike = PhotonNetwork.Instantiate(Path.Combine("Photon Prefabs", "LightStrike"), playerManager.controller.transform.position, playerManager.controller.transform.rotation);
                    StartCoroutine(destroyStrike(lightStrike));

                    this.GetComponent<BaseCharacterController>().enabled = false;

                    Debug.Log("Player is gonna die soon");

                    Invoke("Die", 0.4f);
                }
                else
                {
                    Debug.Log("No Equipped Gun Dropped");

                    this.GetComponent<BaseCharacterController>().enabled = false;

                    Debug.Log("Lightning Strike Incoming");
                    PhotonNetwork.Instantiate(Path.Combine("Photon Prefabs", "LightStrike"), playerManager.controller.transform.position, playerManager.controller.transform.rotation);

                    Debug.Log("Player is gonna die soon");

                    Invoke("Die", 0.4f);
                }
            }
        }
```

g. Player Movement: The player should be able to move the controllable character. The character can be moved using the movement keys and they can jump, crouch and shoot.

Fig 4.4.g.1: Player Movement


Dark Game - Main Menu - PC, Mac & Linux Standalone - Unity 2019.4.20f1 [PREVIEW PACKAGES IN USE] <DX11>
File Edit Assets GameObject Component Tools Window Help
Hierarchy
Scene
Game
Animator
Package Manager
Inspector
Lighting
Services
PlayerController
PLAYER NAME
Persp
Project
Console
Animation
Assets > Resources > Photon Prefabs

```
Ground Detection (Script)
Script                    GroundDetection
Ground Mask               Default
Ground Limit              60
Step Offset               0.5
Ledge Offset              0
Cast Distance             0.5
Trigger Interaction       Ignore

Character Movement (Script)
Script                    CharacterMovement
Speed Limiters
Max Lateral Speed         10
Max Rise Speed            20
Max Fall Speed            20
Gravity
Use Gravity               ✓
Gravity                   X 0    Y -30    Z 0
Slopes
Slide On Steep Slope
Slope Limit               45
Slide Gravity Multiplier  2
Ground-Snap
Snap To Ground            ✓
Snap Strength             0.5
Things to Destroy
Game UI                   Canvas
Lights                    Light

Mouse Look (Script)
Script                    MouseLook
Lock Cursor               ✓
Unlock Cursor Key         Escape
Lateral Sensitivity       2
Vertical Sensitivity      2
Smooth
Smooth Time               5
Clamp Pitch               ✓
Min Pitch Angle           -90
Max Pitch Angle           90
Cams                      Camera

Base First Person Controller (Script)
Item Manager (Script)
Health (Script)
Name Tag (Script)
Game UI (Script)
```

CHAPTER 5
CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

In my opinion there are many advantages to creating this type of project. It is a chance to do something different, step outside my comfort zone and learn how games are created. I liked that I had full freedom with the creativity of the game, and I could implement it with my vision in mind. Another advantage was the experience of coding in C# in previous college modules. This was a benefit to me as this is the language used to code in Unity. I was pleasantly surprised with how much coding experience I managed to pick up. As someone who always struggled a bit with coding, I tried to challenge myself by fixing errors without online help. I was able to fix these errors and after a while I was able to write whole scripts without help. This is something I did not expect to happen, and it has made me more confident and eager to create more games in the future. I also got to play around with new technologies including the pathfinding algorithms.

Lots of FPS tutorials were available online and this made it easy to get inspiration and get my project off the ground. The genre of game that I chose did not limit me and I was free to make it as realistic/simple as I desired. I incorporated elements that were not planned like interactable objects, and object animation. The asset store was also a huge help in getting realistic models into my game. It also gave me more time as I did not have to make these models from scratch. I really enjoyed creating the scene and making the GUI for the project. It was a chance to be creative and test out different ideas.

There are some disadvantages associated with the development of an FPS game. It was evident during development that this type of game is extremely large and complex to develop thus requiring a substantial amount of time to even get it to a playable state. As I have no previous experience in game development, or Unity, it meant I had to learn how to use and navigate around a new program. I spent a lot of my initial time watching tutorials and getting myself familiar with Unity. I had basic C# skills, but this was not enough, so research was needed for this.

5.2 FUTURE SCOPE

I plan to continue development of this project in the future as I would love to see how far I could take it. I have got experience with a new program and it has aspired me to create more games. I really enjoyed the environment design aspect of the game. I thoroughly enjoyed creating the GUI and the menus for the game also. The game was a challenge for me but I am happy I took it on. I have also gained experience in time management and goals as this is the longest time frame, I have had a project spread over. I have learned the benefits of starting early as I am usually someone who leaves it till nearer the time and works better under pressure. I feel this experience will stand to me and I can apply what I have learned to other problems I will face after I leave college.

There are many ways that this project can be expanded in the future. I could create a new world with new characters and have the storyline continue on. I could make the game more challenging by adding in a more difficult enemy to defeat, adding more levels or by adding in obstacles to overcome like timers, missions etc. I could expand on weapons and have the option to switch weapon while playing the game. As I did not have time to incorporate a save feature, this is something I would definitely implement in the future.

A website could also be created to compliment the game. News and updates would be available on this. Users could write a short review of the game and give a rating. This feedback would be helpful in providing a better game play experience. I feel that I have not limited my options with the style of game I have chosen. The possibilities are endless, and the story can be developed. I can re-use the environment in as many levels as I wish.

5.3 ADDITIONAL PLATFORM DEVELOPMENT REQUIREMENTS

- iOS: Mac computer running minimum OS X 10.9.4 version and XCode 6.x.
- Android: Android SDK and Java Development Kit (JDK).
- Windows 8/8.1 Store Apps / Windows Phone 8/8.1: 64 bit Windows 8.1 Pro and Visual Studio 2013 Update 2+.
- WebGL: Mac OS X 10.8+ or Windows 7 SP1+ (64-bit editor only)

REFERENCES

Journal / Conference Papers

[1] Apa.org. (2016). [online]
Available at: http://www.apa.org/monitor/2014/02/video-game.aspx [Accessed 4 May 2016].

[2] Steve Benford, Chris Greenhalgh, Gail Reynard, Chris Brown, and Boriana Koleva, "Understanding and constructing shared 8 spaces with mixed-reality boundaries"
ACM Transactions on Computer-Human Interaction, vol. 5, no. 3, pp. 185–223, 1998.

[4] Peter Clarke, "Mobile giants pick each other for universal games," EE Times, Mar. 21, 2001,
Available at: http://www.eetimes.com/story/OEG20010321S0069

[5] David L. Nayland, Virtual Combat: A Guide to Distributed Interactive Simulation, Stackpole Books, Mechanicsburg, PA, 1997.

[6] Michael R. Macedonia, A Network Software Architecture for Large Scale Virtual Environments, Ph.D. thesis, Naval Postgraduate School, Monterey, CA, June 1995.

[10] Simon Powers, Mike Hinds, and Jason Morphett, "DEE: An architecture for distributed virtual environment gaming," Distributed Systems Engineering, vol. 5, no. 3, pp. 107–17, 1998.

[11] Software Testing Techniques: A Literature Review (2016)
Muhammad Abid Jamil, Muhammad Arif, Normi Sham Awang Abubakar, Akhlaq Ahmad.
KICT, International Islamic University, Kuala Lumpur, Malaysia
2016 6th International Conference on Information and Communication Technology for The Muslim World

[12] Emmanuel Frecon and Marten Stenius, "DIVE: A scalable network architecture for distributed virtual environments," Distributed Systems Engineering, vol. 5, no. 3,
pp. 91–100, 1998.

Reference / Handbooks
[9] star, D. (2016).
Difference and advantages between Dijkstra & A star. [online] Stackoverflow.com.
Available at:
http://stackoverflow.com/questions/13031462/difference-and-advantages-between-dijkstra-a-star [Accessed 5 May 2018].

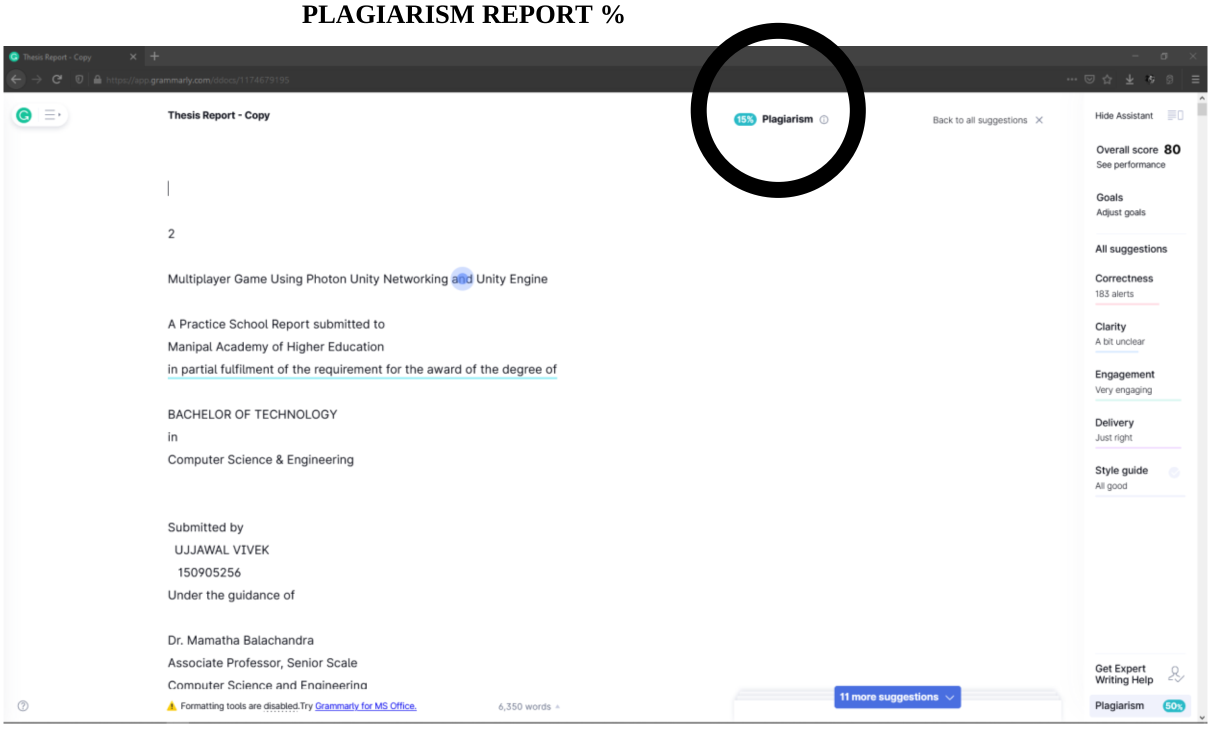
Web
[3] Unity documentation (2021) [online]
Available at: https://docs.unity3d.com/Manual/index.html

[7] Visual Studio Documentation (2021) [online]
Available at: https://docs.microsoft.com/en-us/visualstudio/windows/?view=vs-2019

[8] C# Docs (2020) [online]
Available at: https://docs.microsoft.com/en-us/dotnet/csharp/

PLAGIARISM REPORT

PLAGIARISM REPORT %



Thesis Report - Copy

https://app.grammarly.com/ddocs/1174679195

Thesis Report - Copy

15% Plagiarism

Back to all suggestions

2

Multiplayer Game Using Photon Unity Networking and Unity Engine

A Practice School Report submitted to
Manipal Academy of Higher Education
in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY
in
Computer Science & Engineering

Submitted by
UJJAWAL VIVEK
150905256
Under the guidance of

Dr. Mamatha Balachandra
Associate Professor, Senior Scale
Computer Science and Engineering

Formatting tools are disabled.Try Grammarly for MS Office.

6,350 words

11 more suggestions

PROJECT DETAILS

Student Details			
Student Name	**Ujjawal Vivek**		
Register Number	150905256	Section / Roll No	A
Email Address	Ujjwalvivek21@gmail.com	Phone No (M)	8789389591
Project Details			
Project Title	**Multiplayer game using Photon Unity Networking and Unity Game Engine**		
Project Duration	4 Months	Date of reporting	
Organization Details			
Organization Name	**Manipal Institute of Technology**		
Full postal address with pin code	Manipal - 576104		
Website address	www.manipal.edu		
Internal Guide Details			
Faculty Name	**Dr. Mamatha Balachandra**		
Full contact address with pin code	Dept of Computer Science & Engg, Manipal Institute of Technology, Manipal – 576 104 (Karnataka State), INDIA		
Email address	mamtha.bc@manipal.edu		