A Project Report

on

SHOOTING GAME USING UNREAL ENGINE IN C++

Submitted in partial fulfillment of the

requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

Under The Supervision of
Mr. Anupam LakhanPal
Assistant Professor
Department of Computer Science and Engineering

Submitted By

Aman Sharma - 19SCSE1010441

Mansi - 19SCSE1010276

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING /
DEPARTMENT OF COMPUTERAPPLICATION
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
DECEMBER-2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the thesis/project/dissertation, entitled **"SHOOTING GAME USING UNREAL ENGINE IN C++"** in partial fulfillment of the requirements for the award of the **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of JULY-2021 to DECEMBER-2021, under the supervision of **Mr. Anupam Lakhanpal**, Assistant Professor, Department of Computer Science and Engineering, Galgotias University, Greater Noida.

The matter presented in the project has not been submitted by me/us for the award of any other degree of this or any other places.

Aman Sharma - 19SCSE1010441

Mansi - 19SCSE1010276

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Supervisor Name

Mr. Anupam Lakhanpal
(Assistant Professor)

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of **Aman Sharma (19SCSE1010441), Mansi (19SCSE1010276)** has been held on _____ and his/her work is recommended for the award of BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: December, 2021

Place: Greater Noida

ABSTRACT

The goal of this project is to prove the development of a videogame using Unreal Engine, based on an agile methodology that is viable in an economic, quick and sustainable way. This methodology has four stages that are: preproduction, production, testing and postproduction that were advantageous to finish the project on time. To achieve this, we have designed and developed an action platform game following the previously mentioned stages. In conclusion, we achieve to prove the applicability of the four stages methodology since we made a high quality game in a short period of time, using limited resources. We are going to develop a game using unreal engine in C++. This will be 3D game. Our character will be full animated third person character. We will add sorts of enterprises to add movement system and aiming system, animation on the top of all this. We are going to add shooting mechanics, we are going to have deaths/lives. We will kill of enemy AI with shooting mechanics and also have health for ourselves. Finally, we're going have win/lose conditions. So, if I get shot to pieces, the game's gonna notice that and it's going to restart the mental for and display this handy message telling us what's going on. The same happens if we managed to kill off all the enemies in the level we win and get a similar message.

TABLE OF CONTENTS

List Of Figures

CHAPTER: 1

Introduction About Project

1.1) Introduction

The goal of this project is to prove the development of a videogame using Unreal Engine, based on an agile methodology that is viable in an economic, quick and sustainable way. This methodology has four stages that are: preproduction, production, testing and postproduction that were advantageous to finish the project on time. To achieve this, we have designed and developed an action platform game following the previously mentioned stages. In conclusion, we achieve to prove the applicability of the four stages methodology since we made a high quality game in a short period of time, using limited resources.

1.2) Literature Reviews/Comparative study:

The games industry has had much growth in recent years. It is a great industry to get involved in as it allows creativity, innovation and freedom for developers and hobbyists. They get a chance to experiment with all forms of media including sound design, environment design and programming. As there has been an explosion of the new ‘Indie’ market of games, the gaming industry now allows smaller-scale games to be developed and released more freely than in the past. Today it has become so easy to create a game. A large studio is no longer needed and the freely available tools make it so simple to create an idea from the comfort of your own home. Indie games show more innovation and developers are willing to take risks on their games. ‘Indie’ games also have a large market base with many of the games being released on PC, Android etc. 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 a FPS in a horror style environment

CHAPTER: 2

Requirements, Feasibility and Scope Objective

2.1) OBJECTIVE:

The goal of this project is to prove the development of a videogame using Unreal Engine, based on an agile methodology that is viable in an economic, quick and sustainable way. The purpose of this project is to develop a first person shooter game with good graphics, audio and animations made with basic technologies and a low budget. I wanted to make a game that was easy to install and show what can be done with just a bit of time and effort.

2.2) BACKGROUND:

UNREAL ENGINE

VISUAL STUDIO

C++

A-Star Pathfinding Algorithm

2.3) Requirements

2.3.1) Functional Requirements

This section shows the basic needs for the game in order to have the game up and running in the time that is required.

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

ii. **Terrain:** The game should have a terrain where the characters can exist. It should have realistic physics including gravity, wind and borders.

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

iv. **Weapon:** The player must be provided with a weapon in order to kill and damage an enemy.

v. **Play Game:** The player should be able to select the play game option from the main menu. The play game button will give a short background story (which the player can skip if they wish) before the game will start.

vi. **Controls:** The player should be able to select the controls option from the main menu which will show them the buttons used to control the player.

vii. **Volume:** The player should be able to adjust the volume of the background audio. A slider is presented on the main menu screen.

viii. **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.

ix. **Game Completed:** The player should be able to complete the game and receive recognition from the game that it is over. The player should complete the game once they have collected their soul after defeating the boss. When the player walks through the door and into the soul it will load the game over screen which will then bring them back to the main menu where they can decide to play again or quit.

x. **Pause Game:** The player should be able to pause 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.

xi. **Game:** The player should be able to quit the game and stop playing. To prevent accidental exit, the game will ask is the user sure. They can click yes to exit the game or no to continue.

xii. **Fight/Defeat Enemies:** The player should be able to shoot and kill enemies in the game. These enemies will make the game more challenging as the

player will need to pass them. The player can fight them or attempt to run away from them.

xiii. **Health System:** The player must be able to lose health when attacked by enemies. When the player has run out of health they will have to respawn which will bring them back to the start of the game. Health will be restored to max health and any enemies they have already killed will remain in that state.

xiv. **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.

xv. **Aim And Sight:** The player should be able to aim the gun and look down the sight of the gun. This is achieved by pressing the right mouse click.

xvi. **Intelligent Enemy AI:** The game requires multiple enemies which attack the player. When the user has defeated these enemies, a boss character will be introduced which will be stronger and harder to defeat. This boss character will also be more intelligent and it will provide the player with a challenge.

2.3.2) User requirements:

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 XP 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 application. When the game is compiled, it is built into an executable file which is separate from Unreal Engine which will benefit the user as they will not be required to download and install

Unreal Engine if they want to play the game. The game can also be compiled to run on web browsers. The user must also have a computer, key board, mouse and speakers/headphones to play the game.

2.3.3) Data requirements

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

2.3.4) Environmental requirements

In order to play the game, the users will need a Windows, Linux or Mac operating system environment but the game was designed to use on the Windows platform. There's No Place Like Home 14 This application will run on a computer or laptop capable of running a modern version of the Windows operating system (must be Windows XP or higher)

2.3.5) Usability requirements

The menu for the game should be easily accessible and visible, and clearly laid out to provide a clear understanding of the game menu. It should be easy for the user to accomplish a task and navigate through the menu. The game should not overwhelm the users and the UI should be clearly laid out and visually appealing.

2.3.6) Performance/Response time requirement

It is important to ensure that the game does not suffer any graphical or performance problems as this can lead to a frustrating game play. The game needs to run at a consistent frame rate to ensure that the graphics and animations remain smooth. The response time of the game should be immediate so that when the player initiates an action in the game, the results will be immediate. The player

will need precision for jumping and attacking, so the response time for input methods must be very quick.

Chapter: 3

Analysis, Activity Time Schedule **(changes)**

Sr. No.	Activity/Objective	Duration
1.	Planning and assigning	10 Days
2.	Data Assembling	10 Days
3.	Programming	20 Days
4.	Testing (2-3 round)	20 Days

Quality Plan

The goal of the plan is to ensure that game is an error free game. Throughout the development, testers will be brought in to test the mechanics and the levels of the game to ensure there are no errors in the game.

We will divide quality testing into three main phases:

- Module testing will perform during coding by using debug messages to check that the written code produces wanted results. An important requirement is that the code will compile with zero bugs.
- Integration testing will perform after finish module testing in order to validate if each module can work fine with each other.
- System testing includes two phases: functional testing and usability testing. These will perform after the product reaches its final version. During functional test phase, the tester will test if the product meets the game requirements. The usability test will perform to understand how easy it is to learn to play the game. Any person out of the team members will perform this test by playing the game.

Chapter:4

Game Design And Mechanics

Framework Designing assists the examiner with understanding the usefulness of the framework and models are utilized to speak with clients. Various models present the framework from alternate points of view. External viewpoint showing the framework's specific circumstance or climate. Behavioural point of view showing the conduct of the framework. Structural point of view showing the framework or information engineering

4.1) USE CASE DIAGRAM:

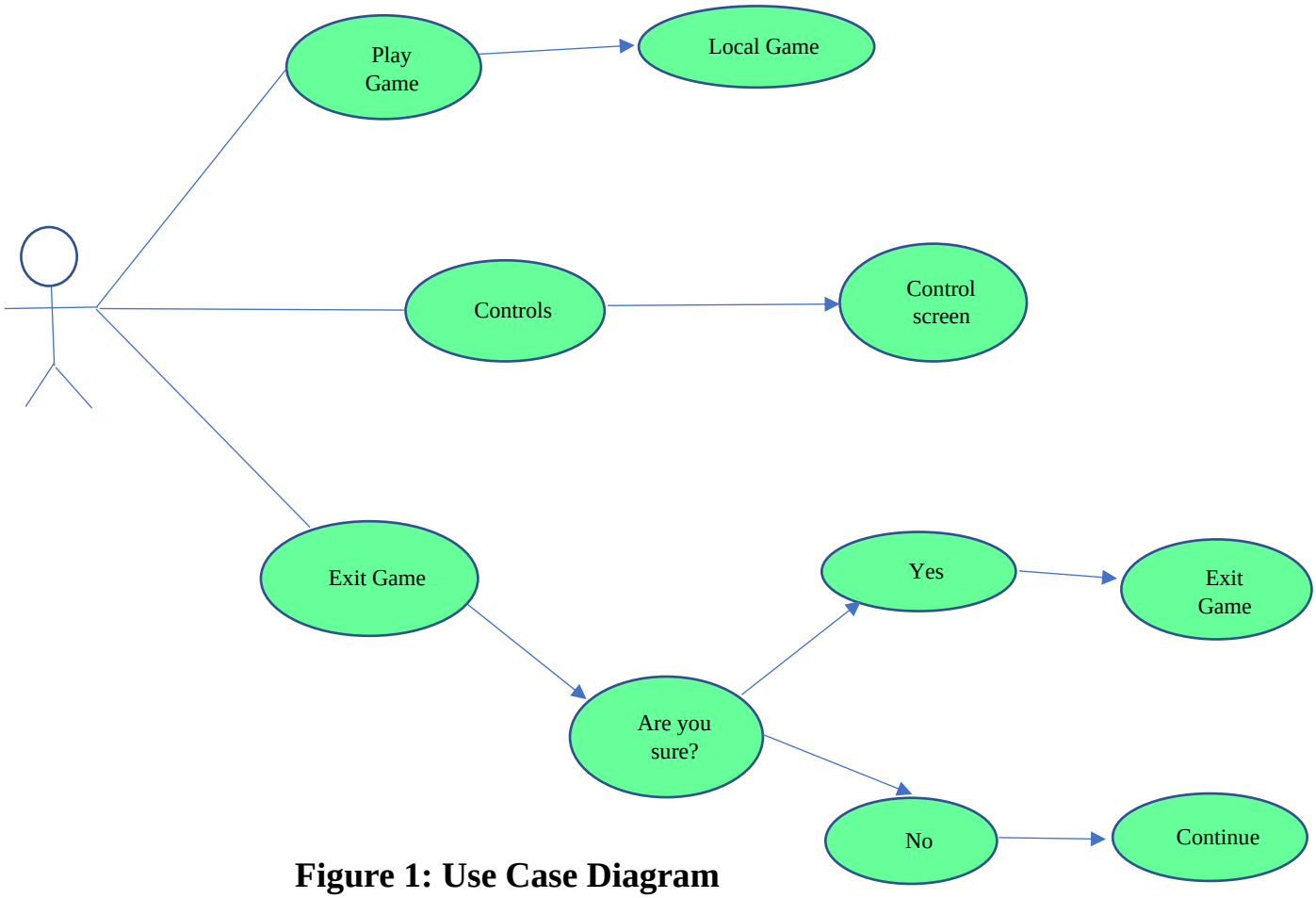


Figure 1: Use Case Diagram

When the player starts up the game the main menu will be displayed. The player will be presented with 3 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, if they select no, they can continue as normal.

4.2) Game Mechanics

4.2.1) Game Flow

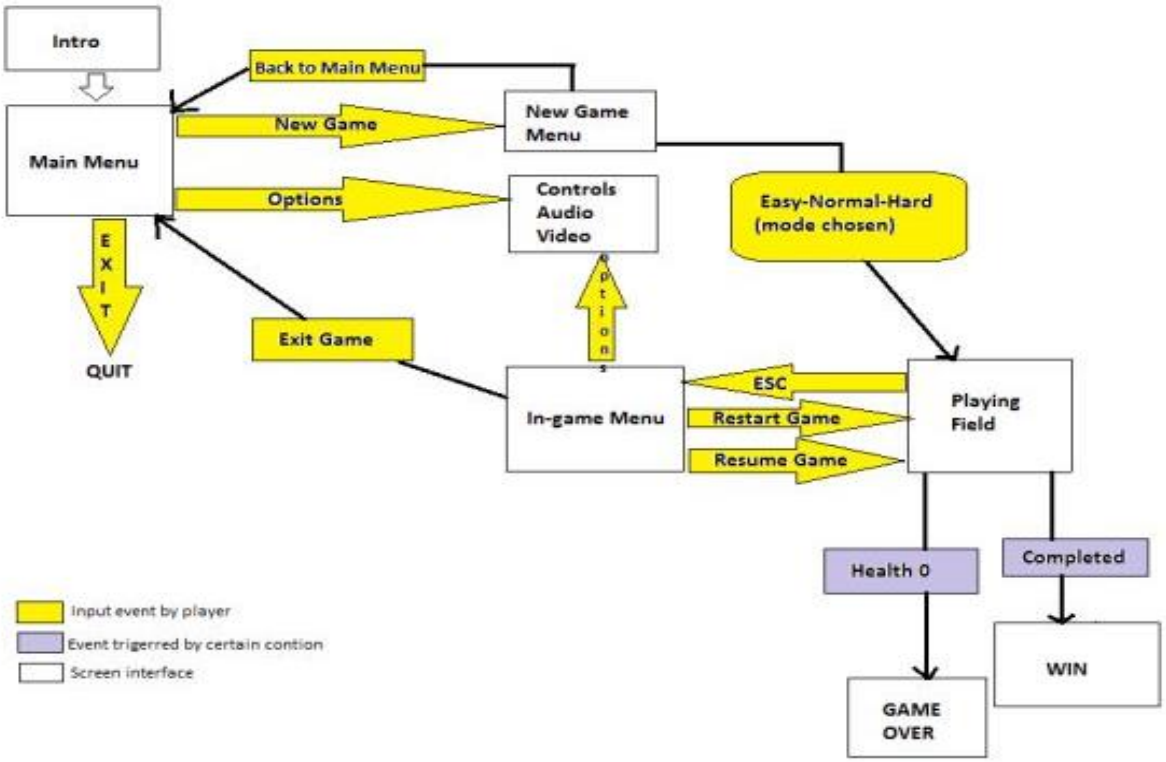


Figure:1 Game Flow

4.2.2) Game Controls

The game will utilize the mouse and keyboard for input. Here is the list how to control the game.

Movement

- Left - A
- Right - D
- Forward - W
- Backward - S
- Jump – Space
- Crouch-C

Actions

- Looking/Aiming – Mouse
- Fire Weapon- Left Mouse
- Use Equipment-Right Mouse
- Change weapon - numb 1-9
- Reload - R
- Use – E

4.2.3) Winning/Losing

The object of the game is to kill all enemy before they kill the player. The player has only one life. When his/her health reduces to zero or lower, the player will die. When the player dies, he/she will lose the game. If the player shoots all enemies before dying, the player wins the game.

4.2.4) Game Modes

The game has a design with only single-player mode. In the single-player mode, one can find three sub-modes: easy, normal and hard. Depends on sub-modes, the number of enemies and pick-ups will vary. When the game becomes harder, the number of enemies will increase and the number of power-ups will reduce. The table below provides information on the number of power-ups and enemies in each mode.

Mode	No. Of Enemies
Easy	10
Medium	15
Hard	20

Table 1: Modes

Chapter: 5

Implementation and Testing

5.1) Setting up your project:

Steps

- 1.1 - Project Setup
- 1.2 - Opening the Project in Visual Studio
- 1.3 - Adding Log Messaging
- 1.4 - Compiling the Project
- 1.5 - Setting the Default Game Mode

5.2) Implementing your Character

Steps

- 2.1 - Making a New Character
- 2.2 - Setting up Axis Mapping
- 2.3 - Implementing Character Movement Functions
- 2.4 - Implementing Mouse Camera Control
- 2.5 - Implementing Character Jumping
- 2.6 - Adding a Mesh to Your Character
- 2.7 - Changing the Camera View
- 2.8 - Add a First Person Mesh to Your Character

5.3) Implementing Projectiles

Steps

- 3.1 - Adding Projectiles to Your Game
- 3.2 - Implementing Shooting
- 3.3 - Setting Up Projectile Collision and Lifetime

- 3.4 - Getting Projectiles to Interact with the World
- 3.5 - Adding Crosshairs to Your Viewport

5.4) Adding Character Animation

Steps

- 4.1 - Animating Your Character
- 4.2 - Setting Up Your Event Graph
- 4.3 - Adding an Animation State Machine
- 4.4 - Adding Animation Transition States
    - 4.4.1 - Add Idle to/from Run Transitions
    - 4.4.2 - Add Idle to Jump Start Transition
    - 4.4.3 - Add Run to Jump Start Transition
    - 4.4.4 - Add Jump Start to Jump Loop Transition
    - 4.4.5 - Add Jump Loop to Jump End Transition
    - 4.4.6 - Add Jump End to Idle Transition
- 4.5 - Associating Animation and Character Blueprint

5.5) Weapon Fire System

The weapon is switched to its firing state on the local client and server (via RPC calls). DetermineWeaponState() is called in StartFire()/StopFire() which performs some logic to decide which state the weapon should be in and then calls SetWeaponState() to place the weapon into the appropriate state. Once in firing state, the local client will repeatedly call HandleFiring() which, in turn, calls FireWeapon(). Then it updates ammo and calls ServerHandleFiring() to do the same on the server. The server version is also responsible for notifying remote clients about each fired round via the BurstCounter variable.

5.6) Instant-Hit Weapon Fire

Instant-hit detection is used for fast firing weapons, such as rifles or laser guns. The basic concept is that when the player fires the weapon, a line check is performed in the direction the weapon is aimed at that instant to see if anything would be hit.

This method allows high precision and works with Actors that do not exist on server side (e.g., cosmetic or torn off). The local client performs the calculations and informs the server of what was hit. Then, the server verifies the hit and replicates it if necessary.

In FireWeapon(), the local client does a trace from the camera location to find the first blocking hit under the crosshair and passes it to ProcessInstantHit(). From there, one of three things happens:

- The hit is sent to the server for verification (ServerNotifyHit() --> ProcessInstantHit_Confirmed()).
- If the hit Actor does not exist on server, the hit is processed locally (ProcessInstantHit_Confirmed()).
- If nothing was hit, the server is notified (ServerNotifyMiss()).

Confirmed hits apply damage to the hit Actors, spawn trail and impact effects, and notify remote clients by setting data about the hit in the HitNotify variable. Misses just spawn trails and set HitNotify for remote clients, which look for HitNotify changes and perform the same trace as the local client, spawning trails and impacts as needed.

The instant-hit implementation also features weapon spread. For trace/verification consistency, local client picks a random seed each time FireWeapon() is executed and passes it in every RPC and HitNotify pack.

5.7) Projectile Weapon Fire

Projectile fire is used to simulate weapons that fire rounds which are slower moving, explode on impact, affected by gravity,

1. These are cases where the outcome of the weapon fire cannot be determined at the exact instant the weapon is fired, such as launching a grenade. For this type of weapon, an actual physical object, or *projectile*, is spawned and sent moving in the direction the weapon is aimed. A hit is determined by the projectile colliding with another object in the world.

For projectile fire, the local client does a trace from camera to check what Actor is under the crosshair in FireWeapon(), similar to the instant-hit implementation. If the player is aiming at something, it adjusts the fire direction to hit that spot and calls ServerFireProjectile() on the server to spawn a projectile Actor in the direction the weapon was aimed.

When the movement component of the projectile detects a hit on the server, it explodes dealing damage, spawning effects, and tears off from replication to notify the client about that event. Then, the projectile turns off collision, movement, and visibility and destroys itself after one second to give client time for replication update.

On clients, explosion effects are replicated via OnRep_Exploded().

5.8) Player Inventory

The player's inventory is an array of AShooterWeapon references stored in the Inventory property of the player's Pawn (AShooterCharacter). The currently equipped weapon is replicated from the server, and additionally, AShooterCharacter stores its current weapon locally

in CurrentWeapon property, which allows the previous weapon to be un-equipped when a new weapon is equipped.

When the player equips a weapon, the appropriate weapon mesh - first-person for local, third-person for others - is attached to the Pawn and an animation is played on the weapon. The weapon is switched to the equipping state for the duration of the animation.

5.9) Player Camera

In first-person mode, the Pawn's mesh is hard-attached to the camera so that the arms always appear relative to the player's view. The downside of this approach is that it means the legs are not visible in the player's view, since the entire mesh rotates to match the camera yaw and pitch.

The basic flow of the camera update is:

- AShooterCamera::UpdateCamera() is executed each tick.
- APlayerCamera::UpdateCamera() is called to update the camera rotation based on the player's input.
- AShooterCharacter::OnCameraUpdate() is called to perform the calculations necessary to rotate the first person mesh to match the camera.

When the player dies, it switches to a *death* camera that has a fixed location and rotation set in the AShooterPlayerController::PawnDied()

1. This function calls AShooterPlayerController::FindDeathCameraSpot(), which cycles through several different locations and uses the first one not obstructed by the level's geometry.

5.10) Menu System

The menu system is created using the Slate UI framework. It consists of **menus**, **menu widgets**, and **menu items**. Each menu has a single menu widget (SSHooterMenuWidget) that is responsible for layout, internal event handling, and animations for all of the menu items. Menu items (SSHooterMenuItem) are compound objects that can perform actions and contain any number of other menu items. These can be as simple as a label or button or "tabs" that contain complete submenus made up of other menu items. This menu can be operated using a keyboard or controller, but there is only limited mouse support at this time.

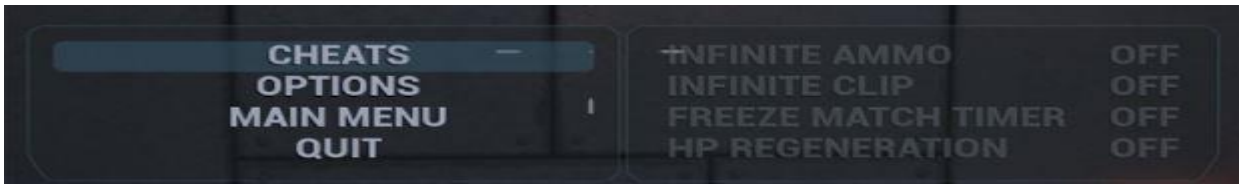
Each menu is *constructed* via the Construct() function, which adds all of the necessary menu items, including sub-items, and attaches delegates to them where necessary. This is done using the helper methods - AddMenuItem(), AddMenuItemSP(), etc. - defined in the MenuHelper namespace in the SShooterMenuWidget.h file.

Navigation to previous menus is done using an array of shared pointers to menus and is stored in the MenuHistory variable of the menu widget. MenuHistory acts like stack to hold previously entered menus and makes it easy to go back. By using this method, no direct relationship is created between menus and the same menu can be reused in different places if necessary.

Animations are performed using interpolation curves defined in SShooterMenuWidget::SetupAnimations(). Each curve has start time, duration, and interpolation method. Animations can be played forward and in reverse and their attributes can be animated at a specific time using GetLerp(), which returns a value from 0.0f to 1.0f. There are several different interpolation methods available, defined in ECurveEaseFunction::Type in SlateAnimation.h.

Main Menu

The main menu is opened automatically when the game starts by specifying the ShooterEntry map as the default. It loads a special GameMode, AShooterGameMode, that uses the AShooterPlayerController_Menu class which opens the main menu by creating a new instance of the FShooterMainMenu class in its PostInitializeComponents() function.



In-Game Menu

The in-game menu is created in the PostInitializeComponents() function of the AShooterPlayerController class, and opened or closed via the OnToggleInGameMenu() function.

5.11) SCREENSHOTS of the Game:


You Lose!
Level will restart in 5 seconds...

Chapter: 6

Future Scope and Limitations of the project

6.1 FUTURE SCOPE:

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 multiplayer option could also be introduced which would expand the game. It would have to be published to a server where users can play as a team and help each other in the battle. A difficulty option could also be put in so users stay challenged and interested. 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.

6.2 Limitation:

This game is not available in multiplayer mode.

6.3 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. 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.

REFERENCES:

1. https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/CPPTutorials/FirstPersonShooter/
2. Apa.org. (2016). [online] Available at: http://www.apa.org/monitor/2014/02/videogame.aspx [Accessed 4 May 2016].
3. Rival{Theory}. (2016). Features - Rival{Theory}. [online] Available at: http://rivaltheory.com/rain/features/ [Accessed 5 May 2016].
   star, D. (2016).
4. Difference and advantages between dijkstra & A star. [online]Stackoverflow.com. Available at: http://stackoverflow.com/questions/13031462/difference-and-advantagesbetween-dijkstra-a-star [Accessed 5 May 2016].
5. YouTube. (2016). CodersExpo. [online] Available at: https://www.youtube.com/user/CodersExpo [Accessed 2016].
6. Rival{Theory}. (2016). Features - Rival{Theory}. [online] Available at: http://rivaltheory.com/rain/features/ [Accessed 5 May 2016].
7. Arongranberg.com. (2016). A* Pathfinding Project. [online] Available at: http://arongranberg.com/astar/ [Accessed 12 Mar. 2016].