FPS SURVIVAL GAME USING UNITY

A PROJECT REPORT

Submitted by

SUJITH KUMAR S 211418104272

VIGNESH V 211418104309

VAIDHYANATHAN S 211418104299

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

MAY 2022

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **"FPS SURVIVAL GAME USING UNITY"** is the bonafide work of **"SUJITH KUMAR S [211418104272], VIGNESH V [211418104309], VAIDHYANATHAN S [211418104299]"** who carried out the project work under my supervision.

SIGNATURE

Dr. S. MURUGAVALLI, M.E., Ph.D.,
HEAD OF THE DEPARTMENT

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

SIGNATURE

Dr. S. HARIHARAN, M.E., Ph.D.,
ASSISTANT PROFESSOR

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the Anna University Project Viva-Voce Examination held on.....

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION BY THE STUDENT

We **"SUJITH KUMAR S [211418104272], VIGNESH V [211418104309], VAIDHYANATHAN S [211418104299]"** hereby declare that this project report titled "**FPS SURVIVAL GAME USING UNITY**", under the guidance of "**Dr. S. HARIHARAN, M.E., Ph.D.**" is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

SUJITH KUMAR S

VIGNESH V

VAIDHYANATHAN S

ACKNOWLEDGEMENT

We express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We would like to extend our heartfelt and sincere thanks to our Directors **Tmt.C.VIJAYARAJESWARI**, **Dr.C.SAKTHIKUMAR, M.E., Ph.D.** and **Dr.SARANYASREE SAKTHIKUMAR B.E., M.B.A., Ph.D.** for providing us with the necessary facilities for completion of this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.** for his timely concern and encouragement provided to us throughout the course.

We thank the Head of the CSE Department, **Dr. S. MURUGAVALLI, M.E., Ph.D.,** for the support extended throughout the project.

We would like to thank our Project Coordinator **Dr. N. PUGHAZENDHI, M.E.**, **Ph.D.** Project Guide **Dr. S. HARIHARAN, M.E., Ph.D.** and all the faculty members of the Department of CSE for their advice and suggestions for the successful completion of the project.

ABSTRACT

The successful design of the FPS game with correct direction, attractive graphics and models will give the best experience to play the game. First person shooter game is type of 3d video game genre which is now a day's quite popular in game industry. The main design element of this type of games is combat and action centred storyline. This game comes also comes under role player category too, because player must play with protagonist point of view. Our aim is to create an attractive game that can be played in both low and high end devices.

It has been found in researches that there is an association between playing first-person shooter survival video games and having superior mental flexibility. It was found that people playing such games require a significantly shorter reaction time for switching between complex tasks, mainly because when playing fps survival games they require to rapidly react to fast moving visuals by developing a more responsive mind set and to shift back and forth between different sub-duties.

Despite the economic instability and crisis deeply affecting the world, the analysts published that the game industry has grown at a rate of 57% surprisingly. Even as we type these words, millions of people game in front of their computers. The reason of this growth can be stated that the game industry can appeal any users with different tastes.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	v
	LIST OF TABLES	viii
	LIST OF FIGURES	ix
1.	**INTRODUCTION**	1
2.	**LITERATURE SURVEY**	4
3.	**SYSTEM ANALYSIS**	14
	3.1 EXISTING SYSTEM	15
	3.2 PROPOSED SYSTEM	15
	3.3 FEASIBILITY STUDY	16
	3.3.1 Technical Feasibility	16
	3.3.2 Economic Feasibility	16
	3.3.3 Source Feasibility	17
	3.4 DEVELOPMENT ENVIRONMENT	17
	3.5 REQUIREMENT SPECIFICATION	18
	3.5.1 Hardware Requirements	18
	3.5.2 Software Requirements	18
4.	**SYSTEM DESIGN**	19
	4.1 UML Diagrams	20
	4.1.1 Use Case Diagram	20
	4.1.2 Activity Diagram	21
	4.1.3 Sequence Diagram	22
	4.1.4 Class Diagram	23
	4.1.5 Collaboration Diagram	24
	4.2 Data Flow Diagram	25

CHAPTER NO.	TITLE	PAGE NO.
5.	**SYSTEM ARCHITECTURE**	26
	5.1 SYSTEM ARCHITECTURE DIAGRAM	27
	5.2 MODULE DESCRIPTION	28
	5.3 ALGORITHM DESCRIPTION	31
6.	**SYSTEM IMPLEMENTATION**	33
	6.1 DEVELOPER-SIDE CODING	34
7.	**SYSTEM TESTING**	57
	7.1 TESTING OBJECTIVES	58
	7.2 TESTING TYPES	58
	7.2.1 Unit Testing	58
	7.2.2 Integration Testing	59
	7.3 TEST CASES AND RESULTS	59
8.	**CONCLUSION**	65
	8.1 CONCLUSION & FUTURE ENHANCEMENT	66
9.	**APPENDICES**	67
	A.1 SAMPLE SCREENSHOTS	68
10.	**REFERENCES**	72

LIST OF TABLES

TABLE NO.	TABLE DESCRIPTION	PAGE NO.
7.1	Test Cases and Results	59

LIST OF FIGURES

FIG NO.	FIGURE DESCRIPTION	PAGE NO.
4.1.1	Use Case Diagram	20
4.1.2	Activity Diagram	21
4.1.3	Sequence Diagram	22
4.1.4	Class Diagram	23
4.1.5	Collaboration Diagram	24
4.2	Data Flow Diagram	25
5.1	System Architecture Diagram	27
5.2.1	Image of crossbow	28
5.2.2	Image of gun	28
5.2.3	Image of energy drink	29
5.2.4	Image of bat	29
5.2.5	Image of axe	29
5.2.6	Image of battery	29
5.2.7	Image of knife	30
5.2.8	Screenshot of inventory	30
9.1.1	Screenshot of environment	68
9.1.2	Screenshot of zombie	68
9.1.3	Screenshot of battery inside a building	69
9.1.4	Screenshot of start menu	69
9.1.5	Screenshot of loading screen	70
9.1.6	Screenshot of ending screen	70
9.1.7	Screenshot of player inside building	71
9.1.8	Screenshot of game over screen	71

INTRODUCTION

1. INTRODUCTION

Game Development is the art of creating games and describes the design, development, and release of a game. It may involve concept generation, design, build, test, and release. While you create a game, it is important to think about the game mechanics, rewards, player engagement and level design. Games can be as small or large as you like. If it lets the player interact with content and can manipulate the game's elements, you can call it a 'game'.

Games are not just a computer software that are made to benefit user's daily life, rather it is made for user's entertainment purpose, so we need to pay attention to what the user wants from the game, how to make it more entertaining, just making any game will not do, that is why making games are more challenging .

Games increase player's brain and help to increase its learning ability and also help the person to take better decisions which will help him/her for facing difficult situations. Gaming also improves reflex, Vision, and Creativity of player. In this work, we are developing FPS Survival game i.e. First-person shooter, it is a video game genre centred around gun and other weapon-based combat in a first-person that requires players to master the skills necessary to operate several types of guns and other weapons that are typically used in eliminating enemy. This game also comes under the role playing category too.

This FPS Survival game involves a single player entering a horror environment. Initially the player must find a weapon and need to equip it from the inventory menu. The environment is inhabited by creatures (i.e., enemies) that will attack the player if he goes into the area that the creatures inhabiting. If the player successfully kills the

creatures the player moves to the upcoming enemies and environment and finally, he must find the keys. The player can exit the game by pressing escape key or quit button in the menu option.

This game contains lot of pickup objects such as Battery, Drink, weapons, etc., where the player can equip each of them from the inventory. Each of the objects has its specific purpose in the game. For example, Battery will be used by the player when the battery percentage decreases by using the Torch light and Night vision mode. Drink will be used, when the player health decreases by being attacked by the enemies and weapons like Gun, bat, crossbow, knife, and axe will be used, when the player needs to defend himself from the enemies.

In this game we added some sound effects for player, enemies, trees, lake etc., which will vary according to the movement of the background objects in the game. This will be an additional effect for the users who are playing this game.

We also added the feature of save system, which is when the player who plays the game wants to leave the game in a continued state, he can quit the current game and can load the existing game from the menu, whenever he wants to continue the same game.

LITERATURE SURVEY

2. LITERATURE SURVEY

- **TITLE NAME:** Using Games to Study Law of Motions in Mind

AUTHOR NAME: Hiroyuki lida, MohdNor Akmal Khalid.

IEEE – Journal Article - 2020

DESCRIPTION:

The development of games had been seen as an essential field of study in artificial intelligence. Although game concepts have been adopted in various problem domains, understanding the nature of game playing's underlying mechanism has been limited. This study investigates such a mechanism by adopting a logistic model of game outcome uncertainty where a measure of game refinement had been derived.

Application of such a measure to the popular board and sports games have been conducted to determine its implication in determining sophisticated design and has defined the harmonic balance between skill and chance. A theoretical analysis using practical data confirms the effectiveness of the proposed model, where reasonable game length is determined based on the definition of a gamified experience. The model is further expanded with the analogy of physics in mind. The law of motions in the game is formalized by considering the derivatives of the game progress model, where the link between game refinement theory and flow theory is identified and discussed. In essence, the sophisticated game's underlying mechanism leads to a more natural yet pleasurable human experience.

- **TITLE NAME:** Optimizing Player and Viewer Amusement in Suspense Video Games.

AUTHOR NAME: Pablo Delatorre, Carlos Leon, Alberto Salguero Hidalgo, Alan Tapscott.

IEEE – Journal Article - 2019

DESCRIPTION:

Broadcast video games need to provide amusement to both players and audience. To achieve this, one of the most consumed genres is suspense, due to the psychological effects it has on both roles. Suspense is typically achieved in video games by controlling the amount of delivered information about the location of the threat. However, previous research suggests that players need more frequent information to reach similar amusement than viewers, even at the cost of jeopardizing viewers' engagement. To obtain models that maximize amusement for both interactive and passive audiences, we conducted an experiment in which a group of subjects played a suspenseful video game while another group watched it remotely. The subjects were asked to report their perceived suspense and amusement, and the data were used to obtain regression models for two common strategies to evoke suspense in video games: by alerting when the threat is approaching and by random circumstantial indications about the location of the threat. The results suggest that the optimal level is reached through randomly providing the minimal amount of information that still allows players to counteract the threat. We reckon that these results can be applied to a broad narrative media, beyond interactive games.

- **TITLE NAME:** Lag Compensation for First-Person Shooter Games in CloudGaming.

AUTHOR NAME: Zhi Li, Hugh Melvin, Rasa Bruzgience, Peter Pocta, Lea Skorin-Kapov, Andrej Zgank.

SPRINGER – 2018

DESCRIPTION:

Cloud gaming is an emerging technology that combines cloud computing with computer games. Compared to traditional gaming, its core advantages include ease of development/deployment for developers, and lower technology costs for users given the potential to play on thin client devices. In this chapter, we firstly describe the approach, and then focus on the impact of latency, known as lag, on Quality of Experience, for so called First-Person Shooter games.

We outline our approach to lag compensation whereby we equalize within reason the up and downlink delays in real-time for all players. We describe the testbed in detail, the open-source Gaming Anywhere platform, the use of NTP to synchronize time, the network emulator, and the role of the centralized log server. We then present results that firstly validate the mechanism and use small scale and preliminary subjective tests to assess and prove its performance. We conclude the chapter by outlining ongoing and future work.

- **TITLE NAME:** Playtime Measurement with Survival Analysis

AUTHOR NAME: Markus Viljanen, Antti Airola, Jukka Heikkonen, Tapio Pahikkala

IEEE – Journal Article- 2018

DESCRIPTION:

Maximizing product use is a central goal of many businesses, which makes retention and monetization two central analytics metrics in games. Player retention may refer to various duration variables quantifying product use: total playtime or session play time are popular research targets, and active playtime is well-suited for subscription games. Such research often has the goal of increasing player retention or conversely decreasing player churn.

Survival analysis is a framework of powerful tools well suited for retention type data. This paper contributes new methods to game analytics on how playtime can be measured using survival analysis. These methods include visualizations, metrics, and an AB-test based on the survival curve. All these methods work on censored data and enable computation of confidence intervals. This is especially important in time- and sample-limited data that occurs during game development. Throughout this paper, we illustrate the application of these methods to the development of the Hipster Sheep mobile game.

- **TITLE NAME:** Rapid Skill Capture in a First-Person Shooter.

AUTHOR NAME: David Buckley, Ke Chen, Joshwa Knowles.

IEEE - Journal Article **-** 2017

DESCRIPTION:

Various aspects of computer game design, including adaptive elements of game levels, characteristics of "bot" behavior, and player matching in multiplayer games, would ideally be sensitive to a player's skill level. Yet, while game difficulty and player learning have been explored in the context of games, there has been little work analyzing skill per se, and how this is related to the interaction of a player with the controls of the game - the player's input. To this end, we present a data set of 476 game logs from over 40 players of a first-person shooter game (Red Eclipse) as a basis of a case study. We then extract features from the keyboard and mouse input and provide an analysis in relation to skill. Finally, we show that a player's skill can be predicted using less than a minute of their keyboard presses. We suggest that the techniques used here are useful for adapting games to match players 'skill levels rapidly, arguably more rapidly than solutions based on performance averaging such as TrueSkill.

- **TITLENAME:** Challenges and Opportunities in Game Artificial Intelligence Education Using Angry Birds

AUTHOR NAME: Du-MimYoon, Kyung-joong Kim**.**

IEEE – Journal Article **-** 2015

DESCRIPTION:

Games have been an important tool for motivating undergraduate students majoring in computer science and engineering. However, it is difficult to build an entire game for education from scratch, because the task requires high-level programming skills and expertise to understand the graphics and physics. Recently, there have been many different game in artificial intelligence (AI) competitions, ranging from board games to the state-of-the-art video games (car racing, mobile games, first-person shooting games, real-time strategy games, and so on).

The competitions have been designed such that participants develop their own AI module on top of public/commercial games. Because the materials are open to the public, it is quite useful to adopt them for an undergraduate course project. In this paper, we report our experiences using the Angry Birds AI Competition for such a project-based course. In the course, teams of students consider computer vision, strategic decision-making, resource management, and bug-free coding for their outcome. To promote understanding of game contents generation and extensive testing on the generalization abilities of the student's AI program, we developed software to help them create user-created levels. Students actively participated in the project and the final outcome was comparable with that of successful entries in the 2013 International Angry Birds AI Competition. Furthermore, it leads to the development of a new parallelized

Angry Birds AI Competition platform with undergraduate students aiming to use advanced optimization algorithms for their controllers.

- **TITLE NAME:**Adaptive Shooting for Bots in First Person Shooter Games Using Reinforcement Learning

AUTHOR NAME: Frank G. Glavin, Michael G. Madden.

IEEE – Journal Article **-** 2015

DESCRIPTION:

In current state-of-the-art commercial first-person shooter games, computer-controlled bots, also known as nonplayer characters, can often be easily distinguishable from those controlled by humans. Tell-tale signs such as failed navigation, “sixth sense” knowledge of human players' whereabouts and deterministic, scripted behaviors are some of the causes of this. We propose, however, that one of the biggest indicators of non-human like behavior in these games can be found in the weapon shooting capability of the bot.

Consistently perfect accuracy and “locking on” to opponents in their visual field from any distance are indicative capabilities of bots that are not found in human players. Traditionally, the bot is handicapped in some way with either a timed reaction delay or a random perturbation to its aim, which does not adapt or improve its technique over time. We hypothesize that enabling the bot to learn the skill of shooting through trial and error, in the same way a human player learns, will lead to greater variation in game play and produce less predictable nonplayer characters. This paper describes a reinforcement learning shooting mechanism for adapting shooting over time based on a dynamic reward signal from the amount of damage caused to opponents.

- **TITLE NAME:** Learning to win in a first-person shooter game.

AUTHOR NAME: ChishyanLiaw, Wei-Hua Andrew Wang, Chung-Chi &Yu-Liang Hsu.

SPRINGER-2013

DESCRIPTION:

Designing a character's behavior in a first-person shooter (FPS) game typically requires a considerable amount of effort due to a complex in-game environment. This paper proposes an efficient strategy of training a character in the Quake III Arena, a FPS game, to be more adaptive and also to have human-like learning capabilities and behaviors. Specifically, a particle swarm optimization (PSO) algorithm is introduced to provide a character with self-learning abilities. Like many computer games, the Quake III Arena utilizes a rule-based system which is constrained by several weights in the software. The effectiveness of these constrained weights is dependent upon the programmer's knowledge about the game. In order to increase a character's learning ability, an efficient method using the PSO algorithm is developed to determine the constrained weights for the behavior control. In the conducted experiments, the PSO algorithm is implemented to design a non-player character which is shown to be superior to the other characters originally created in the Quake III game. This efficient training strategy decreases the amount of effort required by game designers to design an intelligent character's behavior.

- **TITLE NAME:** Applying inexpensive AI techniques to computer games

AUTHOR NAME: A. Khoo, R. Zubek

IEEE – Magazine Article - 2002

DESCRIPTION:

Groo (Generic Robot, Object-Oriented) and tt14m (Trash-Talking 14-year-old Moron) are two systems that use simple and computationally inexpensive artificial intelligence mechanisms to produce engaging character behavior for computer games, while remaining within the performance constraints of modern game development. Groo engages in intelligent tactical behavior in a first-person shooter death-match game using a fairly simple and static behavior network, while tt14m uses simple text processing to attempt engagement in the social aspects of the game "Counter-Strike".

SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

It is difficult to precisely define the existing FPS games due to an increasing number of FPS games on the market, however the basic concepts found in most FPS games are heroes, enemies, guns and different skills for different heroes, resource collection and consumption, bases with defensive structures, creeps, and lanes.

Disadvantages of Existing System

- The existing games have high graphics and it does not have a fixed fps for devices which results in lag and rendering issues.

3.2 PROPOSED SYSTEM

A first-person shooter (FPS) is a genre of action video game that is played from the point of view of the protagonist. The gamer is expected to propel his avatar through the game by moving it forward, backward, sideways, and so on using the game controller. Forward movements of the controller result in the avatar moving forward through the different types of FPS video games have on adolescent behaviour. It helps the gamer to rapidly react to fast moving visuals by developing a more responsive mind set and to shift back and forth between different sub-duties.

Advantages of Proposed System

- The rendering of objects will be the same for all the devices due to commonly set settings. This avoids lag issues.

3.3 FEASIBILITY STUDY

3.3.1 Technical Feasibility

This FPS Survival Game can be played in the desktop. The main technologies that are associated with project are

- Unity 2020.3.30f1 (64-bit)
- Adobe Photoshop CS6
- Code Editor: Visual Studio Code
- C#

3.3.2 Economic Feasibility

FPS survival is a single player offline game so there is no hosting cost. For the PC game players, it is completely a free to play game, which is so easy to play and also a user-friendly game for the players. Additionally, if we add some premium weapons or player skins in future the player may need to pay to acquire them which is a choice for the player.

3.3.3 Source Feasibility

The resources that are required for this project are:
Any laptop/PC with a minimum RAM of 4GB and a decent GPU can be used for the development of this game. The software needed to develop this game are available no charge. We have used free assets in this game. But if we want to release the game for monetizing purposes then graphic designers are required to get proper recognition.

3.4 DEVELOPMENT ENVIRONMENT

Unity is a cross-platform game engine developed by Unity Technologies, first announced, and released in June 2005 at Apple Inc.'s Worldwide Developers Conference as a Mac OS X-exclusive game engine. The engine has since been gradually extended to support a variety of desktop, mobile, console and virtual reality platforms. It is particularly popular for iOS and Android mobile game development and used for games such as Pokémon Go, Monument Valley, Call of Duty: Mobile, Beat Saber and Cuphead. It is considered easy to use for beginner developers and is popular for indie game development.
The engine can be used to create three-dimensional (3D) and two-dimensional (2D) games, as well as interactive simulations and other experiences. The engine has been adopted by industries outside video gaming, such as film, automotive, architecture, engineering, construction, and the United States Armed Forces.
Unity is a cross-platform engine, and the Unity editor is supported on Windows, macOS, and the Linux platform. while the engine itself currently supports building games for more than 19 different platforms, including mobile, desktop, consoles, etc., Officially supported Mobile platforms are iOS, Android TV. Desktop platforms are

Mac, Linux. Web platform is WebGL. Console platforms are PlayStation, Xbox, Nintendo Switch, Stadia.

3.5 REQUIREMENT SPECIFICATION

3.5.1 Hardware Requirements

Processor: I5 6th Generation
RAM: 8 GB
Hard Disk: 1 TB
Graphics Card: NVIDIA GeForce GT 730

3.5.2 Software Requirements

Unity 2020.3.30f1 (64-bit) - We need to install this version of editor with Unity Hub. The system requirement for using this will be Windows 7 SP1+, 10, 64-bit versions only.
Unity Hub- Unity hub is a standalone application that streamlines the way you find, download, and manage your unity projects and installations.
Adobe Photoshop- Adobe Photoshop is a graphics editor in which we have used this to develop our inventory menu.
Visual Studio Code - Visual Studio is a platform where we develop our computer programs. Here we used this Visual Code to develop our script which we have written in C#.
C# - C# is a programming language used for developing web applications and services. In this game we used this C# language to develop our code script.

SYSTEM DESIGN

4. SYSTEM DESIGN

4.1 UML DIAGRAMS

4.1.1 Use Case Diagram

This Use Case diagram is used to represent the dynamic behavior of the game. It encapsulates the system's functionality by incorporating use cases, player, enemy and their relationships. The player will initialize the game and will start to play the game by picking up the needed objects from the game scenario and the player will equip those objects from the inventory menu and when the enemy attacks the player, the player can kill the enemy by using the equipped objects. The player can also exit the game by using the options menu or by using 'Esc' key.

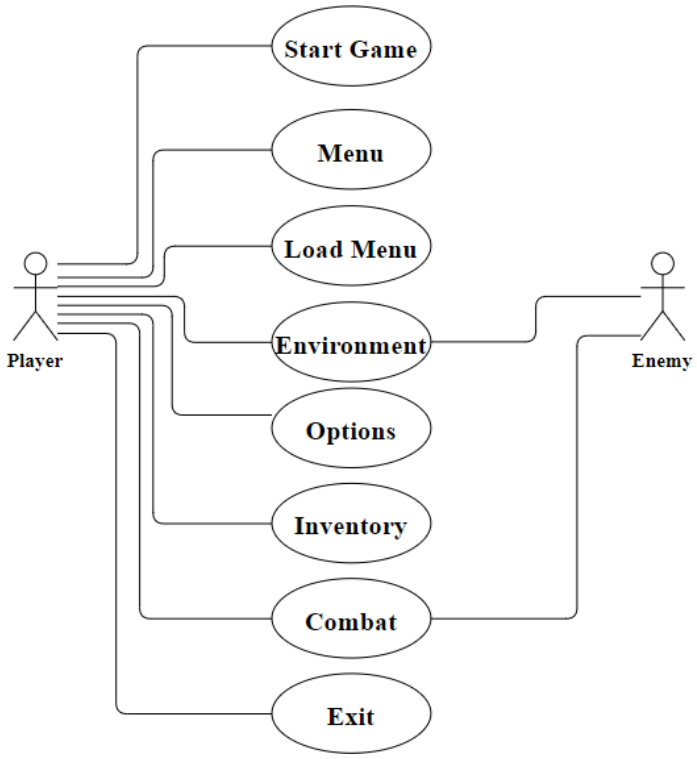


Fig (4.1.1) Use Case Diagram

4.1.2 Activity Diagram

In UML, the activity diagram is used to demonstrate the flow of control within the system rather than the implementation. In this, the player will start the game by clicking the game options from the Menu. The player will either load the existing game or a new game. After this the player will enter the game environment and start to play the game. The player can select options menu to save the game, or he can return to the menu. When the player finds the game objectives i.e., keys, the game will be over or if the game objectives in not over, the player can return to the menu.

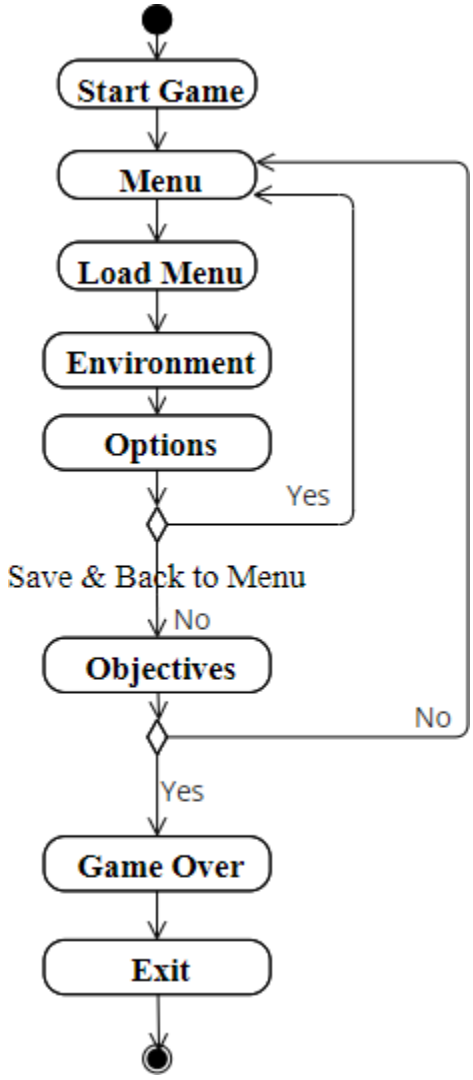


Fig (4.1.2) Activity Diagram

4.1.3 Sequence Diagram

The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. The player will launch the game from the Menu, and he will start to play the game. The player will provide the necessary input to control the hero i.e., to turn left, right, pickup objects etc., According to the inputs the game engine will run, and the game objective is to kill the enemy by the given player inputs. The game will restart when the hero dies. The player can exit the game from options menu or by pressing the 'Esc' key.

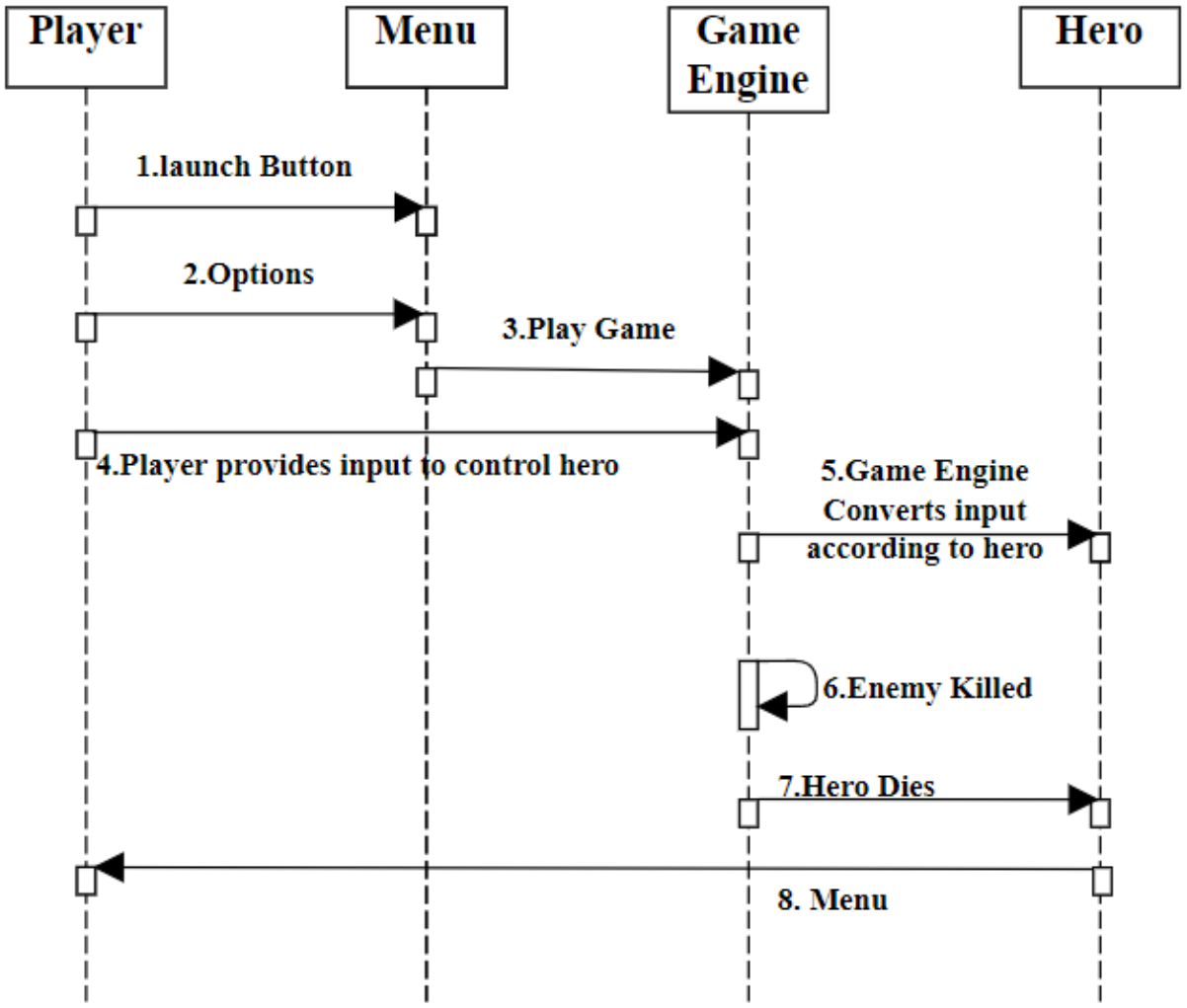


Fig (4.1.3) Sequence Diagram

4.1.4 Class Diagram

A class diagram is used to describe the attributes and operations of a class. This Class diagram consists of three components such as name of the class, attributes, and methods. The player class consists of various attributes which describes the player actions and several methods which describes the Player functions. The enemy class consists of the enemy actions as attributes and enemy functions as methods. The menu class consists of the game options as attributes and game functions as methods. The environment class consists of the game surroundings as attributes and game functions as methods.

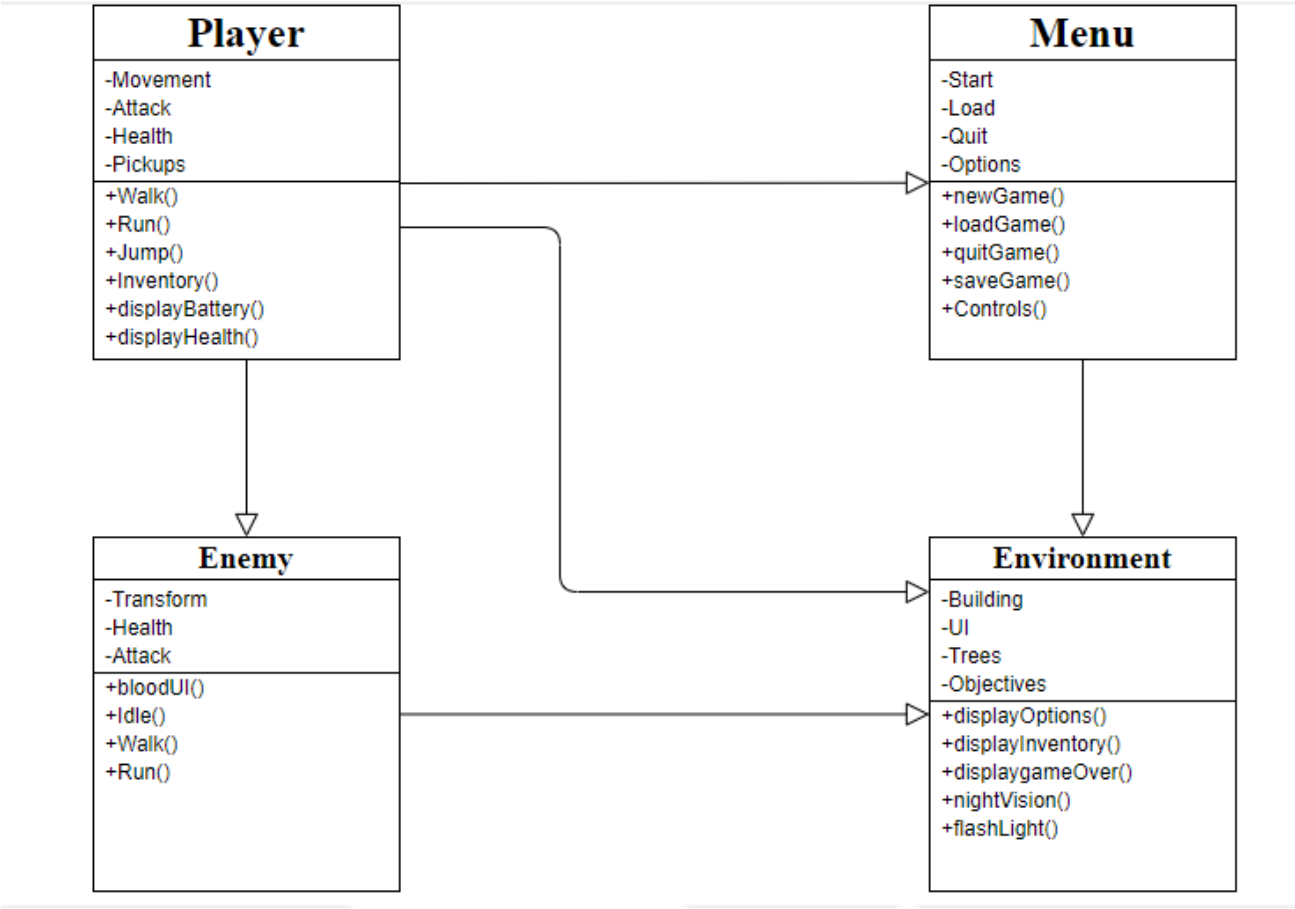


Fig (4.1.4) Class Diagram

4.1.5 Collaboration Diagram

The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. Here the player starts the game, then the main menu will open, in that he starts a new game, after starting the game, he enters the environment and fights with the zombies and then the game gets over.

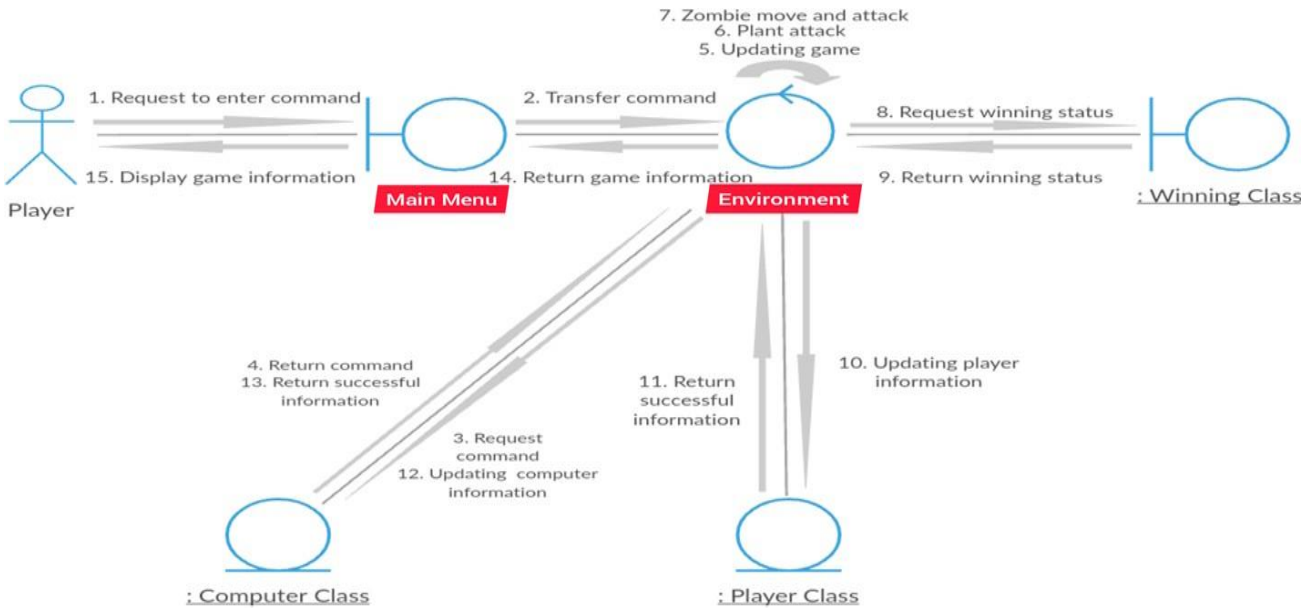


Fig (4.1.5) Collaboration Diagram

4.2 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. It shows how data enters and leaves the system. This DFD consists of Intro scene where the game will get started and the Menu scene consists of options such as play game, load game, and quit game. The main scene consists of inventory, where the objects will get stored and options menu, where we can save or quit the game. The win scene consists of player options and Exit button, where the player can exit the game when he wins. The lose scene consists of player options, go to menu scene, and Exit button. When the player losses he can either restart the game from the menu scene or exit the game using Exit button.

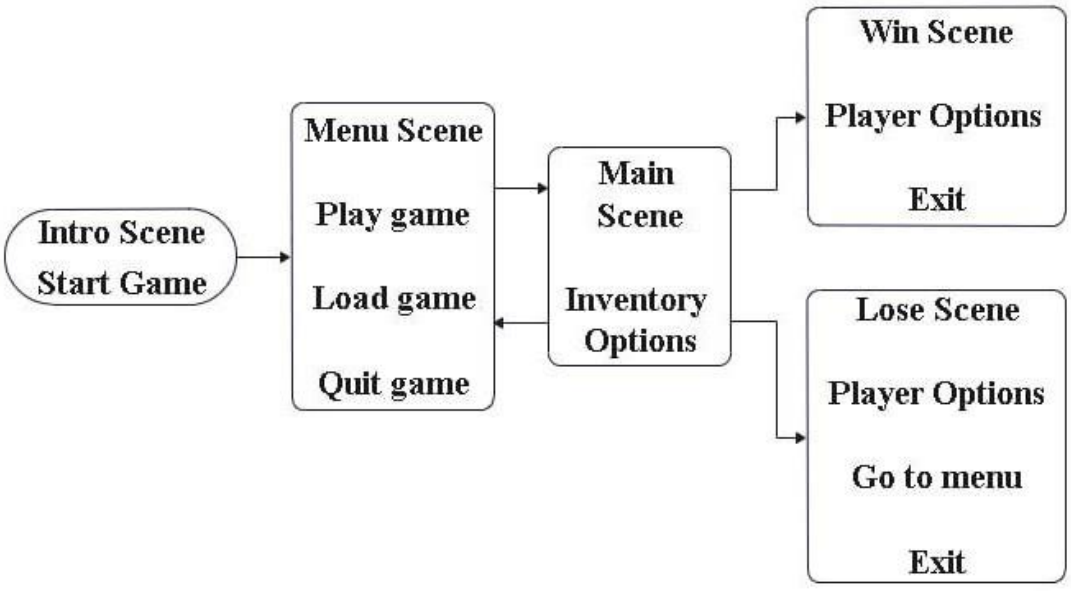


Fig (4.2) Data Flow diagram

SYSTEM ARCHITECTURE

5. SYSTEM ARCHITECTURE

5.1 SYSTEM ARCHITECTURE DIAGRAM

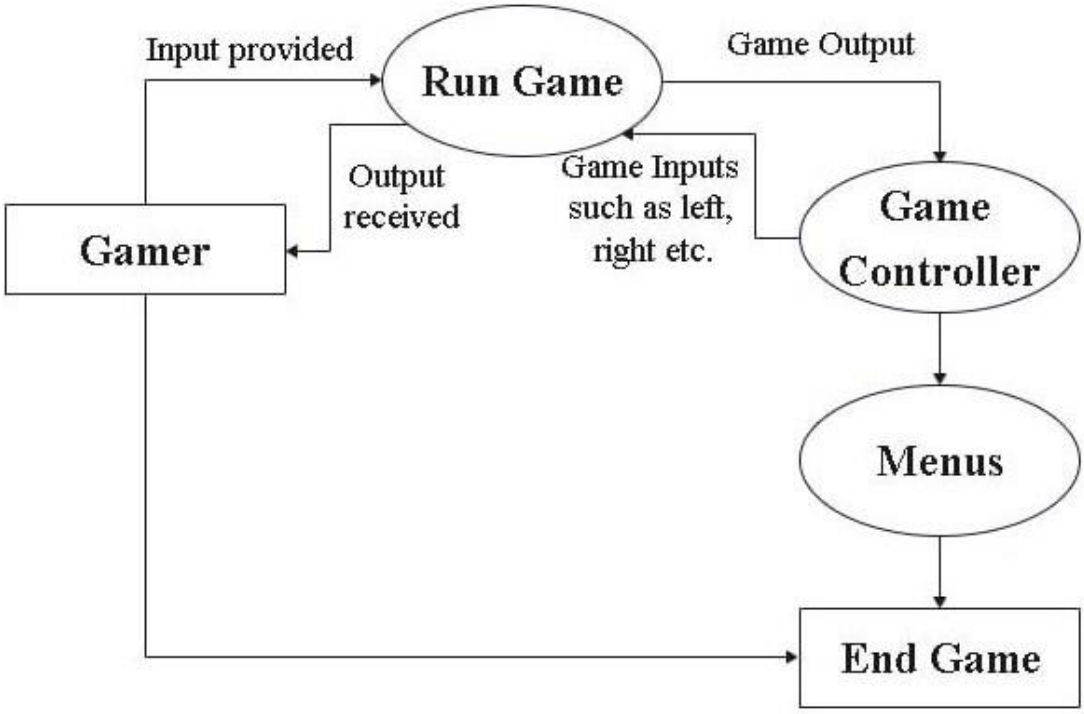


Fig (5.1) System Architecture Diagram

To start this game the gamer will provide the necessary input from the menu i.e., Load game, Start game, Exit. According to the options selected, the gamer will receive the output from the system, such as to start a new game or to start the already existing game or to exit the game. After this, the gamer will begin to run the game by giving game inputs such as to turn left, right, picking up the objects, attacking enemies etc., and the game will receive the output for the same from the Game controller. The objects that the gamer has picked up will be stored in the Menu i.e., Inventory. After collecting all the Keys, the gamer can exit the game from the options menu or by pressing the ‘Esc’ key.

5.2 MODULE DESCRIPTION

Display Screen

Player will open the game by clicking the application in the screen.

UI

In the opening screen, the UI will show VVS Creations and then the main menu will appear. The player can start playing the game by clicking the start button.

In the main screen the player can load the saved game or can start a new game. The character of the person in the game will be seen in FPP mode where the user can only see the weapon in the hands of the character. The FPS character can move by clicking the arrow keys or can use the ‘ASWD’ keys and can jump using the space bar.

Pickups

The player can pick up objects like ammo, battery, etc. by going next to it and pressing ‘E’, can use flashlight using ‘F’ key and can use night vision glasses using ‘N’ key.

Crossbow

Weapon used by the player that shoots a short arrow with great force which will kill the enemy.

Fig (5.2.1) Crossbow

Gun

The player should pick up and equip this weapon and the usage of this weapon is to shoot the enemies.

Fig (5.2.2) Gun

Drink

When enemy attack the player, the player health will decrease instantly. So, to regain his health the player needs to pickup this drink and should equip it from the inventory.

Fig (5.2.3) Drink

Bat

This weapon will find somewhere in the game scenario and the player should equip this to safeguard himself from the enemies.

Fig (5.2.4) Bat

Axe

This is a weapon with steel edge and wooden handle, which will be used by the player to kill the enemies.

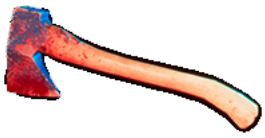


Fig (5.2.5) Axe

Battery

When the player uses the flashlight and night vision mode, the battery will decrease instantly. So, to regain the battery the player needs to pick up this battery.

Fig (5.2.6) Battery

Knife

This is a weapon with sharp edges, which can be used by the player to kill the enemies.

Fig (5.2.7) Knife

Environment

The environment is an island which consists of buildings, bridges, vehicles, river, and lakes.

Inventory

The player can open this inventory menu by pressing the key 'I'. The objects which we have picked up from the game scenario will be shown in this Inventory menu i.e., Keys, Weapons, Health, Ammo, Objective, and Batteries. The player can equip any of those items from this inventory menu.

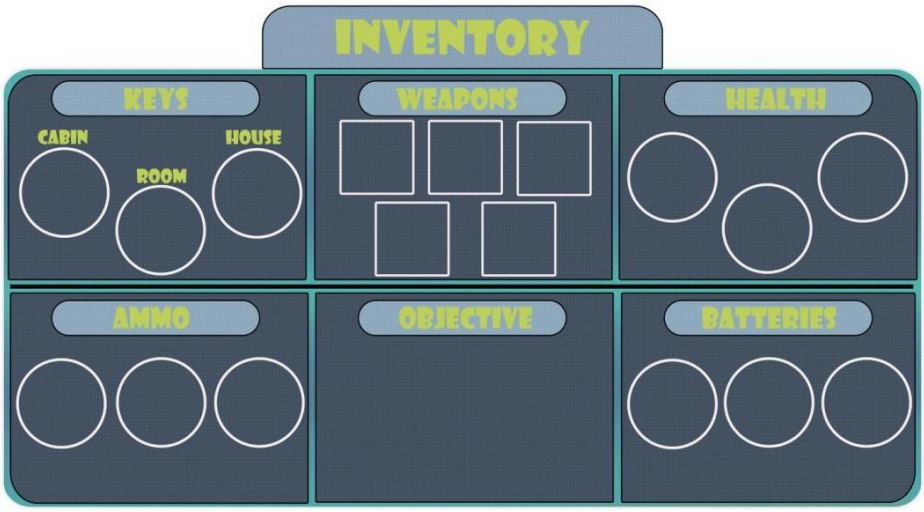


Fig (5.2.8) Inventory

Storyline

The player will be left alone in a no man's land where zombies are surrounded, the objective of the player is to find the Zombie extinction key, before finding the key the zombies will attack the player, he should overcome all the attacks and find the key. If he finds the correct key, then the game will end.

Exit

The player can exit the game by clicking the exit button.

5.3 ALGORITHM DESCRIPTION

Step 1: The player launches the game.

Step 2: The player enters the game.

Step 3: First person character and game environment will be displayed on the screen.

Step 4: The player can press arrow keys or ASWD for movement and space key for jump.

Step 5: The player can pick up objects like Weapon, Ammo, Battery, etc. by going next to it and pressing the key 'E'.

Step 6: Once picking an object from the environment the player has to press 'I' to open inventory menu.

Step 7: Then the player has to click on the pickup in the inventory menu to equip it

Step 8: To avoid darkness, the player can be able to turn ON the flashlight by pressing the key 'F'.

Step 9: Once the existing flashlight is over the player has to find for new battery in the environment.

Step 10: Then he can use it by picking and equipping from the inventory

Step 11: The player can also be able to switch to night vision mode by pressing the key ‘N’.

Step 12: Once the existing night vision mode is over the player has to follow the same procedure like flashlight as mentioned above.

Step 13: when the player enters the enemy range, the enemy will start attacking the user.

Step 14: The player can use his weapon for killing the enemy.

Step 15: If player’s health or the enemy’s health reaches to zero, they die and the player can regain his health by picking up the energy drink.

Step 16: If enemies killed player, then it goes to main menu.

Step 17: The player can also save the game using the options menu.

Step 18: The player could exit the game once if he finds the key or whenever he wished to exit the game, by pressing the esc button in the keyboard.

SYSTEM IMPLEMENTATION

6. SYSTEM IMPLEMENTATION

6.1DEVELOPER-SIDE CODING

CharController_Motor2:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class CharController_Motor2 :MonoBehaviour
{
public float speed = 10.0f;
public float sensitivity = 30.0f;
public float WaterHeight = 15.5f;
CharacterControllercharacter;
public GameObjectcam;
float moveFB, moveLR;
float gravity = -9.8f;
public float SpeedH = 10f;
public float SpeedV = 10f;
private static float yaw = 0f;
private static float pitch = 0f;
private float minPitch = -30f;
private float maxPitch = 30f;
void Start(){
character = GetComponent<CharacterController> ();
}
void CheckForWaterHeight(){
if (transform.position.y<WaterHeight) {
```

```
gravity = 0f;
} else  {
gravity = -9.8f;
}}
void Update(){
moveFB = Input.GetAxis ("Horizontal") * speed;
moveLR = Input.GetAxis ("Vertical") * speed;
yaw += Input.GetAxis("Mouse X") * SpeedH;
pitch -= Input.GetAxis("Mouse Y") * SpeedV;
pitch = Mathf.Clamp(pitch, minPitch, maxPitch);
CheckForWaterHeight ();
Vector3 movement = new Vector3 (moveFB, gravity, moveLR);
CameraRotation (cam, yaw, pitch);
If (Input.GetKey(KeyCode.W)||  Input.GetKey(KeyCode.S) ||
Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.D))
{
movement = transform.rotation * movement;
}
character.Move (movement * Time.deltaTime);
}
void CameraRotation(GameObject cam, float yaw, float pitch)
{
transform.eulerAngles = new Vector3(pitch, yaw, 0f);
cam.transform.eulerAngles = new Vector3(pitch, yaw, 0f);
}}
```

EnemyAttack:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class EnemyAttack :MonoBehaviour
{
    private NavMeshAgentNav;
    private NavMeshHitHit;
    private bool blocked = false;
    private bool RunToPlayer = false;
  private float DistanceToPlayer;
  private bool IsChecking = true;
  private int FailedChecks = 0;
  [SerializeField] Transform Player;
  [SerializeField] Animator Anim;
  [SerializeField] GameObjectEnemy;
    [SerializeField] float MaxRange = 35.0f;
    [SerializeField] int MaxChecks = 3;
    [SerializeField] float ChaseSpeed = 8.5f;
    [SerializeField] float WalkSpeed = 1.6f;
    [SerializeField] float AttackDistance = 2.3f;
    [SerializeField] float AttackRotateSpeed = 2.0f;
    [SerializeField] float CheckTime = 3.0f;
    void Start()
    {
        Nav = GetComponentInParent<NavMeshAgent>();
```

```
    }
    void Update()
    {
DistanceToPlayer = Vector3.Distance(Player.position,Enemy.transform.position);
if(DistanceToPlayer<MaxRange)
{
if (IsChecking == true)
{
IsChecking = false;
blocked = NavMesh.Raycast(transform.position, Player.position, out Hit,
NavMesh.AllAreas);
     if(blocked == false){
RunToPlayer = true;
     FailedChecks = 0;}
if (blocked == true)
{
RunToPlayer = false;
Anim.SetInteger("State",1);
    FailedChecks++;}
StartCoroutine(TimedCheck());
     }}
if(RunToPlayer == true)
{
Enemy.GetComponent<EnemyMove>().enabled = false;
if (DistanceToPlayer>AttackDistance)
{
Nav.isStopped = false;
```

```
Anim.SetInteger("State" , 2);
Nav.acceleration = 24;
Nav.SetDestination(Player.position);
     Nav.speed = ChaseSpeed;}
if (DistanceToPlayer<AttackDistance - 0.5f){
Anim.SetInteger("State",3);
Nav.acceleration = 180;
Vector3 Pos = (Player.position - Enemy.transform.position).normalized;
Quaternion PosRotation = Quaternion.LookRotation(new Vector3(Pos.x, 0 ,Pos.z));
Enemy.transform.rotation = Quaternion.Slerp(Enemy.transform.rotation, PosRotation,
Time.deltaTime * AttackRotateSpeed);}}
 else if(RunToPlayer == false){
Nav.isStopped = true;
       }}
   private void OnTriggerEnter(Collider other)
   {
       if   (other.gameObject.CompareTag("Player"))
       {
RunToPlayer = true;
       }
IEnumeratorTimedCheck()
{
       yield return new WaitForSeconds(CheckTime);
       IsChecking = true;
if(FailedChecks>MaxChecks)
       {
Enemy.GetComponent<EnemyMove>().enabled = true;
```

```
Nav.isStopped = false;
Nav.speed = WalkSpeed;
FailedChecks = 0;}}}
```

EnemyDamage:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class EnemyDamage :MonoBehaviour
{
   public int EnemyHealth = 100;
   public bool HasDied = false;
   private Animator Anim;
   [SerializeField] GameObjectBloodSplatBat;
   [SerializeField] GameObjectBloodSplatAxe;
   [SerializeField] GameObjectBloodSplatKnife;
public void Start()
   {
Anim = GetComponentInParent<Animator>();
BloodSplatBat.gameObject.SetActive(false);
BloodSplatAxe.gameObject.SetActive(false);
BloodSplatKnife.gameObject.SetActive(false);
   }
   void Update()
   {
      if (EnemyHealth<= 0)
```

```
        {
            if (HasDied == false)
            {
Anim.SetTrigger("Death");
HasDied = true;
            }}}
    private void OnTriggerEnter(Collider other)
    {
        if   (other.gameObject.CompareTag("PKnife"))
        {
EnemyHealth -= 10;
BloodSplatKnife.gameObject.SetActive(true);
        }
         if (other.gameObject.CompareTag("PAxe"))
        {
EnemyHealth -= 15;
BloodSplatAxe.gameObject.SetActive(true);
        }
         if (other.gameObject.CompareTag("PBat"))
        {
EnemyHealth -= 20;
BloodSplatBat.gameObject.SetActive(true);
        }
        if   (other.gameObject.CompareTag("PCrossbow"))
        {
EnemyHealth -= 50;
Destroy(other.gameObject, 0.05f);
```

```
    }}}
```

EnemyKilled:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemyKilled :MonoBehaviour
{
   [SerializeField] int ZombieNumber;
   void Start()
   {
if(ZombieNumber == 1)
      {
         SaveScript.Zombie1 = 0;
}
if(ZombieNumber == 2)
      {
         SaveScript.Zombie2 = 0;
      }
if(ZombieNumber == 3)
      {
         SaveScript.Zombie3 = 0;
      }
if(ZombieNumber == 4)
      {
         SaveScript.Zombie4 = 0;
      }
```

```
if(ZombieNumber == 5)
	{
	   SaveScript.Zombie5 = 0;
	}
if(ZombieNumber == 6)
	{
	   SaveScript.Zombie6 = 0;
	}
if(ZombieNumber == 7)
	{
	   SaveScript.Zombie7 = 0;
	}
if(ZombieNumber == 8)
	{
	   SaveScript.Zombie8 = 0;
	}
if(ZombieNumber == 9)
	{
	   SaveScript.Zombie9 = 0;
	}
if(ZombieNumber == 10)
	{
	   SaveScript.Zombie10 = 0;
	}
if(ZombieNumber == 11)
	{
	   SaveScript.Zombie11 = 0;
```

}}}

EnemyMove:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class EnemyMove :MonoBehaviour
{
    private NavMeshAgentNav;
    private  Animator Anim;
    private Transform TheTarget;
    private float DistanceToTarget;
    private int TargetNumber = 1;
    private bool HasStopped = false;
    [SerializeField] float StopDistance = 2.0f;
    [SerializeField] int MaxTargets = 10;
    [SerializeField] float WaitTime = 2.0f;
    private bool Randomizer = true;
  private int NextTargetNumber;
    [SerializeField] Transform Target1;
    [SerializeField] Transform Target2;
    [SerializeField] Transform Target3;
    [SerializeField] Transform Target4;
    [SerializeField] Transform Target5;
    [SerializeField] Transform Target6;
    [SerializeField] Transform Target7;
```

```
    [SerializeField] Transform Target8;
    [SerializeField] Transform Target9;
    [SerializeField] Transform Target10;
    void Start()
    {
        Nav = GetComponent<NavMeshAgent>();
Anim = GetComponent<Animator>();
TheTarget = Target1;
    }
    void Update()
    {
DistanceToTarget = Vector3.Distance(TheTarget.position,transform.position);
if(DistanceToTarget>StopDistance){
Nav.SetDestination(TheTarget.position);
Nav.isStopped = false;
Anim.SetInteger("State",0);
NextTargetNumber = TargetNumber;
}
if(DistanceToTarget<StopDistance)
{
Nav.isStopped = true;
Anim.SetInteger("State",1);
StartCoroutine(Idle());
}}
    void SetTarget()
    {
        if (TargetNumber == 1)
```

```
        {
TheTarget = Target1;
        }
        if (TargetNumber == 2)
        {
TheTarget = Target2;
        }
        if (TargetNumber == 3)
        {
TheTarget = Target3;
        }
        if (TargetNumber == 4)
        {
TheTarget = Target4;
        }
        if (TargetNumber == 5)
        {
TheTarget = Target5;
        }
        if (TargetNumber == 6)
        {
TheTarget = Target6;
        }
        if (TargetNumber == 7)
        {
TheTarget = Target7;
        }
```

```
        if (TargetNumber == 8)
        {
TheTarget = Target8;
        }
        if (TargetNumber == 9)
        {
TheTarget = Target9;
        }
        if (TargetNumber == 10)
        {
TheTarget = Target10;
        }}
IEnumeratorIdle()
    {
        yield return new WaitForSeconds(WaitTime);
if(HasStopped == false)
        {
HasStopped = true;
            if (Randomizer == true)
            {
                Randomizer = false;
TargetNumber = Random.Range(1, MaxTargets);
if(TargetNumber == NextTargetNumber)
{
TargetNumber++;
                if (TargetNumber>= MaxTargets)
                {
```

```
TargetNumber = 1;
            }}
SetTarget();
        yield return new WaitForSeconds(WaitTime);
HasStopped = false;
        Randomizer = true;
        }}}}
```

HealthScript:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class HealthScript :MonoBehaviour
{
   [SerializeField] Text HealthText;
   [SerializeField] GameObjectDeathPanel;
   void Start()
   {
DeathPanel.gameObject.SetActive(false);
HealthText.text = SaveScript.PlayerHealth + "%" ;
   }
  void Update()
   {
     if (SaveScript.HealthChanged == true)
     {
SaveScript.HealthChanged = false;
```

```
HealthText.text = SaveScript.PlayerHealth + "%" ;
    }
    if (SaveScript.PlayerHealth<= 0f)
    {
SaveScript.PlayerHealth = 0;
DeathPanel.gameObject.SetActive(true);
    }
  }}
```

LoadMainMenu:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class LoadMainMenu :MonoBehaviour
{
  void Start()
  {
SceneManager.LoadScene(1);
  }}
```

OptionsMenu:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Rendering.PostProcessing;
```

```
using UnityEngine.SceneManagement;
public class OptionsMenu :MonoBehaviour
{
    [SerializeField] GameObjectVisualsPanel;
    [SerializeField] GameObjectControlsPanel;
    [SerializeField] GameObjectSavePanel;
    [SerializeField] GameObjectMenuPanel;
    [SerializeField] PostProcessLayerMyLayer;
    public Slider LightSlider;
    void Start()
    {
        Cursor.visible = true;
        Time.timeScale = 0;
        VisualsPanel.gameObject.SetActive(true);
        ControlsPanel.gameObject.SetActive(false);
        SavePanel.gameObject.SetActive(false);
        MenuPanel.gameObject.SetActive(false);
    }
    void Update()
    {
        Cursor.visible = true;
        Time.timeScale = 0;
    }
    public void LightValue()
    {
RenderSettings.ambientIntensity = LightSlider.value;
    }
```

```
    public void ReturnToMainMenu()
    {
SceneManager.LoadScene(0);
    }
    public void Visuals()
    {
        VisualsPanel.gameObject.SetActive(true);
        ControlsPanel.gameObject.SetActive(false);
        SavePanel.gameObject.SetActive(false);
        MenuPanel.gameObject.SetActive(false);
  }
    public void Controls()
    {
        VisualsPanel.gameObject.SetActive(false);
        ControlsPanel.gameObject.SetActive(true);
        SavePanel.gameObject.SetActive(false);
        MenuPanel.gameObject.SetActive(false);
    }
    public void Save()
    {
        VisualsPanel.gameObject.SetActive(false);
        ControlsPanel.gameObject.SetActive(false);
        SavePanel.gameObject.SetActive(true);
        MenuPanel.gameObject.SetActive(false);
    } public void Menu()
    {
        VisualsPanel.gameObject.SetActive(false);
```

```
        ControlsPanel.gameObject.SetActive(false);
        SavePanel.gameObject.SetActive(false);
        MenuPanel.gameObject.SetActive(true);
    }
}
```

PlayerAttack:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PlayerAttack :MonoBehaviour
{
    private Animator Anim;
    [SerializeField] GameObjectCrosshair;
    [SerializeField] GameObjectPointer;
    void Start()
    {
        Anim = GetComponent<Animator>();
        Crosshair.gameObject.SetActive(false);
        Pointer.gameObject.SetActive(true);
    }
    void Update()
    {
       if (SaveScript.HaveKnife == true)
       {
          if (Input.GetKeyDown(KeyCode.Mouse0))
          {
```

```
Anim.SetTrigger("KnifeLMB");
            }
            if (Input.GetKeyDown(KeyCode.Mouse1))
            {
Anim.SetTrigger("KnifeRMB");
            }
        }
        if (SaveScript.HaveBat == true)
        {
            if (Input.GetKeyDown(KeyCode.Mouse0))
            {
Anim.SetTrigger("BatLMB");
            }
            if (Input.GetKeyDown(KeyCode.Mouse1))
            {
Anim.SetTrigger("BatRMB");
            }
        }
         if (SaveScript.HaveAxe == true)
        {
            if (Input.GetKeyDown(KeyCode.Mouse0))
            {
Anim.SetTrigger("AxeLMB");
            }
            if (Input.GetKeyDown(KeyCode.Mouse1))
            {
Anim.SetTrigger("AxeRMB");
```

```
        }
    }
    if (SaveScript.HaveGun == true)
    {
        if (Input.GetKey(KeyCode.Mouse1))
        {
      Crosshair.gameObject.SetActive(true);
      Pointer.gameObject.SetActive(false);
        }
        if (Input.GetKeyUp(KeyCode.Mouse1))
        {
      Crosshair.gameObject.SetActive(false);
      Pointer.gameObject.SetActive(true);
        }
    }
    if (SaveScript.HaveCrossbow == true)
    {
        if (Input.GetKey(KeyCode.Mouse1))
        {
      Crosshair.gameObject.SetActive(true);
      Pointer.gameObject.SetActive(false);
        }
        if (Input.GetKeyUp(KeyCode.Mouse1))
        {
      Crosshair.gameObject.SetActive(false);
      Pointer.gameObject.SetActive(true);
} } } }
```

SaveLoad:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class SaveLoad :MonoBehaviour
{
   [SerializeField] GameObjectLoadButton;
   public int DataExists = 10;
   void Start()
   {
      if (LoadButton != null)
      {
        DataExists = PlayerPrefs.GetInt("PlayerHealth",0);
        LoadButton.gameObject.SetActive(false);
          if (DataExists> 0)
          {
                        LoadButton.gameObject.SetActive(true);
          }
      }
   }
   public void LoadGameData()
   {
SaveScript.SavedGame = true;
   }
   public void SaveGame()
   {
```

```
PlayerPrefs.DeleteAll();
PlayerPrefs.SetInt("PlayerHealth" ,SaveScript.PlayerHealth);
PlayerPrefs.SetFloat("BatteryPower", SaveScript.BatteryPower);
PlayerPrefs.SetInt("BottleAmt", SaveScript.Bottle);
PlayerPrefs.SetInt("BatteryAmt",SaveScript.Battery);
PlayerPrefs.SetInt("BulletsClips", SaveScript.Bullet);
PlayerPrefs.SetInt("BulletsAmt",SaveScript.Bullets);
PlayerPrefs.SetInt("ArrowsAmt",SaveScript.Arrows);
PlayerPrefs.SetInt("BottleL",SaveScript.BottlesLeft);
PlayerPrefs.SetInt("AmmoL",SaveScript.AmmoLeft);
PlayerPrefs.SetInt("BatteryL",SaveScript.BatteryLeft);
PlayerPrefs.SetInt("ArrowL",SaveScript.ArrowLeft);
PlayerPrefs.SetInt("Zombie1Alive",SaveScript.Zombie1);
PlayerPrefs.SetInt("Zombie2Alive",SaveScript.Zombie2);
PlayerPrefs.SetInt("Zombie3Alive",SaveScript.Zombie3);
PlayerPrefs.SetInt("Zombie4Alive",SaveScript.Zombie4);
PlayerPrefs.SetInt("Zombie5Alive",SaveScript.Zombie5);
PlayerPrefs.SetInt("Zombie6Alive",SaveScript.Zombie6);
PlayerPrefs.SetInt("Zombie7Alive",SaveScript.Zombie7);
PlayerPrefs.SetInt("Zombie8Alive",SaveScript.Zombie8);
PlayerPrefs.SetInt("Zombie9Alive",SaveScript.Zombie9);
PlayerPrefs.SetInt("Zombie10Alive",SaveScript.Zombie10);
PlayerPrefs.SetInt("Zombie11Alive",SaveScript.Zombie11);
PlayerPrefs.SetInt("Zombie12Alive",SaveScript.Zombie12);
    if (SaveScript.Knife == true)
PlayerPrefs.SetInt("KnifeInv",1);
    if (SaveScript.Bat == true)
```

```
PlayerPrefs.SetInt("BatInv",1);
        if (SaveScript.Axe == true)
 PlayerPrefs.SetInt("AxeInv",1);
        if (SaveScript.Gun == true)
 PlayerPrefs.SetInt("GunInv",1);
        if (SaveScript.Crossbow == true)
 PlayerPrefs.SetInt("CrossbowInv",1);
        if (SaveScript.CabinKey == true)
PlayerPrefs.SetInt("CabinK",1);
        if (SaveScript.HouseKey == true)
PlayerPrefs.SetInt("HouseK",1);
        if (SaveScript.RoomKey == true)
PlayerPrefs.SetInt("RoomK",1);
if(SaveScript.ArrowRefill == true)
PlayerPrefs.SetInt("ArrowR",1);
    }
}
```

SYSTEM TESTING

7. SYSTEM TESTING

7.1 TESTING OBJECTIVES

The purpose of testing is to get errors. Testing is that the method of making an attempt to get each conceivable fault or weakness during a work product. It provides the way to envision the practicality of parts, sub-assemblies, assemblies and/or a finished product. it's the method of travail software package with the intent of guaranteeing that the package meets its necessities associate degreed user expectations and doesn't fail in an unacceptable manner. There are varied sorts of check. every check sort addresses a selected testing demand.

7.2 TESTING TYPES

7.2.1 Unit Testing

Unit Testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. Unit tests perform basic tests at component level and test a specific business process, application and / or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results. Unit testing is usually conducted as part of the combined code and unit test phase of the software life cycle, although it is not uncommon for coding and unit testing to be conducted as to distinct phases.

7.2.2 Integration Testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the components. Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications

7.3 TEST CASES AND RESULTS

S. No	Test Cases	Excepted Output	Actual Output	Status
1	Game	User runs and enters the game. User begins to play the game. When player finds the key or is killed by enemies, the play ends	Game begins and ends successfully	PASS
2	Jump	User presses space key to jump. User moves in “up” direction	Player position changes	PASS

3	Run	User pressed shift key to run. Player gains movement in direction. Player velocity increases.	Player position and velocity changes	PASS
4	Walk	User presses arrow keys or WASD to walk. Player gains movement in direction	Player position changes	PASS
5	Torch light	User presses the key ‘T’ to turn ON the flashlight.	Flashlight is turned ON.	PASS
6	Night vision mode	User presses the key ‘N’ to turn ON the night vision mode.	Night vision mode is turned ON.	PASS
7	Battery	User picks up the battery to regain his energy to use the torch light and night vision mode.	Battery level is increased.	PASS
8	Weapon	The user should search for the weapon and need to equip it.	The weapon is picked up.	PASS
9	Change Weapon	The user can be able to change the weapon by picking up different weapon.	The player changes current weapon.	PASS

10	Reload Weapon	If the user picks up the Ammo, the current weapon ammo will increase accordingly.	The ammo counter is changed.	PASS
11	Axe	User picks up the Axe weapon and it can be used by the player to kill the enemy.	The Axe weapon generates damage to the enemy.	PASS
12	Play Again	Once the player died it asks for play again	User restarts the game	PASS
13	Cross bow	User picks up the Cross bow and equips it to create a crucial damage to the enemy, which will also kill the enemy.	The Crossbow weapon generates damage to the enemy.	PASS
14	Bat	This weapon will find somewhere in the game scenario and the player should equip this to safeguard himself from the enemies.	The Bat weapon generates damage to the enemy.	PASS
15	Gun	User will pick up and equip this weapon from the inventory and the usage of this weapon is	This weapon generates damage to the enemy.	PASS

		to shoot the enemies.		
16	Fire Effect	User fires the weapon, and the fire particle effect is requested from graphics engine. The game displays the effect.	A fire particle effect is displayed.	PASS
17	Blood effect	When the user attacks the enemy with his weapon, the blood effect will be requested from graphics engine and the game will display the effect.	Blood effect is displayed on the screen.	PASS
18	Aiming	User moves the mouse. Objects move according to physics engine and the view of player moves accordingly.	Players' view has been changed	PASS
19	Environment	The game determines user direction and displays the appropriate graphic models.	A 3D terrain is displayed	PASS
20	Player - Environment	Player collides with walls, ground, or any concrete	The velocity of player decreases	PASS

		objects. The player is blocked, and velocity decreases accordingly.		
21	Enemies	User encounters enemies. The game requests the graphic model for enemies. Enemy model is displayed	Enemies are displayed	PASS
22	Enemy - Environment	Enemy collides with walls, ground, or any concrete objects The enemy is blocked, and its velocity decreases accordingly.	The velocity of enemy decreases	PASS
23	Take Damage	Player collides with Enemy. If the player is with Punch weapon, damage is assigned to enemy, otherwise, damage is assigned to player.	The player's health or Enemies health is changed	PASS
24	Die	If the health of the player or enemy reaches to zero, player or enemy dies.	The player dies or the enemy dies	PASS

25	Knife	User picks up the Knife weapon and equips it to create damage to the enemy, which will also kill the enemy.	The Knife weapon generates damage to the enemy.	PASS
26	Exit	Once the player died it asks for exit. If player kills all the enemies, he can exit the game by pressing “esc” on keyboard	User exits game	PASS

Table 7.1 Test Cases and Results

CONCLUSION

8. CONCLUSION

8.1 CONCLUSION & FUTURE ENHANCEMENT

The aim of this project was to create a 3D game which was the First-Person Shooter game with Unity game engine. The requirement is to have a fundamental knowledge about the unity game engine and programming. A game engine is the core of creating a game. The integration of model design, level design is the game engine, which is complex and powerful. The Unity game engine supports visualized design; thus, it is a strong game engine which is suitable for a beginner. However, it is not very easy to learn the Unity game engine well. There are various functions to be realized.

The most important aspect of our game design is to create more new playabilities. This game is a product of a beginner who favors creating a virtual world with his fundamental knowledge. There might be more possibilities for this game to be enhanced with more developed concepts in the future and there will be more updates to enhance the game play of the game.

APPENDICES

9. APPENDICES

A.1 SAMPLE SCREENSHOTS

Fig (9.1.1) Screenshot of environment

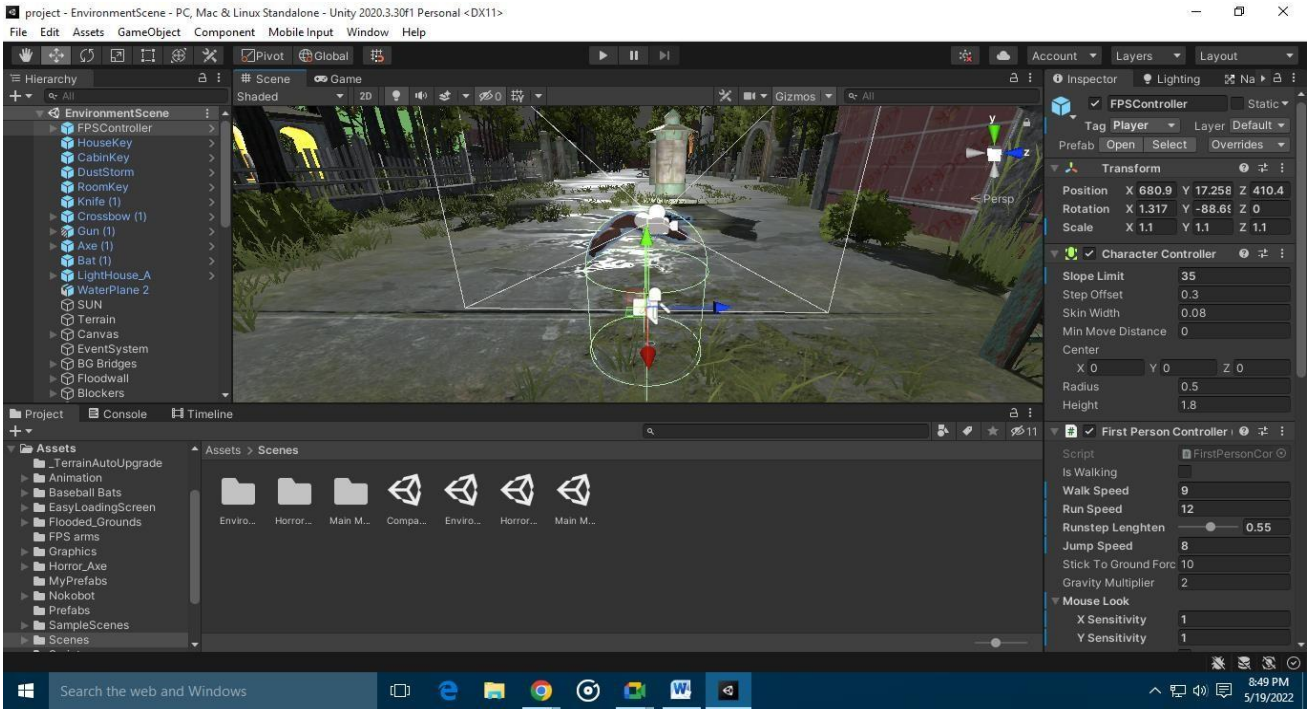


Fig (9.1.2) Screenshot of zombie

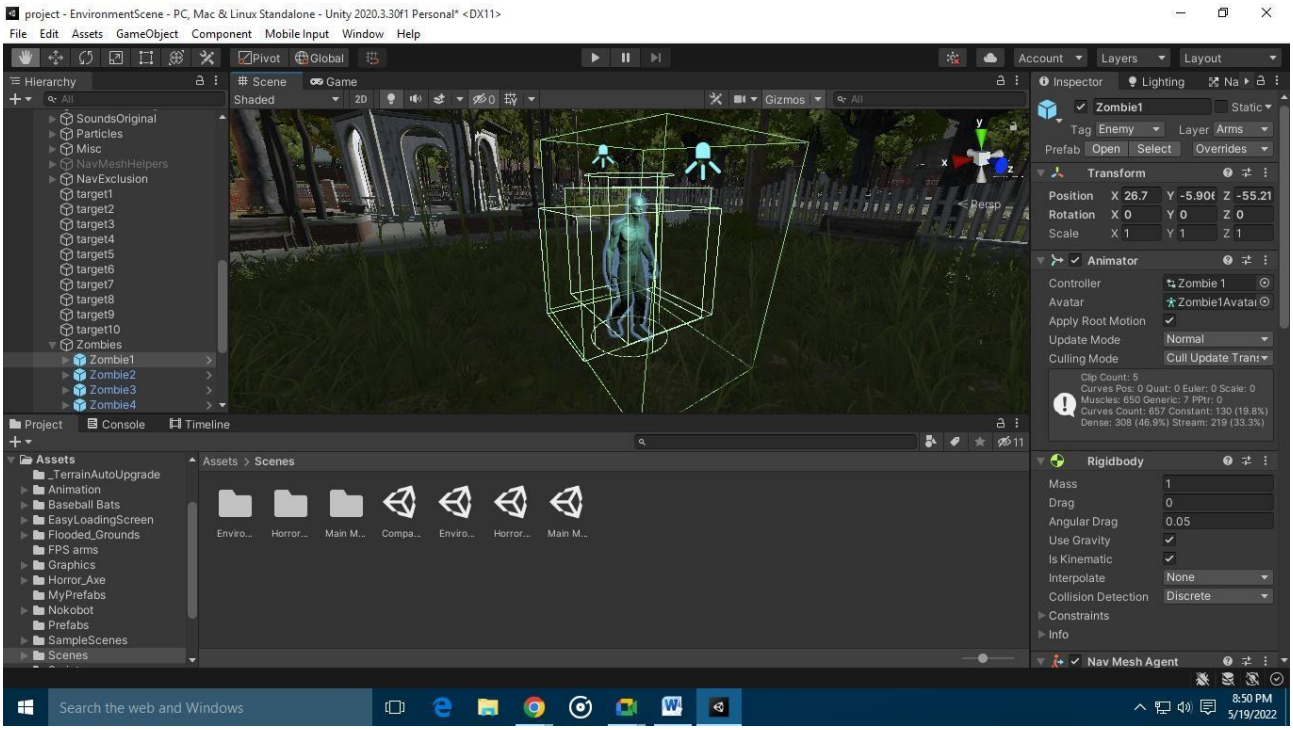


Fig (9.1.3) Screenshot of battery inside a building

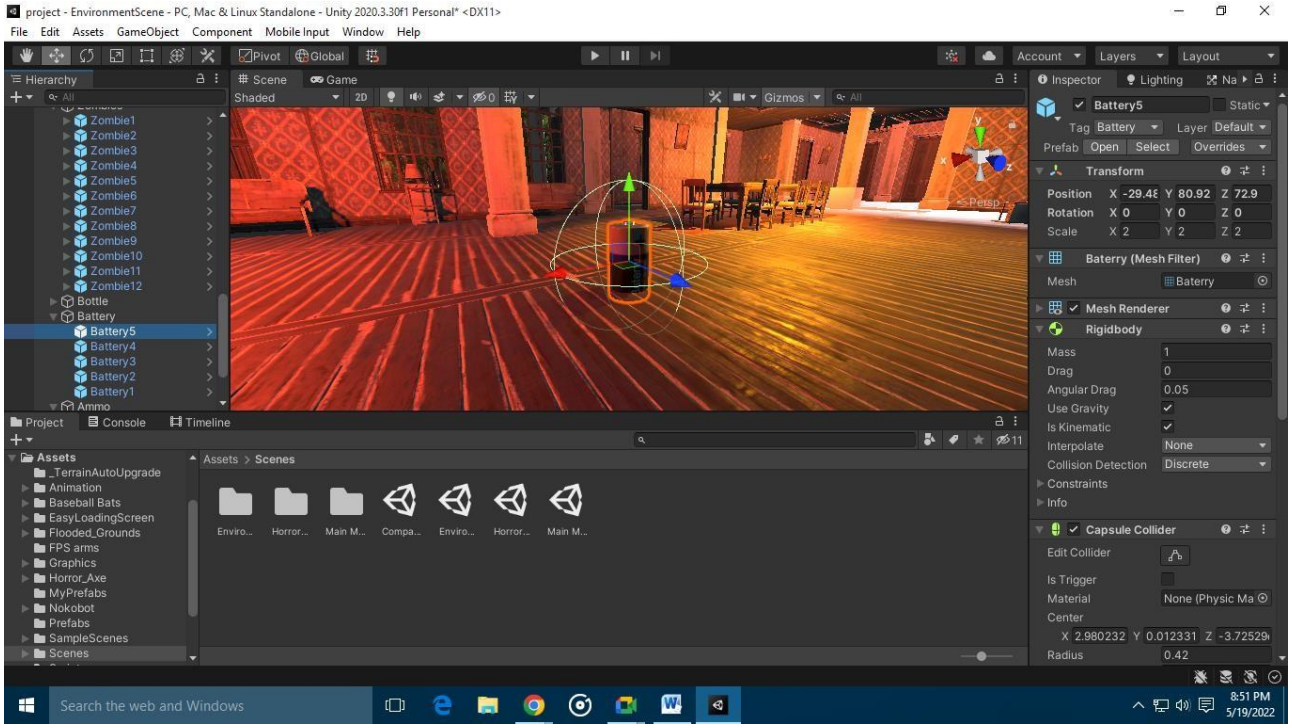


Fig (9.1.4) Screenshot of start menu

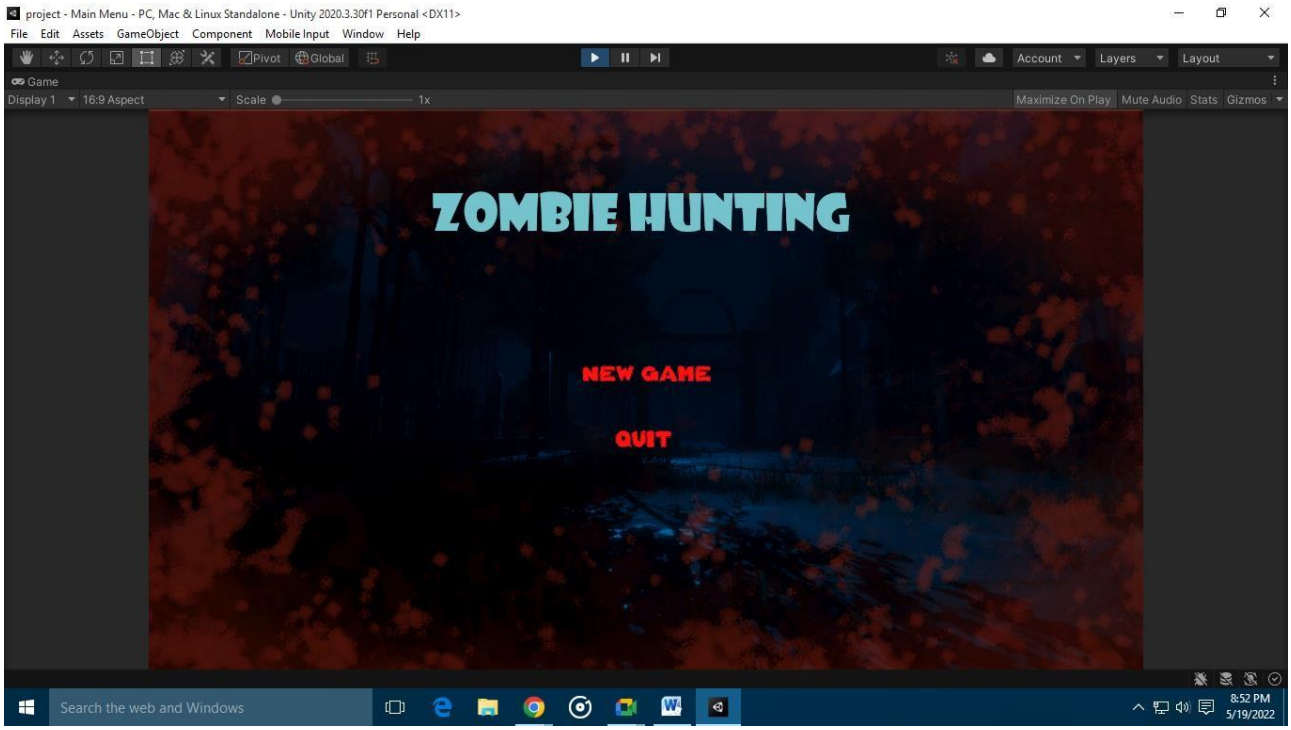


Fig (9.1.5) Screenshot of loading screen

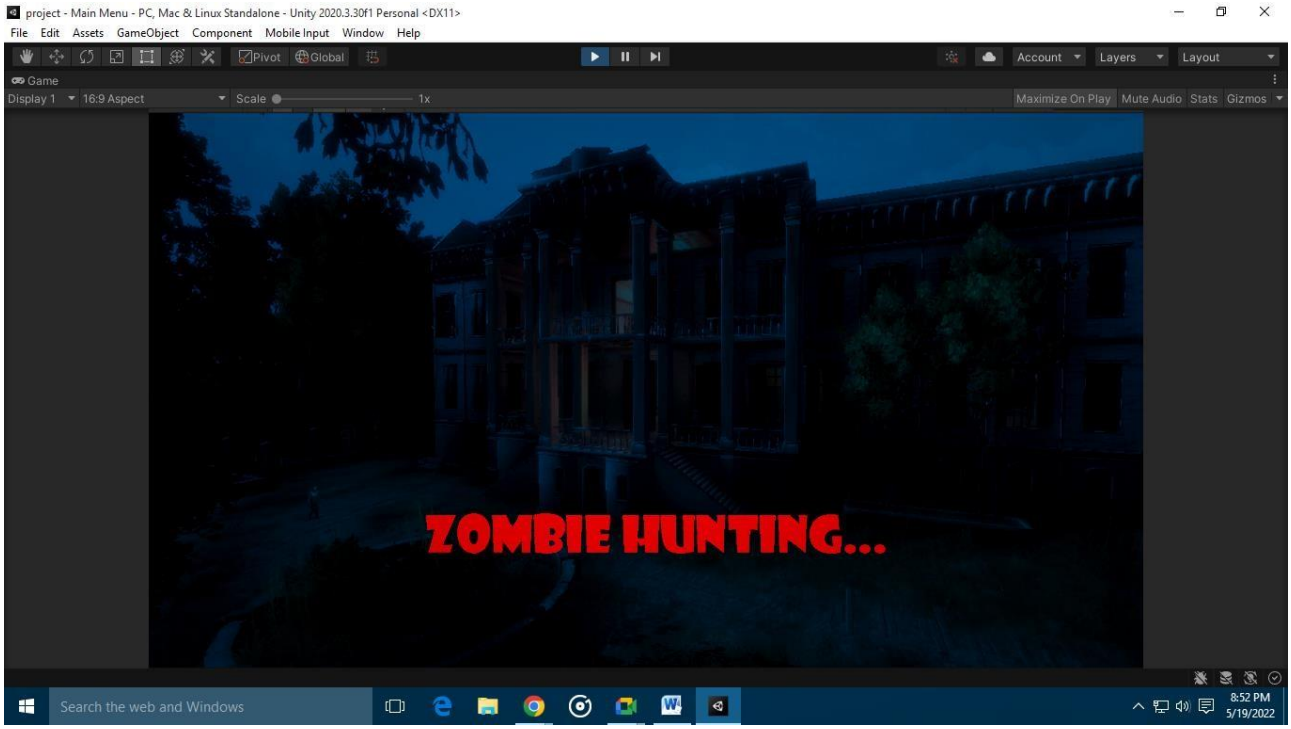


Fig (9.1.6) Screenshot of ending screen

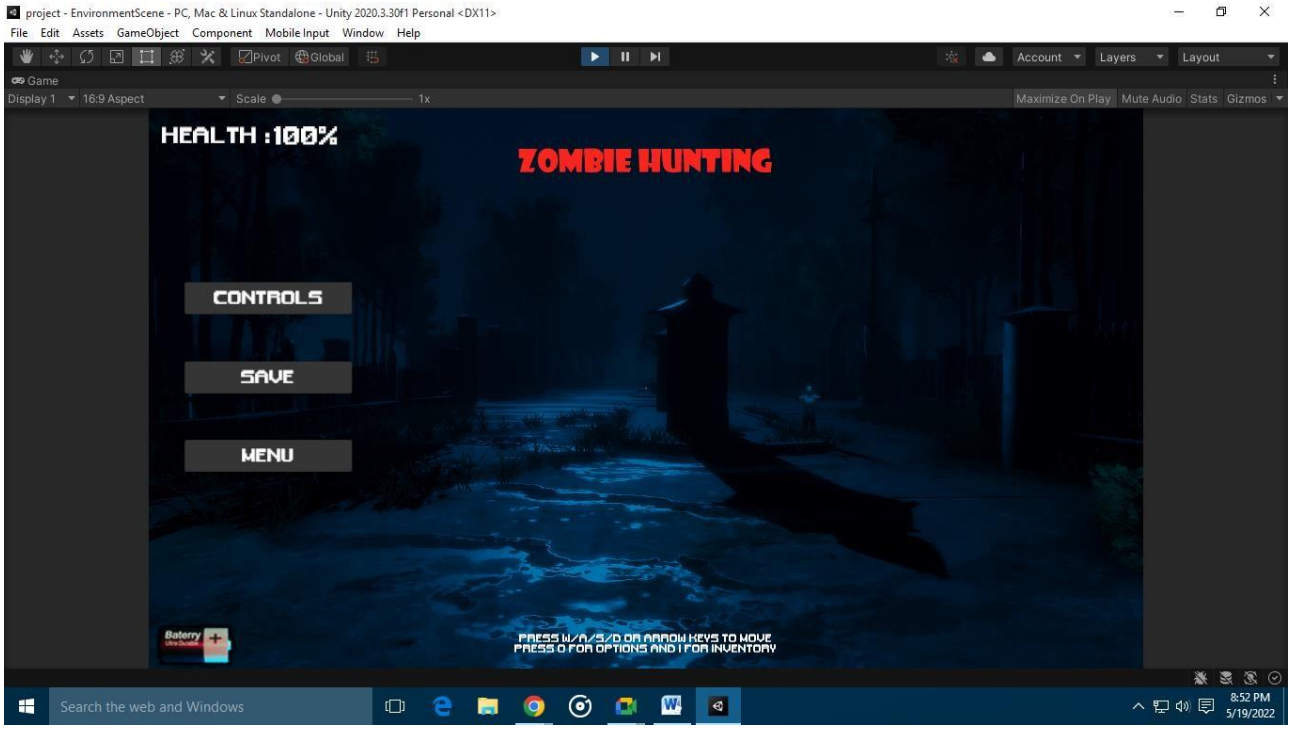


Fig (9.1.7) Screenshot of player inside building

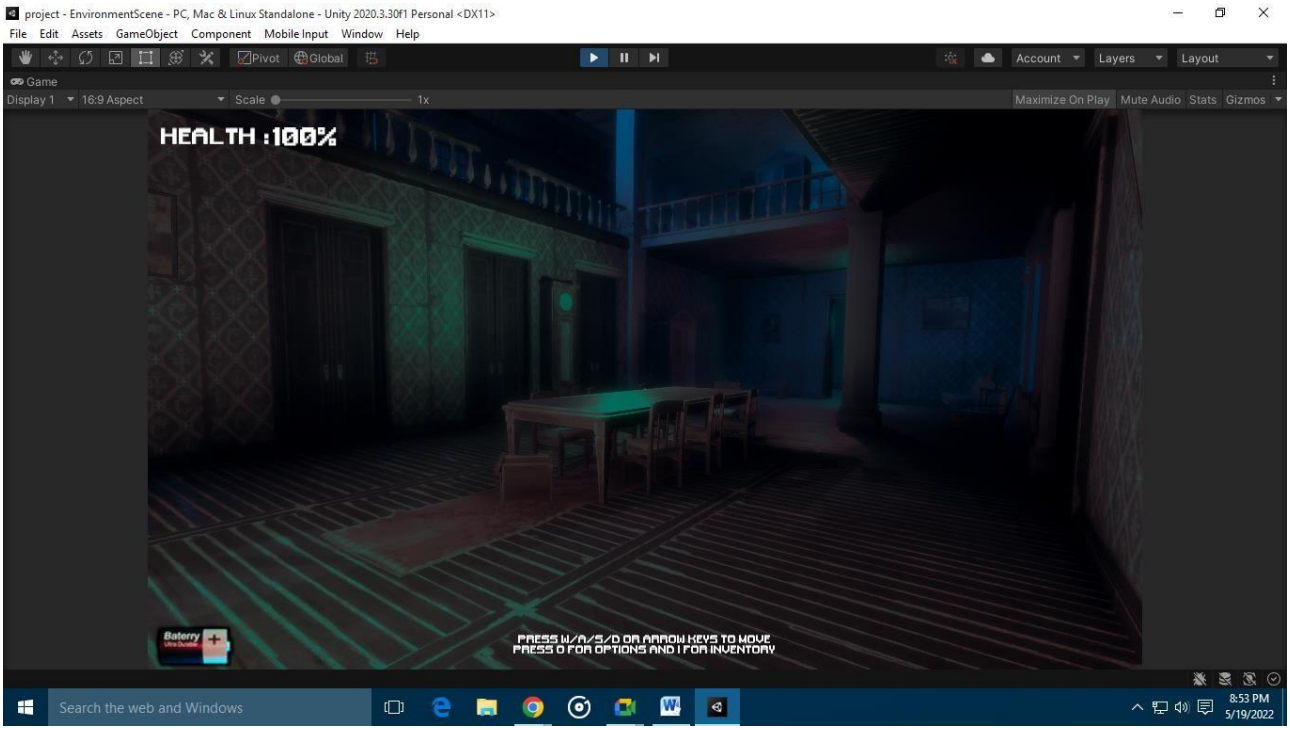
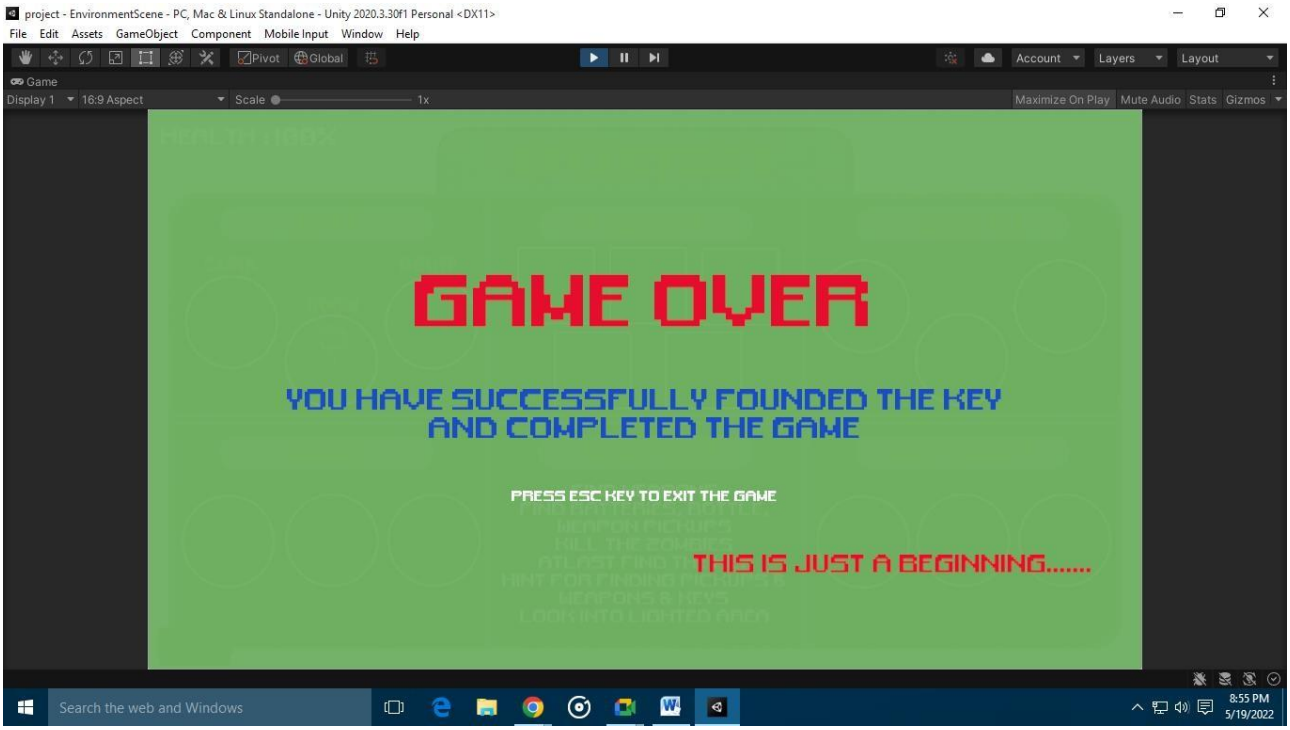


Fig (9.1.8) Screenshot of game over screen



REFERENCES

10. REFERENCES

- Rapid Skill Capture in a First-Person Shooter | IEEE Journals & Magazine | IEEE Xplore
- Applying inexpensive AI techniques to computer games | IEEE Journals & Magazine | IEEE Xplore
- Learning to win in a first-person shooter game | SpringerLink
- Using Games to Study Law of Motions in Mind | IEEE Journals & Magazine | IEEE Xplore
- Challenges and Opportunities in Game Artificial Intelligence Education Using Angry Birds | IEEE Journals & Magazine | IEEE Xplore
- Optimizing Player and Viewer Amusement in Suspense Video Games | IEEE Journals & Magazine | IEEE Xplore
- Lag Compensation for First-Person Shooter Games in Cloud Gaming | SpringerLink
- Adaptive Shooting for Bots in First Person Shooter Games Using Reinforcement Learning | IEEE Journals & Magazine | IEEE Xplore
- An implementation of affective adaptation in survival horror games | IEEE Conference Publication | IEEE Xplore