University of Mauritius

Software Engineering Year 1 Group D3 (2022)

Software Design Fundamentals and Programming - SIS 1040Y

NO ESCAPE (3D GAME)

Presented by:

Aaishane Shanto 2117162

Tavish Kumar Imrit 2117266

Deesha Beerachee 2117144

Mungur Muhammad Ismaail Jaabir 2117149

Date submitted: 13 July 2022

Acknowledgement

This assignment would not have been accomplished without the support of our lecturers namely, Mrs Zarine Cadersaib and Mr Somveer Kishnah.

We all would like to express our gratitude to the two lecturers above to allow us to create an amazing game in 3D and provided us with invaluable guidance while evaluating our work and giving useful criticism.

Taking into account the deadline, we each planned our tasks that has to be done effectively to complete the assignment.

Abstract

In this document, the 3D application of our game namely NO ESCAPE is described.

The description has been outline, the various difficulties that occurred during development, as well as the goals and purposes for selecting this particular style of game.

To help people reading this document better understand the game, this report includes numerous figures and tables.

Additionally, testing were conducted to ensure that the program functions correctly. We have also included links to the various tutorials and sources that have assisted us in creating this terrific game.

Table of contents:

Acknowledgment 2

Abstract 2

List of tables 6

List of figures 7

1. Introduction

    1.1. Introduction 8

    1.2. Problem Statement 8

    1.3. Aims & Objectives 9

    1.4. Scope 9

    1.5. Distribution of tasks 10

2. Background Study

    2.1. Basic Concepts 11

    2.2. Existing Applications 12

    2.3. Potential Tools 15

    2.4. Summary of Findings 18

3. Analysis

    3.1. Proposed System – Brief Description of Game NO ESCAPE 19

    3.2. Functional Requirements 20

    3.3. Non-Functional Requirements 21

    3.4. Use Case Diagrams 22

    3.5. Tools Chosen 22

4. Design

4.1. Architectural Design 23

4.2. UI Design 25

4.3. Program Designs 26

5. Implementation

5.1. System Requirements 28

5.2. Component: Dependencies for every .h and .cpp files 28

5.3. Component: Character Movement 32

5.4. Component: Character Aiming and firing Movement 36

5.5. Component: Camera 38

5.6. Component: Crosshair 41

5.7. Component: Item and Weapon interpolation 45

5.8. Weapon 55

5.9. Weapon rarity 64

5.10. Enemy movement 70

5.11. Enemy attack 75

5.12. Enemy death 75

5.13. Enemy health 79

5.14. Character health 81

6. Testing

6.1 Crosshair spreading 85

6.2 Crouching aiming 86

6.3 HUD Item inventory 86

6.4 Datable to change weapon rarity & damage 88

6.5 Weapon equip 89

6.6 Enemy Movement 90

6.7 Enemy attack and death 91

6.8 Health HUD and damage 92

7. Conclusion

7.1. Achievements 93

7.2. Challenges and difficulties encountered 94

8. References

8.1. References 94

List of tables:

Table 1: Distribution of table

Table 2: Component tested –Crosshair spreading

Table 3: Component tested – Crouching aiming

Table 4: Component tested – HUD Item inventory

Table 5: Component tested – Datable to change weapon & damage

Table 6: Component tested – Weapon equip

Table 7: Component tested – Enemy movement

Table 8: Component tested – Enemy attack and death

Table 9: Component tested – Health HUD and damage

List of figures:

Figure 1: Resident Evil – Image 1

Figure 2: Resident Evil – Image 2

Figure 3: Days Gone – Image 1

Figure 4: Days Gone – Image 2

Figure 5: Call of Duty: World at War – Zombies– Image 1

Figure 6: Call of Duty: World at War – Zombies– Image 2

Figure 7: Use case diagram for game

Figure 9: Village Image 1

Figure 10: Village Image 2

Figure 11: Village Image 3 with fog

Figure 12: Main menu

Figure 13: Credits screen

Figure 14: Flowchart for main menu screen

Figure 15: Flowchart for pause menu screen

1. INTRODUCTION

1.1 Introduction

No escape is a third-person shooter game developed in a 3d environment as an assignment for module software fundamentals and programming. It compromises elements that we have been taught and self-learning.

This section offers a summary of:

- The problem we have selected and how the game's creation intended to solve it.

- The aims and objectives of the game.

- The scope of the game.

- The distribution of tasks in the development of this program.

1.2 Problem Statement

In this era, people are being increasingly publicised to top-notch technology, particularly in the gaming industry. As the video game business is flourishing, so has the number of games that requires a bulk amount of storage and high-end performing devices. NO ESCAPE is a game created for users that are or above 16 years of age. Players who cannot afford adequate time or money have unfavourable experience as a result of these issues. In addition, many games need a significant amount of time to grind in order to progress, and they are considerably more sophisticated than early developed games. This game challenges the ability of users to take immediate and quick actions as well as decisions.

1.3 Aims & Objectives

<u>**Aims:**</u>

The major target of this project is to create a game giving off thrilling atmosphere. After several discussion and with our skills, we wanted to our players to demonstrate their ability to survive in a hostile environment in the game. Moreover they will have to show their skill to efficiently use their limited resources in various situations. Yet, the game should entertaining and challenging enough to give the player an interesting experience. In Addition, the game will be completely free, hence solving the problem to pay to have a good quality game.

<u>**Objectives:**</u>

- The game should be entertaining and fun to play.
- The game should give a thrilling and survival experience to the player.
- The game should show the amount of resources such as med kits, bullets to allow the player to help the player to use them efficiently.
- The game should show a health bar to indicate the player when to use med kits after receiving damage.
- The player needs to survive all the waves of the game to win.

1.4 Scope

The scope of the game is as follows:

- NO ESCAPE is an adventure, action and survival based game.
- It includes only one level, which very intense and difficult.
- It is a 16+ game, anyone below 16 years old cannot play.
- The game is a single-player with a third-person perspective.
- The game is written in C++ and with some use of blueprints.

1.5 Distribution of tasks table

	Aaishane	Deesha	Tavish	Ismaail
Acknowledgment		100%		
Abstract		100%		
Table of content		100%		
List of tables		100%		
List of figures		100%		
1.1		100%		
1.2		100%		
1.3	100%			
1.4	50%			50%
1.5	75%	25%	100%	
2.1			50%	50%
2.2				
2.3		100%		
2.4				100%
3.1	100%			
3.2	50%	50%		
3.3	50%			50%
3.4		100%		
3.5		100%		
4.1	100%			
4.2			100%	
4.3		100%		
5.1	100%			
5.2	100%			
5.3		100%		
5.4	100%			
5.5		100%		
5.6	100%			
5.7	100%			
5.8			100%	
5.9			100%	

5.10				100%
5.11				100%
5.12				100%
5.13				100%
5.14				100%
6.1	100%			
6.2		100%		
6.3			100%	
6.4				100%
6.5	100%			
6.5	100%			
6.6	100%			
7.1		100%		
7.2		100%		
7.3		100%		
8.1				100%

Table 1: Distribution of table

2. Background study

2.1 Basic concept

NO ESCAPE is a 3D and TPP (third person perspective) game consisting of a single map situated in a village. The player will have to fight several waves of zombies alone in order to survive. As the game go on, the waves will become more difficult till the last wave. There will be various types of zombies to kill and weapons scattered across the map which the player may choose to buy. To beat the game, the player will have to kill the boss of the last wave and exterminate every last zombies.

2.2 Existing applications

Resident Evil 4

Figure 1: Resident Evil – Image 1

Figure 2: Resident Evil – Image 2

Resident Evil 4 is a 2004 survival horror third-person shooter game developed by Capcom Production Studio 4 and published by Capcom. The game take place in a rural part of Europe. The player control U.S. government special agent Leon S. Kennedy, who is sent on a mission to rescue the U.S. president's daughter Ashley Graham, who has been kidnapped by a cult. The environment of this game is similar to our game's environment as it occurs in a village.

Days Gone

Figure 3: Days Gone – Image 1

Figure 4: Days Gone – Image 2

Days Gone is an open-world, action-adventure game set two years after a devastating global pandemic. The player controls a character who is a bounty hunter and struggle to survive in a hostile environment by fighting zombies and people. The player also has a variety of weapons and traps to use to fights the hordes of zombies.

Our game also contains the experience where the player is attacked by a large number of zombies.

Call of Duty: World at War – Zombies

Figure 5: Call of Duty: World at War – Zombies– Image 1

Figure 6: Call of Duty: World at War – Zombies– Image 2

Call of Duty: World at War – Zombies is a first-person shooter and horror video game. It is single player as well multi player (maximum 4 persons) game. It is set to take veterans and newcomers alike on a bold and terrifying journey

2.3 Potential tools

1. Unreal Engine

Unreal Engine is a set of development tools for creating 2D and 3D games. C++ is the primary programming language that is utilised. Unreal is ideally suited for creating games with a high level of photo-realism. It allows creators in a variety of industries the freedom and control they need to create cutting-edge entertainment, captivating visualizations, and immersive virtual environments.

2. Mixamo

Mixamo is an online tool for animating 3D characters. Mixamo's technologies use machine learning methods to automate the steps of the character animation process, including 3D modelling to rigging and 3D animation. . Mixamo also provides an online, automatic model rigging service known as the Auto-Rigger, which was the industry's first online rigging service. The AutoRigger applies machine learning to understand where the limbs of a 3D model are and to insert a "skeleton", or rig, into the 3D model as well as calculating the skinning weights. (Wikipedia, November 2021.)

3. DesignDoll

Design Doll is a software program that can freely manipulate human body models in 3D space. You can create postures and compositions that artists demand, with easy, intuitive operations simultaneously. (Terawell)

4. Adobe Photoshop

Adobe Photoshop is the most widely used photo editing and alteration software. It can be used for everything from full-featured photo editing to making detailed digital paintings and sketches that look like they were done by hand

5. **Blender**

Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. Advanced users employ Blender’s API for Python scripting to customize the application and write specialized tools; often these are included in Blender’s future releases. Blender is well suited to individuals and small studios who benefit from its unified pipeline and responsive development process. (blender.org)

6. **MonoDevelop**

MonoDevelop enables developers to quickly write desktop and web applications on Linux, Windows and macOS. It also makes it easy for developers to port. NET applications created with Visual Studio to Linux and macOS maintaining a single code base for all platforms. (monodevelop.com)

7. **Urho3D**

Urho3D is a cross platform 2D and 3D game engine implemented in C++ and released under MIT license that has a variety of features to help you build your game. Urho3D is a cross platform 2D and 3D game engine implemented in C++ and released under MIT license that has a variety of features to help you build your game. (Google)

8. **Unity**

User bring their imagination, skills and determination to myriad real-world applications for the global audience. Equipped with Unity solutions, creators bring their visions and stories to life – to entertain, inform, and shape the world. (brand.unity.com)

9. **CryEngine**

CryEngine (stylized as CRYENGINE) is a game engine designed by the German game developer Crytek. It has been used in all of their titles with the initial version being used in Far Cry, and continues to be updated to support new consoles and hardware for their games. It has also been used for many third-party games under Crytek's licensing scheme, including Sniper: Ghost Warrior 2 and SNOW. (Wikipedia, December 2021.)

10. **Godot**

Godot aims to offer a fully integrated game development environment. It allows developers to create a game, needing no other tools beyond those used for content creation (visual assets, music, etc.). The engine's architecture is built around the concept of a tree of "nodes". (Wikipedia)

11. **Microsoft Visual Studio**

The Visual Studio IDE (integrated development environment) is a software program for developers to write and edit their code. Its user interface is used for software development to edit, debug and build code. Visual Studio includes a code editor supporting IntelliSense (the code completion component) as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a code profiler, designer for building GUI applications, web designer, class designer, and database schema designer. (incredibuilt.com)

12.GameMaker Studio

GameMaker Studio is also a game engine that offers up a simple drag-and-drop interface style and workflow options for multiple platforms that allow you to export the game to desktop, web, UWP, mobile, and console platforms, GameMaker Studio 2 is an excellent option in terms of game making software for beginners, and features a list of tools that includes skinning, docking, and object editor, and a script editor. (neit.edu)

13. RPG Maker

PG Maker is, as the name suggests, a role-playing-game maker that doesn’t really require programming skills to create games from scratch. It can be used for game creation across platforms such as Windows, Mac, IOS and Android – all without coding experience.

2.4 Summary of Findings

After a lot of researches and online videos about game-making, our team has decided to create an offline Third-Person Shooting game known as “NO ESCAPE” Where the player has to kill hordes of zombies and survive.

As beginners, we decided to use Unreal Engine, version 4.27.2 to build the game as it provides many facilities such as navigations, tutorials, blueprints, modules references and presets. Also, it is most suitable to use unreal engine to make 3D games. These features provides a friendly working environment for beginners. Moreover, there are many tutorial videos which teach us to use the engine and make a game easily.

The engine also has an online marketplace in its website where we can download animations and 3d models for free and the engine also allows us to import external assets and include it in the game.

The language to use is C++ as it is the first programming language taught at the university in Software Engineering course. Additionally, it is one of the most common language that is used to create video games. Hence, it will be easier for us to program the game.

Adobe Photoshop and Blender will be used for editing and modifying parts of the game.

3. Analysis

3.1 Proposed System– Brief Description of NO ESCAPE

The video game is a 3D, single-player, strategical and horror game where a player controls a character which can move freely across a large map, locate and kill zombies on in a village.

The game consists of only 1 massive level where the player firstly spawns in the forest where he has to walk to the village and start killing the zombies. Along the way, the players firstly collect a weapon along with some ammo.

When player will first enter the village, he will encounter some zombies guarding the entrance. He will have to kill the zombies before going deeper in the village. Along his path he will encounter some special zombies which will deal higher damage, are faster, and has more health. The character also has to exterminate every zombies of the village in order to win.

There player will be able to search different weapon with different rarity. Each weapon rarity is assigned with a different damage. The greater the damage if the weapon's rarity is better. The player can collect some ammo in the village while fighting the zombies

To make the game more difficult the player will have to survive without any health he has to efficiently use his limited ammo capacity. If the player dies he has to restart the level again.

The game conclude when the player kills all the zombies in the village.

The game is named "NO ESCAPE"

3.2 Functional requirements

FR01: The system shall display “NO ESCAPE” when starting the game.

FR02: After displaying the introductory video, a ‘Main Menu’ shall be appeared.

FR03: The system shall display ‘Play New Game’ option on the ‘Main Menu’.

FR04: The system shall display ‘Credits’ option on the ‘Main Menu’.

FR05: The system shall display ‘Quit’ option on the ‘Main Menu’.

FR06: The system shall display Sound and Music icons on the ‘Settings’ option.

FR07: The user shall be able to view their ‘Credits’ in the ‘Credits’ option.

FR08: The system shall display a ‘Back’ option in the ‘Credit’ option.

FR09: The system shall display ‘New game’ option if ever a user wants to play a new game.

FR10: When continuing on ‘Play game’ option, the system shall display a pause menu whereby the user will be allowed to choose from ‘Resume’, ‘Main menu’ and ‘Quit’.

FR11: The system shall allow the user to resume the game.

FR12: The system shall allow the user to quit the game.

FR13: The game shall allow the character to run, walk and jump, turn left and right, crouch and stay idle.

FR14: The game shall allow the character to attack the zombies in multiple ways

FR15: The system shall only one wave.

FR16: The user only wins if he/she kills all the zombies in the wave.

FR17: The system shall perform a death animation on the screen if the user gets killed by one or many zombies.

FR18: After losing, the user shall be allowed to choose from wither ‘Main menu’ or ‘Quit’ options.

FR19: The system shall display the amount of inventory that the character has.

FR20: The system shall display a small map and the coordinates of the character on the screen.

FR21: The system shall display a bigger version of the map, when the user press 'M'.

FR22: The system shall display the health of the character.

FR23: The system shall display the number of ammo in the inventory.

FR24: The system shall display the type of weapons in the inventory.

FR25: The system shall display a widget when being near an item or a weapon.

FR26: The system shall display the health of the enemies above their head.

3.3 Non-Functional requirements

NF01: The system shall be written using C++ programming language.

NF02: The system shall be implemented using Unreal Engine.

NF03: The game shall be in third person perspective.

NF04: The game shall run on Windows.

NF05: The system shall have a minimum of two clicks for any function button in the game.

NF06: The system shall respond within at least 3 seconds the user clicks a button.

NF07: The system shall be compiled using Visual Studio.

NF08: The system shall require 30 GB of available hard drive space.

NF09: The game shall not use more than 8 GB of RAM to operate.

NF10: The game shall not contain any 18+ content.

3.4 Use Case diagram

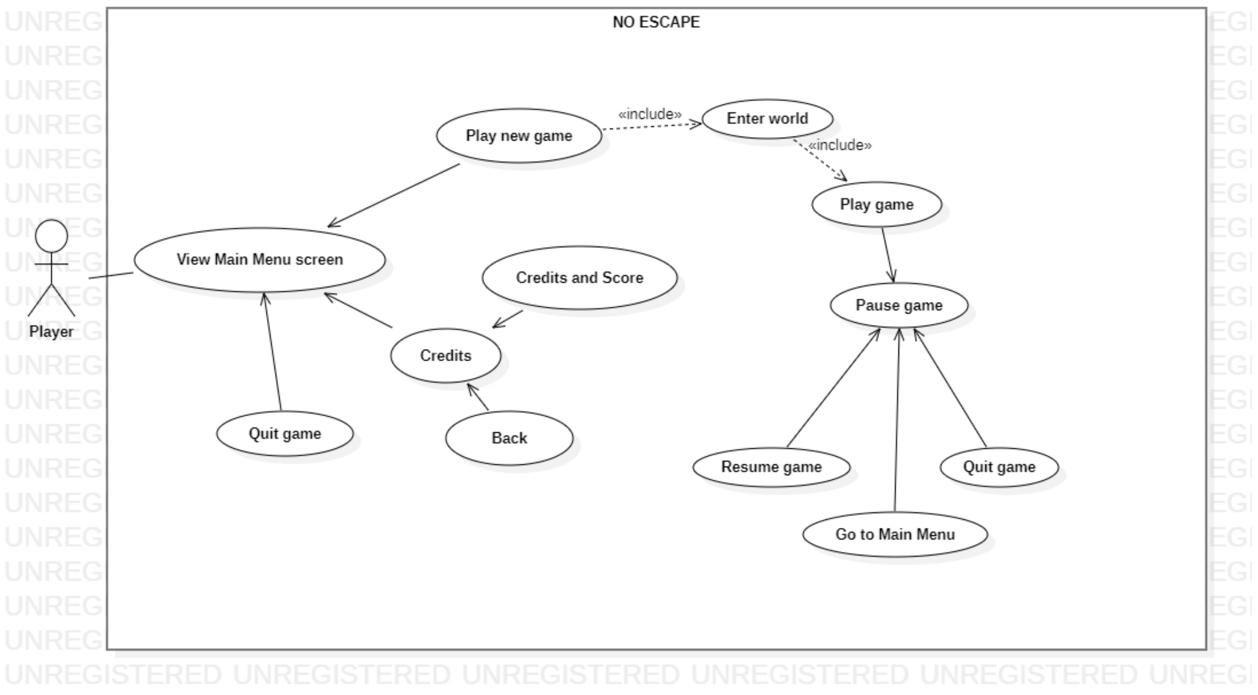


Figure 7: Use case diagram for game

3.5 Tools chosen

Game engine

The game engine chosen for our game is Unreal Engine (version 4).

It is a free software and allows you to create 2D and 3D games.

Documentation, tutorials, support resources and free content such as templates, samples are available for use.

It also has numerous assets that are available in its marketplace online.

C++ is the programming language. We choose this as we have been learning C++ in semester 1 & 2 and this has helped us to improve our coding skills.

Code editor

A code editor is essential as this is what allows you to create codes to implement the game. We used Visual Studio Community 2019 with C++ as the programming language.

Graphics editor

We have used Adobe Photoshop for editing and designing the health and progress bar, the map and so on.

4. Design

4.1 Architectural design

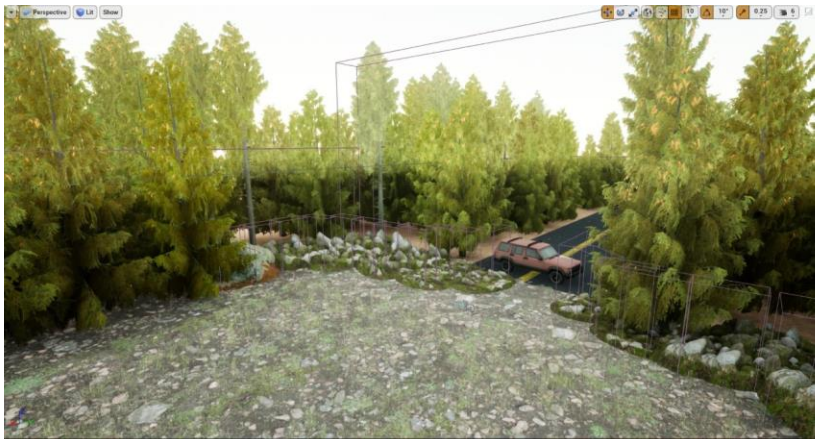


Figure 8: Forest Image 1

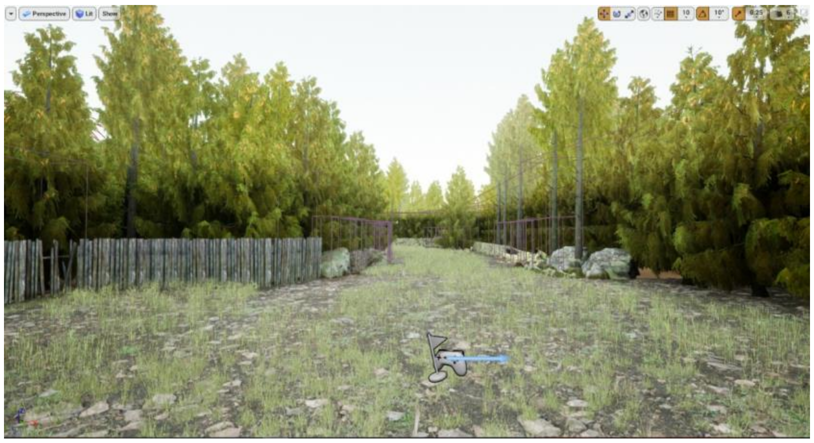


Figure 8: Forest Image 2

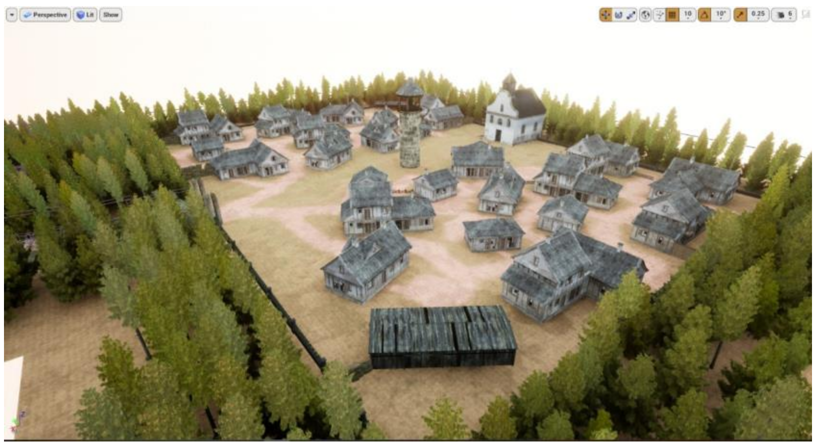


Figure 9: Village Image 1

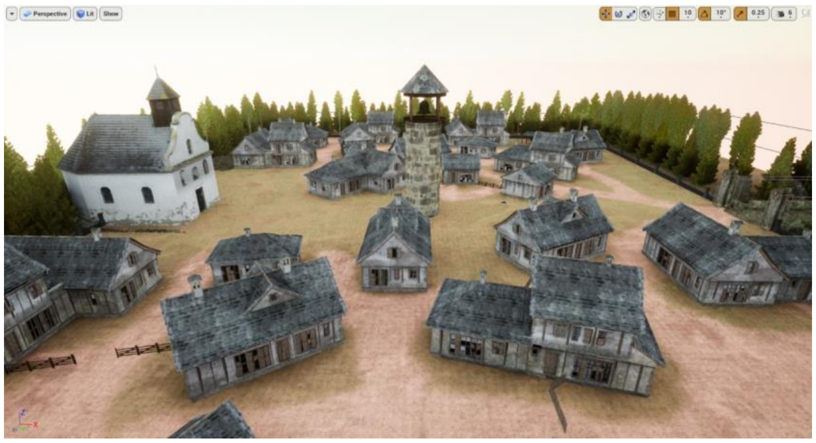


Figure 10: Village Image 2

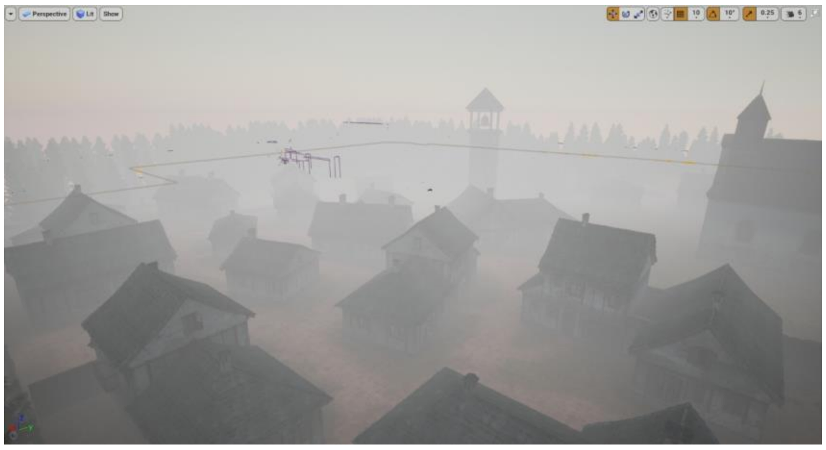


Figure 11:Village Image 1 with fog

4.2 UI Design

Main menu screen

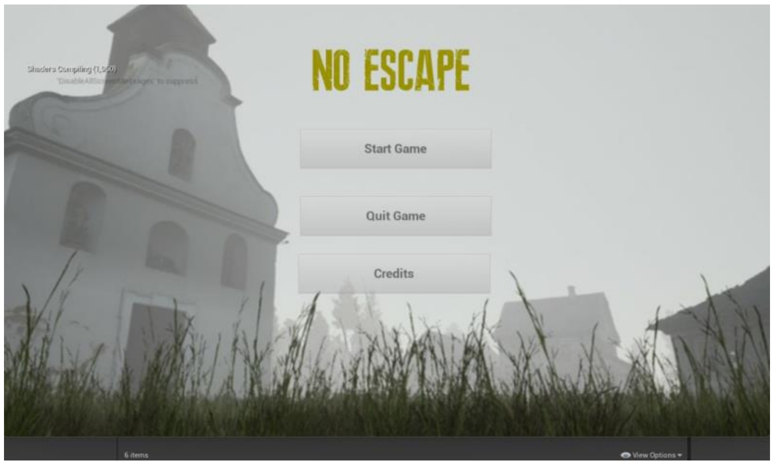


Figure 12:Main menu

Credits screen

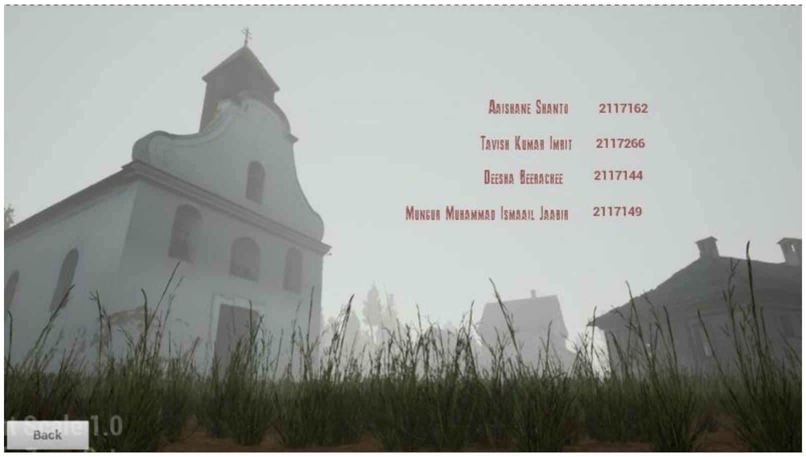

Aaishane Shanto 2117162
Tavish Kumar Imrit 2117266
Deesha Beerachee 2117144
Mungur Muhammad Ismaail Jaabir 2117149
Back

Figure 13: Credits screen

4.3 Program Designs

Main menu screen

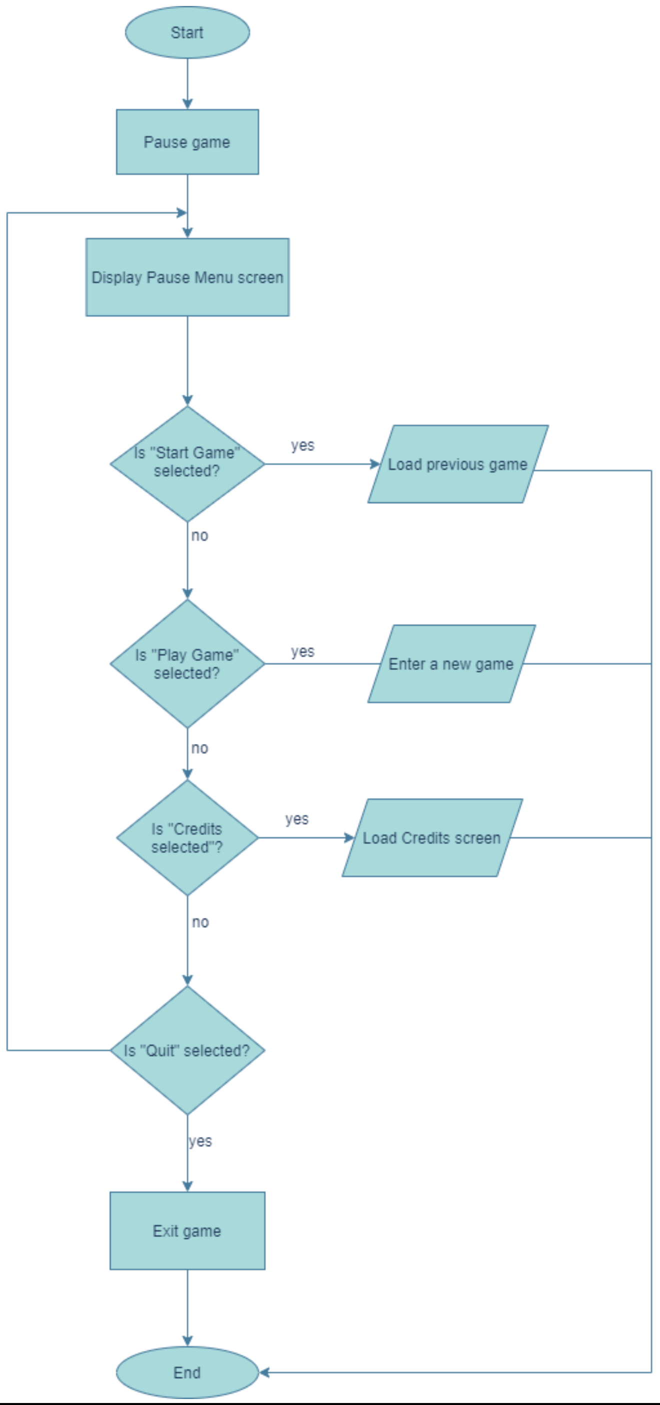


Figure 14: Flowchart for main menu screen

Pause Menu screen

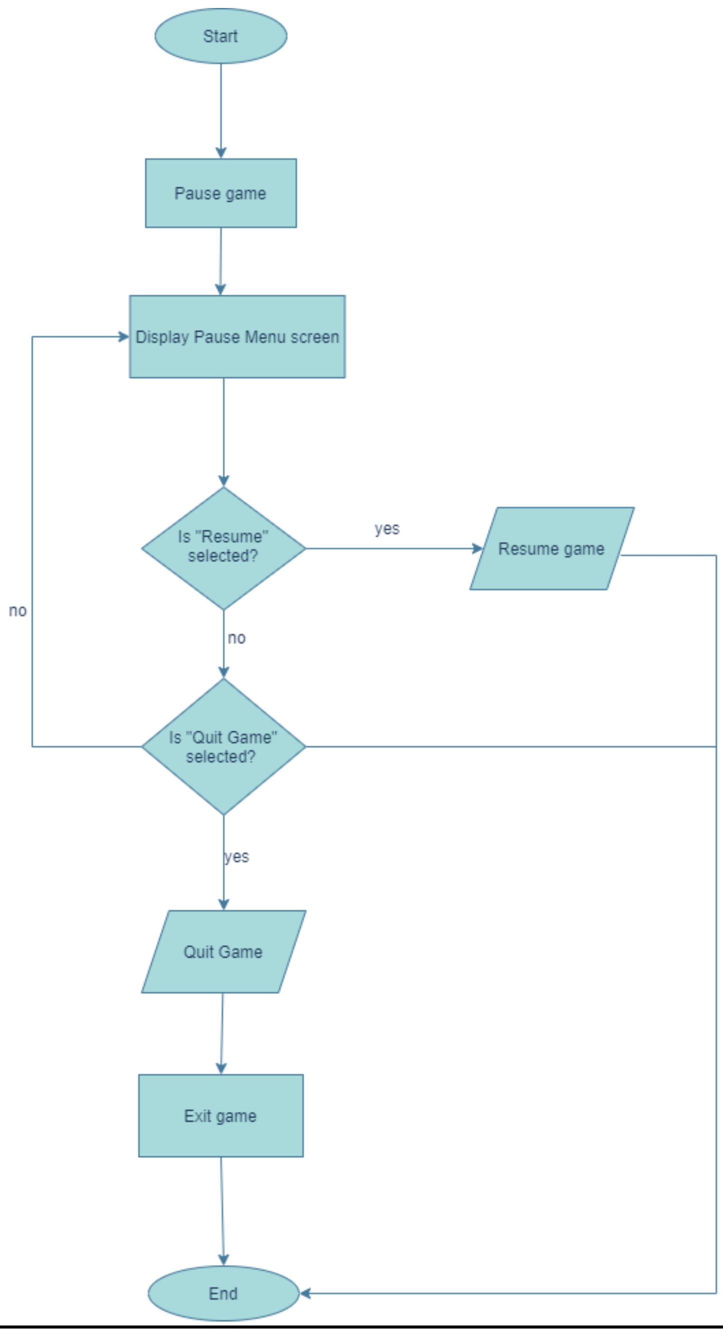


Figure 15: Flowchart for pause menu screen

5. Implementation

5.1 System requirements

- The system should have a windows 8 OS or a later version to be able to run this game
- The system should have a processor AMD Ryzen 5 5600H series with 3.30GHz or a similar processor power
- The system requires at least 8 GB RAM, GeForce GTX 1650 GPU
- The system requires 30 GB memory storage

5.2 Component: Dependencies for every .h and .cpp files

This component shows all the dependencies among the .cpp and .h files because some variables from a file are being used in another file.

ShooterCharacter.h

```
#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "AmmoType.h"
#include "ShooterCharacter.generated.h"
```

ShooterCharacter.cpp

```
#include "ShooterCharacter.h"
#include "GameFramework/SpringArmComponent.h"
#include "Camera/CameraComponent.h"
#include "GameFramework/CharacterMovementComponent.h"
#include "Kismet/GameplayStatics.h"
#include "Sound/SoundCue.h"
#include "Engine/SkeletalMeshSocket.h"
#include "DrawDebugHelpers.h"
#include "particles/ParticleSystemComponent.h"
#include "Item.h"
#include "Components/WidgetComponent.h"
#include "Weapon.h"
#include "Components/SphereComponent.h"
#include "Components/BoxComponent.h"
#include "Components/CapsuleComponent.h"
#include "Engine/DataTable.h"
#include "Ammo.h"
#include "BulletHitInterface.h"
#include "Enemy.h"
```

```
#include "EnemyController.h"
#include "BehaviorTree/BlackboardComponent.h"
```

ShooterPlayerController.h

```
#include "CoreMinimal.h"
#include "GameFramework/PlayerController.h"
#include "ShooterPlayerController.generated.h"
```

ShooterPlayerController.cpp

```
#include "ShooterPlayerController.h"
#include "Blueprint/UserWidget.h"
```

Weapon.h

```
#include "CoreMinimal.h"
#include "Item.h"
#include "AmmoType.h"
#include "Engine/DataTable.h"
#include "Weapon.generated.h"
```

Weapon.cpp

```
#include "Weapon.h"
```

ShooterAnimInstance.h

```
#include "CoreMinimal.h"
#include "Animation/AnimInstance.h"
#include "ShooterAnimInstance.generated.h"
```

ShooterAnimInstance.cpp

```
#include "ShooterAnimInstance.h"
#include "ShooterCharacter.h"
#include "GameFramework/CharacterMovementComponent.h"
#include "Kismet/KismetMathLibrary.h"
```

Item.h

```
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Engine/DataTable.h"
#include "Item.generated.h"
```

Item.cpp

```
#include "Item.h"
#include "Components/BoxComponent.h"
```

```
#include "Components/WidgetComponent.h"
#include "Components/SphereComponent.h"
#include "ShooterCharacter.h"
#include "Camera/CameraComponent.h"
#include "Kismet/GameplayStatics.h"
#include "Sound/SoundCue.h"
```

Enemy.h

```
#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "BulletHitInterface.h"
#include "Enemy.generated.h"
```

Enemy.cpp

```
#include "Enemy.h"
#include "Kismet/GameplayStatics.h"
#include "Sound/SoundCue.h"
#include "Particles/ParticleSystemComponent.h"
#include "Blueprint/UserWidget.h"
#include "Kismet/KismetMathLibrary.h"
#include "EnemyController.h"
#include "BehaviorTree/BlackboardComponent.h"
#include "Components/SphereComponent.h"
#include "ShooterCharacter.h"
#include "Components/CapsuleComponent.h"
#include "Components/BoxComponent.h"
#include "Engine/SkeletalMeshSocket.h"
```

EnemyAnimInstance.h

```
#include "CoreMinimal.h"
#include "Animation/AnimInstance.h"
#include "EnemyAnimInstance.generated.h"
```

EnemyAnimInstance.cpp

```
#include "EnemyAnimInstance.h"
#include "Enemy.h"
```

EnemyController.h

```
#include "CoreMinimal.h"
#include "AIController.h"
#include "EnemyController.generated.h"
```

EnemyController.cpp

```
#include "EnemyController.h"
#include "BehaviorTree/BlackboardComponent.h"
#include "BehaviorTree/BehaviorTreeComponent.h"
#include "BehaviorTree/BehaviorTree.h"
#include "Enemy.h"
```

BulletInterface.h

```
#include "CoreMinimal.h"
#include "UObject/Interface.h"
#include "BulletHitInterface.generated.h"
```

BulletInterface.cpp

```
#include "BulletHitInterface.h"
```

Ammo.h

```
#include "CoreMinimal.h"
#include "Item.h"
#include "AmmoType.h"
#include "Ammo.generated.h"
```

Ammo.cpp

```
#include "Ammo.h"
#include "Components/BoxComponent.h"
#include "Components/WidgetComponent.h"
#include "Components/SphereComponent.h"
#include "ShooterCharacter.h"
```

5.3 Component: Character Movement

This component is used to show the movements -crouching, jumping abilities and turning effect and angle of the Character.

ShooterCharacter.h

```
protected:

/*called for forward/bavkwards input*/
	void MoveForward(float Value);
	/*called for side to side inputs*/
	void MoveRight(float Value);

	void TurnAtRate(float Rate);

	void LookUpAtRate(float Rate);


	/*rotate controller based on mouse x movement
	the param value is the input value from mouse movement*/
	void Turn(float Value);

	void LookUp(float Value);


	float BaseLookUpRate;


	 void CrouchButtonPressed();


	virtual void Jump()override;


private:

/* Base turn rate, in deg/sec */
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta =
(AllowPrivateAccess = "true"))
	float BaseTurnRate;

	/*base look up*/
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta =
(AllowPrivateAccess = "true"))


//true when crouching
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Movement, meta =
(AllowPrivateAccess = "true"))
	bool bCrouching;


	//regular movement speed
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Movement, meta =
(AllowPrivateAccess = "true"))
	float BaseMovementSpeed;

	//crouch movement speed
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Movement, meta =
(AllowPrivateAccess = "true"))
	float CrouchMovementSpeed;
```

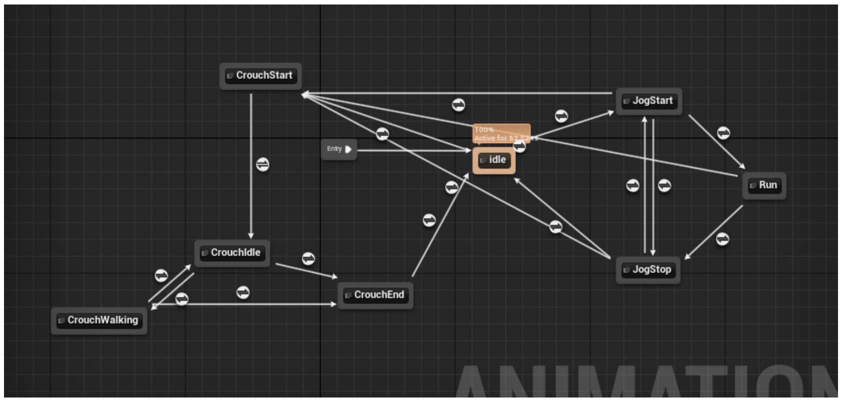
ShooterCharacter.cpp

```
AShooterCharacter::AShooterCharacter() :

//base rates for turning/looking up
	BaseTurnRate(45.f),
	BaseLookUpRate(45.f),

	bCrouching(false),
	BaseMovementSpeed(650.f),
	CrouchMovementSpeed(300.f),

    GetCharacterMovement()->bOrientRotationToMovement = false;//character moves in
direction of input
	GetCharacterMovement()->RotationRate = FRotator(0.f, 540.f, 0.f);
	GetCharacterMovement()->JumpZVelocity = 600.f;
	GetCharacterMovement()->AirControl = 0.2f;



void AShooterCharacter::BeginPlay()

{

Super::BeginPlay();


GetCharacterMovement()->MaxWalkSpeed = BaseMovementSpeed;

}



void AShooterCharacter::MoveForward(float Value)
{
	if ((Controller != nullptr) && (Value != 0.0f))
	{
		//find out which way is forward
		const FRotator Rotation{ Controller->GetControlRotation() };
		const FRotator YawRotation{ 0, Rotation.Yaw, 0 };

		const FVector Direction{ FRotationMatrix{YawRotation}.GetUnitAxis(EAxis::X)
};
		AddMovementInput(Direction, Value);
    }
}
```

```
void AShooterCharacter::MoveRight(float Value)
{
	if ((Controller != nullptr) && (Value != 0.0f))
	{
		//find out which way is right
		const FRotator Rotation{ Controller->GetControlRotation() };
		const FRotator YawRotation{ 0, Rotation.Yaw, 0 };

		const FVector Direction{ FRotationMatrix{YawRotation}.GetUnitAxis(EAxis::Y)
};
		AddMovementInput(Direction, Value);
	}

}


void AShooterCharacter::TurnAtRate(float Rate)
{
	AddControllerYawInput(Rate * BaseTurnRate * GetWorld()->GetDeltaSeconds());

}

void AShooterCharacter::LookUpAtRate(float Rate)
{
	AddControllerPitchInput(Rate * BaseLookUpRate * GetWorld()->GetDeltaSeconds());
}

void AShooterCharacter::Jump()
{
	if (bCrouching)
	{
		bCrouching = false;
		GetCharacterMovement()->MaxWalkSpeed = BaseMovementSpeed;
	}
	else
	{
		ACharacter::Jump();
	}

}

void AShooterCharacter::CrouchButtonPressed()
{
	if (!GetCharacterMovement()->IsFalling())
	{
		bCrouching = !bCrouching;
	}
	if (bCrouching)
	{
		GetCharacterMovement()->MaxWalkSpeed = CrouchMovementSpeed;
		GetCharacterMovement()->GroundFriction = CrouchingGroundFriction;
	}
	else
	{
```

```
                GetCharacterMovement()->MaxWalkSpeed =BaseMovementSpeed;
                GetCharacterMovement()->GroundFriction = BaseGroundFriction;
        }
}
```

Blueprint for character movement



5.4 Component: Character Aiming and firing Movement

This component shows the aiming and firing movement of the character. It also shows some advance movement where the player can crouch, aim and firing at the same time, aim and firing while jumping and the change in the movement speed while aiming

ShooterCharacter.h

```
protected:

void FireWeapon();//when fire button is pressed

void AimingButtonPressed();
void AimingButtonReleased();

//set BaseTurnRate and BaseLookUpRate based on aiming
void SetLookRates();

void FireButtonPressed();
void FireButtonReleased();

void StartFireTimer();

UFUNCTION()
void AutoFireReset();

void Aim();
void StopAiming();


private:

//turn rate when aiming
UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = Camera, meta =
(AllowPrivateAccess = "true"))
       float AimingTurnRate;
//look up rate when aiming
UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = Camera, meta =
(AllowPrivateAccess = "true"))
       float AimingLookUpRate;

//scale factor for mouse look sensitivity.turn rate when aiming
UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = Camera, meta =
(AllowPrivateAccess = "true")
, meta = (ClampMin = "0.0", ClampMax = "1.0", UIMin = "0.0", UIMax = "1.0"))
float MouseAimingTurnRate;

       //scale factor for mouse look sensitivity.look up rate when aiming
UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = Camera, meta =
(AllowPrivateAccess = "true")
, meta = (ClampMin = "0.0", ClampMax = "1.0", UIMin = "0.0", UIMax = "1.0"))
float MouseAimingLookUpRate;
```

```
/*true when aiming*/
UPROPERTY(VisibleAnywhere,BlueprintReadOnly,Category=Combat,meta=
(AllowPrivateAccess="true"))
bool bAiming;

//knowing when the aiming button is pressed
	bool bAimingButtonPressed;


public:

FORCEINLINE bool GetAiming() const { return bAiming; }
```

ShooterCharacter.cpp

```
AShooterCharacter::AShooterCharacter() :

//true when aiming the weapon
bAiming(false),

//Turn Rates for aiming

AimingTurnRate(20.f),
AimingLookUpRate(20.f),

//mouse look sensitivities scale factors

MouseAimingTurnRate(0.5f),
MouseAimingLookUpRate(0.5f),

//bullet fire timer variables
ShootTimeDuration(0.05f),
//automatic fire variables
AutomaticFireRate(0.1f),
bFiringBullet(false),
bShouldFire(true),
bFireButtonPressed(false),


void AShooterCharacter::Turn(float Value)
{
	float TurnScaleFactor{};
	if (bAiming)
	{
		TurnScaleFactor = MouseAimingTurnRate;
	}
	else
	{
		TurnScaleFactor = MouseHipTurnRate;
	}
	AddControllerYawInput(Value * TurnScaleFactor);
}
```

```cpp
void AShooterCharacter::LookUp(float Value)
{
	float LookUpScaleFactor{};
	if (bAiming)
	{
		LookUpScaleFactor = MouseAimingLookUpRate;
	}
	else
	{
		LookUpScaleFactor = MouseHipLookUpRate;
	}
	AddControllerPitchInput(Value * LookUpScaleFactor);

}


void AShooterCharacter::AimingButtonPressed()
{
	bAimingButtonPressed = true;
	if (CombatState != ECombatState::ECS_Reloading)
	{
		Aim();
	}
}


void AShooterCharacter::AimingButtonReleased()
{
	bAimingButtonPressed = false;
	StopAiming();
}


void AShooterCharacter::Aim()
{
	bAiming = true;
	GetCharacterMovement()->MaxWalkSpeed = CrouchMovementSpeed;
}


void AShooterCharacter::StopAiming()
{
	bAiming = false;
	if (!bCrouching)
	{
		GetCharacterMovement()->MaxWalkSpeed = BaseMovementSpeed;
	}
}
```

5.5 Component: Camera

This component shows the angle and positioning of the camera while the player is moving, crouching or aiming

ShooterCharacter.h

```
protected:

void CameraInterpZoom(float DeltaTime);


private:

	/*Camera boom positioning the camera behind the character*/
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera,meta =
(AllowPrivateAccess= "true"))
	class USpringArmComponent* CameraBoom;

	/*camera that follows the character*/
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta =
(AllowPrivateAccess = "true"))
	class UCameraComponent* FollowCamera;



/*default camera field of view value*/
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess = "true"))
	float CameraDefaultFOV;

	/*field of view value when zoomed in*/
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess = "true"))
	float CameraZoomedFOV;

	/*current field of view this frame*/
	float CameraCurrentFOV;



//distance outward from the camera for the interp destination
	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Items, meta =
(AllowPrivateAccess = "true"))
	float CameraInterpDistance;

	//distance upward from the camera for the interp destinaton
	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Items, meta =
(AllowPrivateAccess = "true"))
	float CameraInterpElevation;

public:

/*returns camreraBoom subobject*/
	FORCEINLINE USpringArmComponent* GetCameraBoom() const { return CameraBoom; }
	/*returns followcamera subobject*/
	FORCEINLINE UCameraComponent* GetFollowCamera() const { return FollowCamera; }
```

ShooterCharacter.cpp

```
CameraInterpDistance(250.f),
CameraInterpElevation(65.f),

//camera fields of view values
	CameraDefaultFOV(0.f),//set in BeginPlay
	CameraZoomedFOV(35.f),
	CameraCurrentFOV(0.f),
	ZoomInterpSpeed(20.f),



/*create a camera boom(pulls in toward character if there is a collision)*/
	CameraBoom = CreateDefaultSubobject<USpringArmComponent>(TEXT("CameraBoom"));
	CameraBoom->SetupAttachment(RootComponent);
	CameraBoom->TargetArmLength = 180.0f; //the camera follows at this disrance behind
the character
	CameraBoom->bUsePawnControlRotation = true; // rotate the Arm based on the
controller
	CameraBoom->SocketOffset = FVector(0.f, 50.f, 70.f);//adjust camera distance

	/*create a follow camera*/
	FollowCamera = CreateDefaultSubobject<UCameraComponent>(TEXT("FollowCamera"));
	FollowCamera->SetupAttachment(CameraBoom,
USpringArmComponent::SocketName);//attach the camera to the end of the boom
	FollowCamera->bUsePawnControlRotation = false;//camera does not rotate relative to
the arm


void AShooterCharacter::BeginPlay()

{



if (FollowCamera)
	{
		CameraDefaultFOV = GetFollowCamera()->FieldOfView;
		CameraCurrentFOV = CameraDefaultFOV;
	}

}



void AShooterCharacter::CameraInterpZoom(float DeltaTime)
{
	/*set current camera field of view*/
	if (bAiming)
	{
		//interpolate to zoomed FOV
		CameraCurrentFOV = FMath::FInterpTo(CameraCurrentFOV,
			CameraZoomedFOV,
			DeltaTime,
			ZoomInterpSpeed);
			GetFollowCamera()->SetFieldOfView(CameraCurrentFOV);
		}
		else
		{
			//interpolate to default FOV
			CameraCurrentFOV = FMath::FInterpTo(CameraCurrentFOV,
				CameraDefaultFOV,
				DeltaTime,
				ZoomInterpSpeed);
		}
		GetFollowCamera()->SetFieldOfView(CameraCurrentFOV);
}



void AShooterCharacter::SetLookRates()
{
		if (bAiming)
		{
			BaseTurnRate = AimingTurnRate;
			BaseLookUpRate = AimingLookUpRate;
		}
		else
		{

			BaseTurnRate = HipTurnRate;
			BaseLookUpRate = HipLookUpRate;
		}

}


void AShooterCharacter::Tick(float DeltaTime)

{

//handle interpolation for zoom when aiming
		CameraInterpZoom(DeltaTime);

}
```

5.6 Component: Crosshair

This component shows how the crosshair is shown on the HUD and how it spreads when the character moves, crouches or aim.

ShooterCharacter.h

```
protected:

void CalculateCrosshairSpread(float DeltaTime);

void StartCrosshairBulletFire();

UFUNCTION()
void FinishCrosshairBulletFire();

public:

/*determines the spread of the crosshairs*/
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Crosshairs, meta =
(AllowPrivateAccess = "true"))
float CrosshairSpreadMultiplier;
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Crosshairs, meta =
(AllowPrivateAccess = "true"))
float CrosshairVelocityFactor;//velocity component for crosshair spread
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Crosshairs, meta =
(AllowPrivateAccess = "true"))
float CrosshairInAirFactor;//in air component for crosshairs spread
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Crosshairs, meta =
(AllowPrivateAccess = "true"))
float CrosshairAimFactor;//aim component for crosshair spread
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Crosshairs, meta =
(AllowPrivateAccess = "true"))
float CrosshairShootingFactor;//shooting component for crosshairs spread


public:

UFUNCTION(BlueprintCallable)
float GetCrosshairSpreadMultiplier() const;
```

ShooterCharacter.cpp

```
AShooterCharacter::AShooterCharacter() :

//crosshair spread factors
	CrosshairSpreadMultiplier(0.f),
	CrosshairVelocityFactor(0.f),
	CrosshairInAirFactor(0.f),
	CrosshairAimFactor(0.f),
	CrosshairShootingFactor(0.f),
void AShooterCharacter::CalculateCrosshairSpread(float DeltaTime)
{
	FVector2D WalkSpeedRange{ 0.f,600.f };
	FVector2D VelocityMultiplierRange{0.f, 1.f};
	FVector Velocity{ GetVelocity() };
	Velocity.Z = 0.f;

	CrosshairVelocityFactor = FMath::GetMappedRangeValueClamped(
		WalkSpeedRange,
		VelocityMultiplierRange,
		Velocity.Size());

	// Calculate Crosshair in Air Factor
	if (GetCharacterMovement()->IsFalling())//is in air?
	{
		//spread the crosshair slowly while in air
		CrosshairInAirFactor = FMath::FInterpTo(
			CrosshairInAirFactor, 2.25f, DeltaTime, 2.25f);
	}
	else//character is on the ground
	{
		//shrink the crosshair rapidly while on the ground
		CrosshairInAirFactor = FMath::FInterpTo(
			CrosshairInAirFactor, 0.f, DeltaTime, 30.f);
	}
	//calculate crosshair aim factor
	if (bAiming)//is character is aiming?
	{
		//shrink the crosshair a small amount very quickly
		CrosshairAimFactor = FMath::FInterpTo(
			CrosshairAimFactor, 0.6f,DeltaTime,30.f);
	}
	else//not aiming
	{
		//spread the crosshair back to normal very quickly
		CrosshairAimFactor = FMath::FInterpTo(
			CrosshairAimFactor, 0.f, DeltaTime, 30.f);
	}
	if (bFiringBullet)//true 0.05 second after firing
	{
		CrosshairShootingFactor = FMath::FInterpTo(
			CrosshairShootingFactor,
			0.3f, DeltaTime, 60.f);
	}
	else
	{
		CrosshairShootingFactor = FMath::FInterpTo(
			CrosshairShootingFactor,
			0.f,
			DeltaTime, 60.f);
	}

	CrosshairSpreadMultiplier = 0.5f +
		CrosshairVelocityFactor +
		CrosshairInAirFactor -
		CrosshairAimFactor+
            CrosshairShootingFactor;

}


bool AShooterCharacter::TraceUnderCrosshairs(FHitResult& OutHitResult, FVector&
OutHitLocation)
{
       //get viewport size
       FVector2D ViewportSize;
       if (GEngine && GEngine->GameViewport)
       {
              GEngine->GameViewport->GetViewportSize(ViewportSize);
       }

       //get screen space location of crosshairs
       FVector2D CrosshairLocation(ViewportSize.X / 2.f, ViewportSize.Y / 2.f);
       //CrosshairLocation.Y -= 50.f;
       FVector CrosshairWorldPosition;
       FVector CrosshairWorldDirection;

       //get world position and direction of crosshairs
       bool bScreenToWorld = UGameplayStatics::DeprojectScreenToWorld(
              UGameplayStatics::GetPlayerController(this, 0),
              CrosshairLocation,
              CrosshairWorldPosition,
              CrosshairWorldDirection);

       if (bScreenToWorld)
       {
              // trace from the crosshair world location outward
              const FVector Start{ CrosshairWorldPosition };
              const FVector End{ Start + CrosshairWorldDirection * 50'000.f };
              OutHitLocation = End;
              GetWorld()->LineTraceSingleByChannel(
                     OutHitResult,
                     Start,
                     End,
                     ECollisionChannel::ECC_Visibility);

              if (OutHitResult.bBlockingHit)
              {
                     OutHitLocation = OutHitResult.Location;
                     return true;
              }
       }



       return false;
}


void AShooterCharacter::Tick(float DeltaTime)

{
```

```
//calculate crosshair spread multiplier
	CalculateCrosshairSpread(DeltaTime);

}



float AShooterCharacter::GetCrosshairSpreadMultiplier() const
{
	return CrosshairSpreadMultiplier;
}
```

5.7 Component: Item and Weapon interpolation

This component shows the interpolation done when a weapon or item is grabbed from the ground or when the player swap a weapon.

ShooterCharacter.h

```
USTRUCT(BlueprintType)
struct FInterpLocation
{
	GENERATED_BODY()

		// Scene component to use for its location for interping
		UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
		USceneComponent* SceneComponent;

	// Number of items interping to/at this scene comp location
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
		int32 ItemCount;
};
DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FEquipItemDelegate, int32, CurrentSlotIndex, 
int32, NewSlotIndex);


protected:

void CameraInterpZoom(float DeltaTime);

//interps capsule half height when crouching/standing
	void InterpCapsuleHalfHeight(float DeltaTime);

void InitializeInterpLocation();








private:

/* interp speed for zooming when aiming*/
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta = 
(AllowPrivateAccess = "true"))
	float ZoomInterpSpeed;



UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))
		USceneComponent* WeaponInterpComp;


	UPROPERTY(EditDefaultsOnly,BlueprintReadOnly, meta = (AllowPrivateAccess= "true"))
	USceneComponent* InterpComp1;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = 
"true"))
		USceneComponent* InterpComp2;

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = 
"true"))
		USceneComponent* InterpComp3;

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = 
"true"))
		USceneComponent* InterpComp4;

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = 
"true"))
		USceneComponent* InterpComp5;

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = 
"true"))
		USceneComponent* InterpComp6;

	UPROPERTY(EditDefaultsOnly,BlueprintReadOnly, Meta = (AllowPrivateAccess = 
"true"))
	TArray<FInterpLocation> InterpLocations;


	//distance outward from the camera for the interp destination
	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Items, meta = 
(AllowPrivateAccess = "true"))
	float CameraInterpDistance;

	//distance upward from the camera for the interp destinaton
	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Items, meta = 
(AllowPrivateAccess = "true"))
	float CameraInterpElevation;



public:

FInterpLocation GetInterpLocation(int32 Index);

	//return index in interplocation array
	int32 GetInterpLocationIndex();

	void IncrementInterpLocItemCount(int32 Index, int32 Amount);
```

ShooterCharacter.cpp

```
ShooterCharacter::AShooterCharacter() :
```

```
//item interpolation variables
CameraInterpDistance(250.f),
CameraInterpElevation(65.f),


//interpolation componenents
	WeaponInterpComp = CreateDefaultSubobject<USceneComponent>(TEXT("Weapon
Interpolation Component"));
	WeaponInterpComp->SetupAttachment(GetFollowCamera());

	InterpComp1 = CreateDefaultSubobject<USceneComponent>(TEXT("Interpolation
Component 1"));
	InterpComp1->SetupAttachment(GetFollowCamera());

	InterpComp2 = CreateDefaultSubobject<USceneComponent>(TEXT("Interpolation
Component 2"));
	InterpComp2->SetupAttachment(GetFollowCamera());

	InterpComp3 = CreateDefaultSubobject<USceneComponent>(TEXT("Interpolation
Component 3"));
	InterpComp3->SetupAttachment(GetFollowCamera());

	InterpComp4 = CreateDefaultSubobject<USceneComponent>(TEXT("Interpolation
Component 4"));
	InterpComp4->SetupAttachment(GetFollowCamera());

	InterpComp5 = CreateDefaultSubobject<USceneComponent>(TEXT("Interpolation
Component 5"));
	InterpComp5->SetupAttachment(GetFollowCamera());

	InterpComp6 = CreateDefaultSubobject<USceneComponent>(TEXT("Interpolation
Component 6"));
	InterpComp6->SetupAttachment(GetFollowCamera());

void AShooterCharacter::BeginPlay()

{

//create FInterplocation struct for each interp location.Add to array
	InitializeInterpLocation();

}



void AShooterCharacter::InterpCapsuleHalfHeight(float DeltaTime)
{
	float TargetCapsuleHalfHeight{};
	if (bCrouching)
	{
		TargetCapsuleHalfHeight = CrouchingCapsuleHalfHeight;
	}
	else
	{
		TargetCapsuleHalfHeight = StandingCapsuleHalfHeight;
	}
	const float InterpHalfHeight{ FMath::FInterpTo(
		GetCapsuleComponent()->GetScaledCapsuleHalfHeight(),
		TargetCapsuleHalfHeight,DeltaTime,
		20.f) };

	// Negative value if crouching ,negative value if standing
	const float DeltaCapsuleHalfHeight{ InterpHalfHeight
		- GetCapsuleComponent()->GetScaledCapsuleHalfHeight() };

	FVector MeshOffset{ 0.f,0.f,-DeltaCapsuleHalfHeight };

	GetCapsuleComponent()->SetCapsuleHalfHeight(InterpHalfHeight);

	GetMesh()->AddLocalOffset(MeshOffset);
}


void AShooterCharacter::InitializeInterpLocation()
{
	FInterpLocation WeaponLocation{ WeaponInterpComp, 0 };
	InterpLocations.Add(WeaponLocation);

	FInterpLocation InterpLoc1{ InterpComp1, 0 };
	InterpLocations.Add(InterpLoc1);

	FInterpLocation InterpLoc2{ InterpComp2, 0 };
	InterpLocations.Add(InterpLoc2);

	FInterpLocation InterpLoc3{ InterpComp3, 0 };
	InterpLocations.Add(InterpLoc3);

	FInterpLocation InterpLoc4{ InterpComp4, 0 };
	InterpLocations.Add(InterpLoc4);

	FInterpLocation InterpLoc5{ InterpComp5, 0 };
	InterpLocations.Add(InterpLoc5);

	FInterpLocation InterpLoc6{ InterpComp6, 0 };
	InterpLocations.Add(InterpLoc6);
}




int32 AShooterCharacter::GetInterpLocationIndex()
{
	int32 LowestIndex = 1;
	int32 LowestCount = INT_MAX;
	for (int32 i = 1; i < InterpLocations.Num(); i++)
	{
		if (InterpLocations[i].ItemCount < LowestCount)
		{
			LowestIndex = i;
			LowestCount = InterpLocations[i].ItemCount;
		}
	}

	return LowestIndex;
```

```
}


void AShooterCharacter::Tick(float DeltaTime)

{

//Interpolate the capsule half height
	InterpCapsuleHalfHeight(DeltaTime);

}



FInterpLocation AShooterCharacter::GetInterpLocation(int32 Index)
{
	if (Index <= InterpLocations.Num())
	{
		return InterpLocations[Index];
	}
	return FInterpLocation();
}
```

Item.h

```
protected:

//called when ItemInterpTimer is finished
	void FinishInterping();

	//handles item interpolation when in the equipinterping state
	void ItemInterp(float DeltaTime);

	FVector GetInterpLocation();


private:

//the curve assets to use for the item's Z location when interping
	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Item Properties", meta = 
(AllowPrivateAccess = "true"))
	class UCurveFloat* ItemZCurve;

	//starting location when interping begins
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Item Properties", meta = 
(AllowPrivateAccess = "true"))
	FVector ItemInterpStartLocation;

	//target interp location in front of the camera
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Item Properties", meta = 
(AllowPrivateAccess = "true"))
	FVector CameraTargetLocation;

	//true when interping
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Item Properties", meta =
(AllowPrivateAccess = "true"))
	bool bInterping;

	//plays when we start interping
	FTimerHandle ItemInterpTimer;

	//duration of the curve and the timer
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Item Properties", meta
= (AllowPrivateAccess = "true"))
	float ZCurveTime;



//x and y for the item while interping in the equipinterping state

	float ItemInterpX;
	float ItemInterpY;

	//initial yaw offset between the camera and the interping item
	float InterpInitialYawOffset;

	//curve used to scale the item when interping
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Item Properties", meta
= (AllowPrivateAccess = "true"))
	UCurveFloat* ItemScaleCurve;


UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Item Properties", meta =
(AllowPrivateAccess = "true"))
	int32 InterpLocIndex;
```

Item.cpp

```
AItem::AItem():



//item interp variables
	ZCurveTime(0.7f),
	ItemInterpStartLocation(FVector(0.f)),
	CameraTargetLocation(FVector(0.f)),
	bInterping(false),
	ItemInterpX(0.f),
	ItemInterpY(0.f),
	InterpInitialYawOffset(0.f),
	ItemType(EItemType::EIT_MAX),
	InterpLocIndex(0),
	SlotIndex(0)


void AItem::FinishInterping()
{
	bInterping = false;
	if (Character)
	{
			//substract 1 from the item count of the interp location struct
			Character->IncrementInterpLocItemCount(InterpLocIndex, -1);
			Character->GetPickupItem(this);
		}
		//set scale back to normal
		SetActorScale3D(FVector(1.f));
}



void AItem::ItemInterp(float DeltaTime)
{
		if (!bInterping) return;

		if (Character && ItemZCurve)
		{
			//elapsed time since ItemInterpTimer
			const float ElapsedTime =
GetWorldTimerManager().GetTimerElapsed(ItemInterpTimer);
			//Getcurve value corresponding to ElapsedTime
			const float CurveValue = ItemZCurve->GetFloatValue(ElapsedTime);

			FVector ItemLocation = ItemInterpStartLocation;
			//get loction of the camera
			const FVector CameraInterpLocation{ GetInterpLocation() };

			//Vector from item to camera interp location X and Y are zero out
			const FVector ItemToCamera{ FVector(0.f,0.f,(CameraInterpLocation -
ItemLocation).Z) };
			//scale factor to multiply with curvevalue
			const float DeltaZ = ItemToCamera.Size();

			//interpolated X value
			const FVector CurrentLocation{ GetActorLocation() };
			const float InterpXValue = FMath::FInterpTo(
				CurrentLocation.X,
				CameraInterpLocation.X,
				DeltaTime,
				30.0f);

			//interpolated Y value
			const float InterpYValue = FMath::FInterpTo(
				CurrentLocation.Y,
				CameraInterpLocation.Y,
				DeltaTime,
				30.0f);

			//set X and Y Of ItemLocation to interped values
			ItemLocation.X = InterpXValue;
			ItemLocation.Y = InterpYValue;


				//Adding curve value to the Z component of the initial location (
scaled by DeltaZ)
				ItemLocation.Z += CurveValue * DeltaZ;
			SetActorLocation(ItemLocation, true, nullptr,
ETeleportType::TeleportPhysics);
```

```
		// Camera rotation this frame
		const FRotator CameraRotation{ Character->GetFollowCamera()-
>GetComponentRotation() };
		// Camera rotation plus inital Yaw Offset
		FRotator ItemRotation{ 0.f, CameraRotation.Yaw + InterpInitialYawOffset,
0.f };
		SetActorRotation(ItemRotation, ETeleportType::TeleportPhysics);


		if (ItemScaleCurve)
		{
			const float ScaleCurveValue = ItemScaleCurve-
>GetFloatValue(ElapsedTime);
			SetActorScale3D(FVector(ScaleCurveValue, ScaleCurveValue,
ScaleCurveValue));
		}
	}
}



FVector AItem::GetInterpLocation()
{
	if (Character == nullptr) return FVector(0.f);

	switch (ItemType)
	{
	case EItemType::EIT_Ammo:
		return Character->GetInterpLocation(InterpLocIndex).SceneComponent-
>GetComponentLocation();
		break;

	case EItemType::EIT_Weapon:
		return Character->GetInterpLocation(0).SceneComponent-
>GetComponentLocation();
		break;
	}

	return FVector();
}



// Called every frame
void AItem::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	//handle item interping when in interping state
	ItemInterp(DeltaTime);

}



void AItem::StartItemCurve(AShooterCharacter* Char, bool bForcePlaySound)
{
	//store a handle to the character
	Character = Char;

	// get array index in interplocation with lowest item count
	InterpLocIndex = Character->GetInterpLocationIndex();
	//Add 1 to the item count for this interp location struct
	Character->IncrementInterpLocItemCount(InterpLocIndex, 1);

	PlayPickupSound(bForcePlaySound);


	ItemInterpStartLocation = GetActorLocation();

	bInterping = true;
	SetItemState(EItemState::EIS_EquipInterping);

	GetWorldTimerManager().SetTimer(ItemInterpTimer,
		this,
		&AItem::FinishInterping,
		ZCurveTime);

	// Get initial Yaw of the Camera
	const float CameraRotationYaw{ Character->GetFollowCamera()-
>GetComponentRotation().Yaw };
	// Get initial Yaw of the Item
	const float ItemRotationYaw{ GetActorRotation().Yaw };
	// Initial Yaw offset between Camera and Item
	InterpInitialYawOffset = ItemRotationYaw - CameraRotationYaw;

}
```

5.8 Component: Weapon

This component shows how releases bullet from the muzzle socket, reload the weapon, pick up a weapon, drop or swap a weapon. Also it show how ammo can be picked up on the ground and automatically increase the ammo capacity

Weapon.cpp

```
AWeapon::AWeapon() :
	ThrowWeaponTime(0.7f),
	bFalling(false),
	Ammo(30),
	MagazineCapacity(30),
	WeaponType(EWeaponType::EWT_SubmachineGun),
	AmmoType(EAmmoType::EAT_9mm),
	ReloadMontageSection(FName(TEXT("reload SMG"))),
	ClipBoneName(TEXT("smg_clip"))

{
	//PrimaryActorTick.bCanEverTick = true;
}

void AWeapon::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);
	//keep weapon upright
	if (GetItemState() == EItemState::EIS_Falling && bFalling)
	{
		const FRotator MeshRotation{ 0.f,
			GetItemMesh()->GetComponentRotation().Yaw,0.f };
		GetItemMesh()->SetWorldRotation(MeshRotation,
			false, nullptr, ETeleportType::TeleportPhysics);
	}
}
void AWeapon::ThrowWeapon()
{
	FRotator MeshRotation(0.f, GetItemMesh()->GetComponentRotation().Yaw, 0.f);
	GetItemMesh()->SetWorldRotation(MeshRotation, false, nullptr,
ETeleportType::TeleportPhysics);

	const FVector MeshForward{ GetItemMesh()->GetForwardVector() };
	const FVector MeshRight{ GetItemMesh()->GetRightVector() };
	//direction in which we throw weapon
	FVector ImpulseDirection = MeshRight.RotateAngleAxis(-20.f, MeshForward);

	float RandomRotation{ 30.f};
	ImpulseDirection = ImpulseDirection.RotateAngleAxis(RandomRotation, FVector(0.f,
0.f, 1.f));
	ImpulseDirection *= 20'000;
	GetItemMesh()->AddImpulse(ImpulseDirection);

	bFalling = true;
	GetWorldTimerManager().SetTimer(
			ThrowWeaponTimer,
			this,
			&AWeapon::StopFalling,
			ThrowWeaponTime);
}

void AWeapon::DecrementAmmo()
{
	if (Ammo - 1 <= 0)
	{
		Ammo = 0;
	}
	else
	{
		--Ammo;
	}
}

void AWeapon::ReloadAmmo(int32 Amount)
{
	checkf(Ammo + Amount <= MagazineCapacity, TEXT("Attempted tpo reload with more
than magazine capacity!"));
	Ammo += Amount;
}

bool AWeapon::ClipIsFull()
{
	return Ammo >= MagazineCapacity;
}

void AWeapon::StopFalling()
{
	bFalling = false;
	SetItemState(EItemState::EIS_Pickup);
}
```

Weapon.h

```
UENUM(BlueprintType)
enum class EWeaponType : uint8
{
	EWT_SubmachineGun UMETA(DisplayName = "SubmachineGun"),
	EWT_AssaultRifle UMETA(DisplayName="AssaultRifle"),

	EWT_MAX UMETA(DsiplayName= "DefaultMAX")
};

USTRUCT(BlueprintType)
struct FWeaponDataTable : public FTableRowBase
{
    GENERATED_BODY()

		UPROPERTY(EditAnywhere, BlueprintReadWrite)
		EAmmoType AmmoType;

	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	int32 WeaponAmmo;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	int32 MagazingCapacity;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	class USoundCue* PickupSound;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	USoundCue* EquipSound;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	USkeletalMesh* ItemMesh;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	FString ItemName;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	UTexture2D* InventoryIcon;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	UTexture2D* AmmoIcon;

//mat instance and mat index here


UPROPERTY(EditAnywhere, BlueprintReadWrite)
	FName ClipBoneName;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	FName ReloadMontageSection;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	TSubclassOf<UAnimInstance> AnimBP;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	UTexture2D* CrosshairsMiddle;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	UTexture2D* CrosshairsLeft;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	UTexture2D* CrosshairsRight;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	UTexture2D* CrosshairsBottom;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	UTexture2D* CrosshairsTop;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float AutoFireRate;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	class UParticleSystem* MuzzleFlash;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
	USoundCue* FireSound;
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
		FName BoneToHide;

	UPROPERTY(EditAnywhere, BlueprintReadWrite)
		bool bAutomatic;

};

UCLASS()
class MYPROJECT8_API AWeapon : public AItem
{
	GENERATED_BODY()
public:
	AWeapon();

	virtual void Tick(float DeltaTime)override;
protected:

	void StopFalling();



private:
	FTimerHandle ThrowWeaponTimer;
	float ThrowWeaponTime;
	bool bFalling;

	//Ammo count for this weapon
	UPROPERTY(EditAnywhere,BlueprintReadWrite,Category= "Weapon Properties",
		meta = (AllowPrivateAccess= "true"))
	int32 Ammo;

	//max ammo that the weapon can hold
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Weapon Properties",
		meta = (AllowPrivateAccess = "true"))
	int32 MagazineCapacity;


	//type of weapon
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Weapon Properties",
		meta = (AllowPrivateAccess = "true"))
	EWeaponType WeaponType;


	//the type of ammo for this weapon
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Weapon Properties",
		meta = (AllowPrivateAccess = "true"))
	EAmmoType AmmoType;

	//FName for the reload montage section
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Weapon Properties",
		meta = (AllowPrivateAccess = "true"))
	FName ReloadMontageSection;

	//true when moving the clip while reloading
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Weapon Properties",
		meta = (AllowPrivateAccess = "true"))
	bool bMovingClip;

	//name for the clip bone
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Weapon Properties",
		meta = (AllowPrivateAccess = "true"))
	FName ClipBoneName;


public:
	//adds impulse to weapon
	void ThrowWeapon();

	FORCEINLINE int32 GetAmmo() const { return Ammo; }
	FORCEINLINE int32 GetMagazineCapacity() const { return MagazineCapacity; }

	//called from character class when firing weapon
	void DecrementAmmo();

	FORCEINLINE EWeaponType GetWeaponType() const { return WeaponType; }
	FORCEINLINE EAmmoType GetAmmoType() const { return AmmoType; }
	FORCEINLINE FName GetReloadMontageSection() const { return ReloadMontageSection; }
	FORCEINLINE FName GetClipBoneName() const { return ClipBoneName; }

void ReloadAmmo(int32 Amount);

	FORCEINLINE void SetMovingClip(bool Move) { bMovingClip = Move; }

	bool ClipIsFull();
};
```

Item.h

```
UENUM(BlueprintType)
enum class EItemState : uint8
{
	EIS_Pickup UMETA(DisplayName = "Pickup"),
	EIS_EquipInterping UMETA(DisplayName = "EquipInterping"),
	EIS_PickedUp UMETA(DisplayName = "PickedUp"),
	EIS_Equipped UMETA(DisplayName = "Equipped"),
	EIS_Falling UMETA(DisplayName = "Falling"),

	EIS_MAX UMETA(DisplayName = "DefaultMAX")
};


UENUM(BlueprintType)
enum class EItemType : uint8
{
	EIT_Ammo UMETA(DisplayName = "Ammo"),
	EIT_Weapon UMETA(DisplayName = "Weapon"),

	EIT_MAX UMETA(DisplayName = "DefaultMAX")
};
```

```
UFUNCTION()
	void OnSphereOverlap(UPrimitiveComponent* OverlappedComponent,
		AActor*OtherActor,
		UPrimitiveComponent*OtherComp,
		int32 OtherBodyIndex,
		bool bFromSweep,
		const FHitResult& SweepResult);


protected:

	UFUNCTION()
	void OnSphereEndOverlap(UPrimitiveComponent* OverlappedComponent,
		AActor* OtherActor,
		UPrimitiveComponent* OtherComp,
		int32 OtherBodyIndex);



//Sets properties of the items components based on state
	virtual void SetItemProperties(EItemState State);

	//called when ItemInterpTimer is finished
	void FinishInterping();

	//handles item interpolation when in the equipinterping state
	void ItemInterp(float DeltaTime);

	FVector GetInterpLocation();

	void PlayPickupSound(bool bForcePlaySound = false);

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;

	//called in AShooterCharacter::GetPickupItem
	void PlayEquipSound(bool bForcePlaySound = false);

private:

//sound played when item is picked up
          UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Item Properties", meta
= (AllowPrivateAccess = "true"))
     class USoundCue* PickupSound;

	//sound played when item is equipped
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Item Properties", meta =
(AllowPrivateAccess = "true"))
	USoundCue* EquipSound;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Item Properties", meta =
(AllowPrivateAccess = "true"))
	EItemType ItemType;



public:
	FORCEINLINE UWidgetComponent* GetPickupWidget() const { return PickupWidget; }
	FORCEINLINE USphereComponent* GetAreaSphere() const { return AreaSphere; }
	FORCEINLINE UBoxComponent* GetCollisionBox() const { return CollisionBox; }
	FORCEINLINE EItemState GetItemState() const { return ItemState; }
	void SetItemState(EItemState State);
	FORCEINLINE USkeletalMeshComponent* GetItemMesh() const { return ItemMesh; }
      FORCEINLINE USoundCue* GetPickupSound() const { return PickupSound; }
	FORCEINLINE void SetPickupSound(USoundCue* Sound) { PickupSound = Sound; }
	FORCEINLINE USoundCue* GetEquipSound() const { return EquipSound; }
	FORCEINLINE void SetEquipSound(USoundCue* Sound) { EquipSound = Sound; }
	FORCEINLINE int32 GetItemCount() const { return ItemCount; }
	FORCEINLINE void SetItemName(FString Name) { ItemName = Name; }

	FORCEINLINE int32 GetSlotIndex() const { return SlotIndex;  }
	FORCEINLINE void SetSlotIndex(int32 Index) { SlotIndex = Index;}
```

Item.cpp

```
AItem::AItem():
	ItemName(FString("Default")),
	ItemCount(0),
	ItemRarity(EItemRarity::EIR_Common),
	ItemState(EItemState::EIS_Pickup),

ItemMesh = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("ItemMesh"));
	SetRootComponent(ItemMesh);

	CollisionBox = CreateDefaultSubobject<UBoxComponent>(TEXT("CollisionBox"));
	CollisionBox->SetupAttachment(ItemMesh);

	CollisionBox->SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
	CollisionBox->SetCollisionResponseToChannel(
		ECollisionChannel::ECC_Visibility,
		ECollisionResponse::ECR_Block);

	PickupWidget = CreateDefaultSubobject<UWidgetComponent>(TEXT("PickupWidget"));
	PickupWidget->SetupAttachment(GetRootComponent());

	AreaSphere = CreateDefaultSubobject<USphereComponent>(TEXT("AreaSphere"));
	AreaSphere->SetupAttachment(GetRootComponent());

void AItem::BeginPlay()
{

	// Hide Pickup Widget
	if (PickupWidget)
	{
		PickupWidget->SetVisibility(false);
	}


	//setup overlap for AreaSphere
	AreaSphere->OnComponentBeginOverlap.AddDynamic(this, &AItem::OnSphereOverlap);
	AreaSphere->OnComponentEndOverlap.AddDynamic(this, &AItem::OnSphereEndOverlap);

	// Set Item properties based on ItemState
	SetItemProperties(ItemState);
}


void AItem::OnSphereOverlap(UPrimitiveComponent* OverlappedComponent,
	AActor* OtherActor, UPrimitiveComponent* OtherComp,
	int32 OtherBodyIndex, bool bFromSweep, const FHitResult& SweepResult)
{
	if (OtherActor)
	{
		AShooterCharacter* ShooterCharacter = Cast<AShooterCharacter>(OtherActor);
		if (ShooterCharacter)
		{
			ShooterCharacter->IncrementOverlappedItemCount(1);
		}

	}
}

void AItem::OnSphereEndOverlap(UPrimitiveComponent* OverlappedComponent,
	AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex)
{
	if (OtherActor)
	{
		AShooterCharacter* ShooterCharacter = Cast<AShooterCharacter>(OtherActor);
		if (ShooterCharacter)
		{
			ShooterCharacter->IncrementOverlappedItemCount(-1);
		}

	}
}

void AItem::SetItemProperties(EItemState State)
{
	switch (State)
	{
	case EItemState::EIS_Pickup:
		// Set Mesh properties
		ItemMesh->SetSimulatePhysics(false);
		ItemMesh->SetEnableGravity(false);
		ItemMesh->SetVisibility(true);
		ItemMesh-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
		ItemMesh->SetCollisionEnabled(ECollisionEnabled::NoCollision);
		// Set AreaSphere properties
		AreaSphere-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Overlap);
		AreaSphere->SetCollisionEnabled(ECollisionEnabled::QueryOnly);
		// Set CollisionBox properties
		CollisionBox-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
		CollisionBox->SetCollisionResponseToChannel(
			ECollisionChannel::ECC_Visibility,
			ECollisionResponse::ECR_Block);
		CollisionBox->SetCollisionEnabled(ECollisionEnabled::QueryAndPhysics);
		break;
	case EItemState::EIS_Equipped:
			PickupWidget->SetVisibility(false);
			// Set Mesh properties
			ItemMesh->SetSimulatePhysics(false);
			ItemMesh->SetEnableGravity(false);
			ItemMesh->SetVisibility(true);
			ItemMesh-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
			ItemMesh->SetCollisionEnabled(ECollisionEnabled::NoCollision);
			// Set AreaSphere properties
			AreaSphere-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
			AreaSphere->SetCollisionEnabled(ECollisionEnabled::NoCollision);
			// Set CollisionBox properties
			CollisionBox-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
			CollisionBox->SetCollisionEnabled(ECollisionEnabled::NoCollision);
			break;
		case EItemState::EIS_Falling:
			// Set mesh properties
			ItemMesh->SetCollisionEnabled(ECollisionEnabled::QueryAndPhysics);
			ItemMesh->SetSimulatePhysics(true);
			ItemMesh->SetEnableGravity(true);
			ItemMesh->SetVisibility(true);
			ItemMesh-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
			ItemMesh->SetCollisionResponseToChannel(
				ECollisionChannel::ECC_WorldStatic,
				ECollisionResponse::ECR_Block);
			// Set AreaSphere properties
			AreaSphere-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
			AreaSphere->SetCollisionEnabled(ECollisionEnabled::NoCollision);
			// Set CollisionBox properties
			CollisionBox-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
			CollisionBox->SetCollisionEnabled(ECollisionEnabled::NoCollision);
			break;
		case EItemState::EIS_EquipInterping:
			PickupWidget->SetVisibility(false);
			// Set mesh properties
			ItemMesh->SetSimulatePhysics(false);
			ItemMesh->SetEnableGravity(false);
			ItemMesh->SetVisibility(true);
			ItemMesh-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
			ItemMesh->SetCollisionEnabled(ECollisionEnabled::NoCollision);
			// Set AreaSphere properties
			AreaSphere-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
			AreaSphere->SetCollisionEnabled(ECollisionEnabled::NoCollision);
			// Set CollisionBox properties
			CollisionBox-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
			CollisionBox->SetCollisionEnabled(ECollisionEnabled::NoCollision);
			break;
		case EItemState::EIS_PickedUp:
			PickupWidget->SetVisibility(false);
			// Set mesh properties
```

```
                ItemMesh->SetSimulatePhysics(false);
                ItemMesh->SetEnableGravity(false);
                ItemMesh->SetVisibility(false);
                ItemMesh-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
                ItemMesh->SetCollisionEnabled(ECollisionEnabled::NoCollision);
                // Set AreaSphere properties
                AreaSphere-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
                AreaSphere->SetCollisionEnabled(ECollisionEnabled::NoCollision);
                // Set CollisionBox properties
                CollisionBox-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
                CollisionBox->SetCollisionEnabled(ECollisionEnabled::NoCollision);
                break;
        }
}
```

5.9 Component: Weapon rarity

This component shows the different rarity available which can be assigned to each weapon along with a Data Table which is used to easily change properties of each weapon with the editor

Item.h

```
UENUM(BlueprintType)
enum class EItemRarity : uint8
{
	EIR_Damaged UMETA(DisplayName = "Damaged"),
	EIR_Common UMETA(DisplayName = "Common"),
	EIR_Uncommon UMETA(DisplayName = "Uncommon"),
	EIR_Rare UMETA(DisplayName = "Rare"),
	EIR_Legendary UMETA(DisplayName = "Legendary"),

	EIR_MAX UMETA(DisplayName = "DefaultMAX")
};



private:

/** Item Rarity data table */
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = DataTable, meta =
(AllowPrivateAccess = "true"))
		class UDataTable* ItemRarityDataTable;

	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Rarity, meta =
(AllowPrivateAccess = "true"))
		int32 NumberOfStars;

	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Rarity, meta =
(AllowPrivateAccess = "true"))
	UTexture2D* IconBackground;


void AItem::BeginPlay()
{


	//sets active stars array passed on rarity
	SetActiveStars();


}

void AItem::SetActiveStars()
{
	// The 0 element is not used
	for (int32 i = 0; i <= 5; i++)
	{
		ActiveStars.Add(false);
	}

	switch (ItemRarity)
	{
	case EItemRarity::EIR_Damaged:
		ActiveStars[1] = true;
		break;
	case EItemRarity::EIR_Common:
		ActiveStars[1] = true;
		ActiveStars[2] = true;
		break;
	case EItemRarity::EIR_Uncommon:
		ActiveStars[1] = true;
		ActiveStars[2] = true;
		ActiveStars[3] = true;
		break;
	case EItemRarity::EIR_Rare:
		ActiveStars[1] = true;
		ActiveStars[2] = true;
		ActiveStars[3] = true;
		ActiveStars[4] = true;
		break;
	case EItemRarity::EIR_Legendary:
		ActiveStars[1] = true;
		ActiveStars[2] = true;
		ActiveStars[3] = true;
		ActiveStars[4] = true;
		ActiveStars[5] = true;
		break;
	}
}

void AItem::OnConstruction(const FTransform& Transform)
{
	// Path to the Item Rarity Data Table
	FString
RarityTablePath(TEXT("DataTable'/Game/DataTable/ItemRarityTable.ItemRarityTable'"));
	UDataTable* RarityTableObject =
Cast<UDataTable>(StaticLoadObject(UDataTable::StaticClass(), nullptr, *RarityTablePath));

	if (RarityTableObject)
	{

		FItemRarityTable* RarityRow = nullptr;
		switch (ItemRarity)
		{
		case EItemRarity::EIR_Damaged:
			RarityRow = RarityTableObject-
>FindRow<FItemRarityTable>(FName("Damaged"), TEXT(""));
			break;
		case EItemRarity::EIR_Common:
			RarityRow = RarityTableObject-
>FindRow<FItemRarityTable>(FName("Common"), TEXT(""));
			break;
		case EItemRarity::EIR_Uncommon:
			RarityRow = RarityTableObject-
>FindRow<FItemRarityTable>(FName("Uncommon"), TEXT(""));
			break;
		case EItemRarity::EIR_Rare:
			RarityRow = RarityTableObject-
>FindRow<FItemRarityTable>(FName("Rare"), TEXT(""));
			break;
		case EItemRarity::EIR_Legendary:
			RarityRow = RarityTableObject-
>FindRow<FItemRarityTable>(FName("Legendary"), TEXT(""));
			break;
		}

		if (RarityRow)
		{

			NumberOfStars = RarityRow->NumberOfStars;
			IconBackground = RarityRow->IconBackground;

		}
	}
}
```

Weapon.h

```
private:

//data table for weapon properties
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = DataTable, meta =
(AllowPrivateAccess = "true"))
	UDataTable* WeaponDataTable;

protected:


	virtual void OnConstruction(const FTransform& Transform) override;
```

Weapon.cpp

```
void AWeapon::OnConstruction(const FTransform& Transform)
{
	Super::OnConstruction(Transform);
	const FString
WeaponTablePath(TEXT("DataTable'/Game/DataTable/WeaponDataTable.WeaponDataTable'"));
	UDataTable* WeaponTableObject = Cast<UDataTable>
		(StaticLoadObject(UDataTable::StaticClass(), nullptr, *WeaponTablePath));


	if (WeaponTableObject)
	{
		FWeaponDataTable* WeaponDataRow = nullptr;
		switch (WeaponType)
		{
		case EWeaponType::EWT_SubmachineGun:
			WeaponDataRow = WeaponTableObject-
>FindRow<FWeaponDataTable>(FName("SubmachineGun"), TEXT(""));
			break;
		case EWeaponType::EWT_AssaultRifle:
			WeaponDataRow = WeaponTableObject-
>FindRow<FWeaponDataTable>(FName("AssaultRifle"), TEXT(""));
			break;

		}

		if (WeaponDataRow)
		{
			AmmoType = WeaponDataRow->AmmoType;
			Ammo = WeaponDataRow->WeaponAmmo;
			MagazineCapacity = WeaponDataRow->MagazingCapacity;
			SetPickupSound(WeaponDataRow->PickupSound);
			SetEquipSound(WeaponDataRow->EquipSound);
			GetItemMesh()->SetSkeletalMesh(WeaponDataRow->ItemMesh);
			SetItemName(WeaponDataRow->ItemName);


			Damage = WeaponDataRow->Damage;
			HeadShotDamage = WeaponDataRow->HeadShotDamage;
		}
	}
}
```

5.10 Enemy movement

Enemy.h

```
private:

//Behavior tree for the AI Character
	UPROPERTY(EditAnywhere, Category = "Behavior Tree", meta = (AllowPrivateAccess =
"true"))
		class UBehaviorTree* BehaviorTree;

//Point for the enemy to move to
	UPROPERTY(EditAnywhere, Category = "Behavior Tree", meta = (AllowPrivateAccess =
"true", MakeEditWidget = "true"))
		FVector PatrolPoint;

	//Second point for the enemy to move to
	UPROPERTY(EditAnywhere, Category = "Behavior Tree", meta = (AllowPrivateAccess =
"true", MakeEditWidget = "true"))
		FVector PatrolPoint2;

class AEnemyController* EnemyController;



public:

FORCEINLINE UBehaviorTree* GetBehaviorTree() const { return BehaviorTree; }
```

EnemyController.h

```
public:
	AEnemyController();
	virtual void OnPossess(APawn* InPawn) override;

private:
	//Blackboard component for this enemy
	UPROPERTY(BlueprintReadWrite, Category = "AI Behavior", meta = (AllowPrivateAccess
= "true"))
		class UBlackboardComponent* BlackboardComponent;
	//Behavior Tree for this enemy
	UPROPERTY(BlueprintReadWrite, Category = "AI Behavior", meta = (AllowPrivateAccess
= "true"))
		class UBehaviorTreeComponent* BehaviorTreeComponent;

public:
	FORCEINLINE UBlackboardComponent* GetBlackboardComponent() const { return
BlackboardComponent; }
```

Enemy.cpp

```
#include "EnemyController.h"
#include "BehaviorTree/BlackboardComponent.h"
#include "BehaviorTree/BehaviorTreeComponent.h"
#include "BehaviorTree/BehaviorTree.h"
#include "Enemy.h"

AEnemyController::AEnemyController()
{
	BlackboardComponent =
		CreateDefaultSubobject<UBlackboardComponent>(TEXT("BlackboardComponent"));
	check(BlackboardComponent);
	BehaviorTreeComponent =

	CreateDefaultSubobject<UBehaviorTreeComponent>(TEXT("BehaviorTreeComponent"));
	check(BehaviorTreeComponent);
}
void AEnemyController::OnPossess(APawn* InPawn)
{
	Super::OnPossess(InPawn);
	if (InPawn == nullptr) return;
	AEnemy* Enemy = Cast<AEnemy>(InPawn);
	if (Enemy)
	{
		if (Enemy->GetBehaviorTree())
		{
			BlackboardComponent->InitializeBlackboard(*(Enemy->GetBehaviorTree()
-> BlackboardAsset));
		}
	}
}


//Get the AI Controller
	EnemyController = Cast<AEnemyController>(GetController());

	if (EnemyController)
	{
		EnemyController->GetBlackboardComponent()->SetValueAsBool(
			FName("CanAttack"),
			true);
	}

	const FVector WorldPatrolPoint = UKismetMathLibrary::TransformLocation(
		GetActorTransform(),
		PatrolPoint);

	const FVector WorldPatrolPoint2 = UKismetMathLibrary::TransformLocation(
		GetActorTransform(),
		PatrolPoint2);

	if (EnemyController)
	{
		EnemyController->GetBlackboardComponent()->SetValueAsVector(
			TEXT("PatrolPoint"),
			WorldPatrolPoint);
		EnemyController->GetBlackboardComponent()->SetValueAsVector(
			TEXT("PatrolPoint2"),
			WorldPatrolPoint2);

		EnemyController->RunBehaviorTree(BehaviorTree);

	}
```

5.11 Enemy attack

Enemy.h

```
protected:

//Called when something overlaps with the agro sphere
	UFUNCTION()
		void AgroSphereOverlap(UPrimitiveComponent* OverlappedComponent,
			AActor* OtherActor,
			UPrimitiveComponent* OtherComp,
			int32 OtherBodyIndex,
			bool bFromSweep,
			const FHitResult& SweepResult);

	UFUNCTION()
		void CombatRangeOverlap(
			UPrimitiveComponent* OverlappedComponent,
			AActor* OtherActor,
			UPrimitiveComponent* OtherComp,
			int32 OtherBodyIndex,
			bool bFromSweep,
			const FHitResult& SweepResult);
	UFUNCTION()
		void CombatRangeEndOverlap(
			UPrimitiveComponent* OverlappedComponent,
			AActor* OtherActor,
			UPrimitiveComponent* OtherComp,
			int32 OtherBodyIndex);



//Activate/deactivate collision for weapon boxes
	UFUNCTION(BlueprintCallable)
		void ActivateLeftWeapon();
	UFUNCTION(BlueprintCallable)
		void DeactivateLeftWeapon();
	UFUNCTION(BlueprintCallable)
		void ActivateRightWeapon();
	UFUNCTION(BlueprintCallable)
		void DeactivateRightWeapon();
```

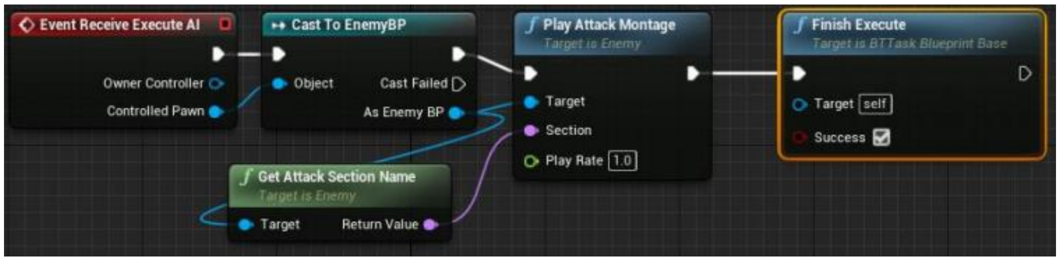
```
private:

 //Overlap sphere for when the enemy becomes hostile
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess =
		"true"))
		class USphereComponent* AgroSphere;

//True when in attack range; time to attack!
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess
		= "true"))
		bool bInAttackRange;

//Sphere for attack range
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess =
		"true"))
		USphereComponent* CombatRangeSphere;


//Collision volume for the left weapon
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess
		= "true"))
		class UBoxComponent* LeftWeaponCollision;

//Collision volume for the right weapon
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess
		= "true"))
		UBoxComponent* RightWeaponCollision;
```

Enemy.cpp

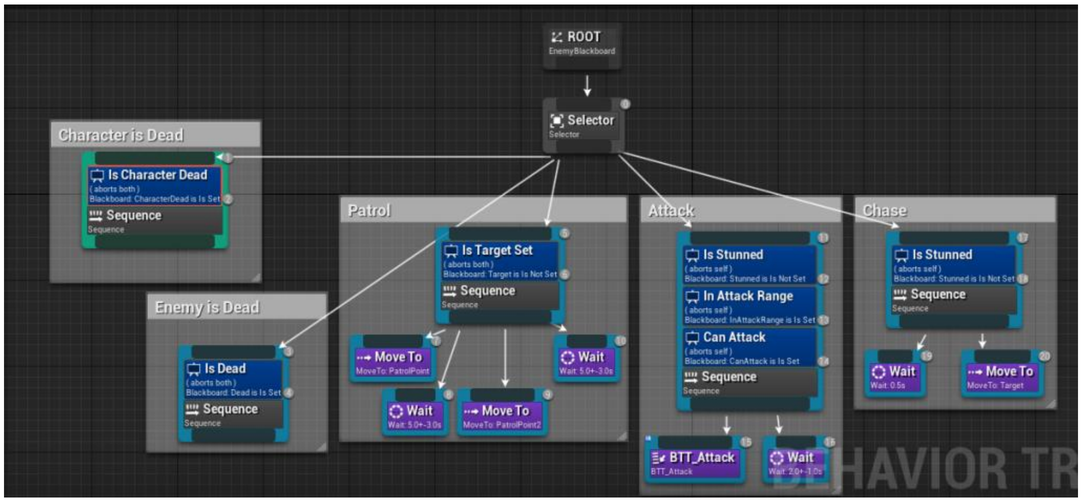
```
AgroSphere = CreateDefaultSubobject<USphereComponent>(TEXT("AgroSphere"));
	AgroSphere->SetupAttachment(GetRootComponent());

//Create the Combat Range Sphere
	CombatRangeSphere = CreateDefaultSubobject<USphereComponent>(TEXT("CombatRange"));
	CombatRangeSphere->SetupAttachment(GetRootComponent());

//Construct left and right weapon collision boxes
	LeftWeaponCollision = CreateDefaultSubobject<UBoxComponent>(TEXT("Left Weapon
Box"));
	LeftWeaponCollision->SetupAttachment(GetMesh(), FName("LeftHandBone"));

	RightWeaponCollision = CreateDefaultSubobject<UBoxComponent>(TEXT("Right Weapon
Box"));
	RightWeaponCollision->SetupAttachment(GetMesh(), FName("RightHandBone"));

//Set collision presets for weapon boxes
	LeftWeaponCollision->SetCollisionEnabled(ECollisionEnabled::NoCollision);
	LeftWeaponCollision->SetCollisionObjectType(ECollisionChannel::ECC_WorldDynamic);
	LeftWeaponCollision-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
	LeftWeaponCollision->SetCollisionResponseToChannel(
		ECollisionChannel::ECC_Pawn,
		ECollisionResponse::ECR_Overlap);

	RightWeaponCollision->SetCollisionEnabled(ECollisionEnabled::NoCollision);
	RightWeaponCollision->SetCollisionObjectType(ECollisionChannel::ECC_WorldDynamic);
	RightWeaponCollision-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Ignore);
	RightWeaponCollision->SetCollisionResponseToChannel(
		ECollisionChannel::ECC_Pawn,
		ECollisionResponse::ECR_Overlap);


void AEnemy::AgroSphereOverlap(UPrimitiveComponent* OverlappedComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const
FHitResult& SweepResult)
{
	if (OtherActor == nullptr) return;
	auto Character = Cast<AShooterCharacter>(OtherActor);
	if (Character)
	{
		//Set the value of the Target Blackboard Keys
		EnemyController->GetBlackboardComponent()->SetValueAsObject(
			TEXT("Target"),
			Character);
	}
}

void AEnemy::CombatRangeOverlap(UPrimitiveComponent* OverlappedComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const
FHitResult& SweepResult)
{
	if (OtherActor == nullptr) return;
	//Use the following in your case:
	auto ShooterCharacter = Cast<AShooterCharacter>(OtherActor);
	if (ShooterCharacter)
	{
		bInAttackRange = true;
		if (EnemyController)
		{
			EnemyController->GetBlackboardComponent()->SetValueAsBool(
				TEXT("InAttackRange"),
				true);
		}
	}
}

void AEnemy::CombatRangeEndOverlap(UPrimitiveComponent* OverlappedComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex)
{
	if (OtherActor == nullptr) return;
	//Use the following in your case:
	auto ShooterCharacter = Cast<AShooterCharacter>(OtherActor);
	if (ShooterCharacter)
	{
		bInAttackRange = false;
```

```
                if (EnemyController)
                {
                        EnemyController->GetBlackboardComponent()->SetValueAsBool(
                                TEXT("InAttackRange"),
                                false);
                }
        }
}


void AEnemy::ActivateLeftWeapon()
{
        LeftWeaponCollision->SetCollisionEnabled(ECollisionEnabled::QueryOnly);
}

void AEnemy::DeactivateLeftWeapon()
{
        LeftWeaponCollision->SetCollisionEnabled(ECollisionEnabled::NoCollision);
}

void AEnemy::ActivateRightWeapon()
{
        RightWeaponCollision->SetCollisionEnabled(ECollisionEnabled::QueryOnly);
}

void AEnemy::DeactivateRightWeapon()
{
        RightWeaponCollision->SetCollisionEnabled(ECollisionEnabled::NoCollision);
}



void AEnemy::OnLeftWeaponOverlap(UPrimitiveComponent* OverlappedComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const
FHitResult& SweepResult)
{
        auto Character = Cast<AShooterCharacter>(OtherActor);
        if (Character)
        {
                DoDamage(Character);
                SpawnBlood(Character, LeftWeaponSocket);
        }
}

void AEnemy::OnRightWeaponOverlap(UPrimitiveComponent* OverlappedComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const
FHitResult& SweepResult)
{

        auto Character = Cast<AShooterCharacter>(OtherActor);
        if (Character)
        {
                DoDamage(Character);
                SpawnBlood(Character, RightWeaponSocket);
        }
}
```

Enemy behaviour tree blueprint



Enemy behaviour tree



5.12 Enemy death

Enemy.h

```
protected:

void Die();

bool bDying;

UFUNCTION(BlueprintCallable)
		void FinishDeath();


UFUNCTION()
	void DestroyEnemy();



private:


FTimerHandle DeathTimer;
	//Time after death until destroy
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess =
		"true"))
		float DeathTime;
```

Enemy.cpp

```
void AEnemy::Die()
{
	HideHealthBar();

	if (bDying) return;
	bDying = true;

	UAnimInstance* AnimInstance = GetMesh()->GetAnimInstance();
	if (AnimInstance && DeathMontage)
	{
		AnimInstance->Montage_Play(DeathMontage);
	}
	if (EnemyController)
	{
		EnemyController->GetBlackboardComponent()->SetValueAsBool(
			FName("Dead"),
			true
		);
		EnemyController->StopMovement();
	}
}

void AEnemy::FinishDeath()
{
	GetMesh()->bPauseAnims = true;
	GetWorldTimerManager().SetTimer(
		DeathTimer,
		this,
		&AEnemy::DestroyEnemy,
		DeathTime
	);
}
```

Enemy animation montages

EnemyAminInstance.h

```
public:


	UFUNCTION(BlueprintCallable)
	void UpdateAnimationProperties(float DeltaTime);

private:

	//Lateral Movement Speed
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Movement, meta =
(AllowPrivateAccess
		= "true"))
		float Speed;
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, meta = (AllowPrivateAccess =
"true"))
		class AEnemy* Enemy;
```

EnemyAminInstance.cpp

```
#include "Enemy.h"

void UEnemyAnimInstance::UpdateAnimationProperties(float DeltaTime)
{
	if (Enemy == nullptr)
	{
		Enemy = Cast<AEnemy>(TryGetPawnOwner());
	}
	if (Enemy)
	{
		FVector Velocity{ Enemy->GetVelocity() };
		Velocity.Z = 0.f;
		Speed = Velocity.Size();
	}
}
```

Enemy.h

```
private:

//Montage containing hit and death animation
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess =
		"true"))
		UAnimMontage* HitMontage;

//Montage containing different attacks
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess =
		"true"))
		UAnimMontage* AttackMontage;

	//The four attack montage section names
	FName AttackA;
	FName AttackB;
	FName AttackC;
	FName AttackD;

protected:

void PlayHitMontage(FName Section, float PlayRate = 1.0f);

UFUNCTION(BlueprintCallable)
		void PlayAttackMontage(FName Section, float PlayRate);

UFUNCTION(BlueprintPure)
		FName GetAttackSectionName();

//Death anim montage for the enemy
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess =
		"true"))
		UAnimMontage* DeathMontage;
```

Enemy.cpp

```
void AEnemy::PlayHitMontage(FName Section, float PlayRate)
{
	UAnimInstance* AnimInstance = GetMesh()->GetAnimInstance();
	if (AnimInstance)
	{
		AnimInstance->Montage_Play(HitMontage, PlayRate);
		AnimInstance->Montage_JumpToSection(Section, HitMontage);
	}
}



void AEnemy::PlayAttackMontage(FName Section, float PlayRate)
{
	UAnimInstance* AnimInstance = GetMesh()->GetAnimInstance();
	if (AnimInstance && AttackMontage)
	{
		AnimInstance->Montage_Play(AttackMontage);
		AnimInstance->Montage_JumpToSection(Section, AttackMontage);
	}
	bCanAttack = false;
	GetWorldTimerManager().SetTimer(
		AttackWaitTimer,
		this,
		&AEnemy::ResetCanAttack,
		AttackWaitTime);

	if (EnemyController)
	{
		EnemyController->GetBlackboardComponent()->SetValueAsBool(
			FName("CanAttack"),
			false);
	}
}

FName AEnemy::GetAttackSectionName()
{
	FName SectionName;
	const int32 Section{ FMath::RandRange(1,4) };
	switch (Section)
	{
	case 1:
		SectionName = AttackA;
		break;
	case 2:
		SectionName = AttackB;
		break;
	case 3:
		SectionName = AttackC;
		break;
	case 4:
		SectionName = AttackD;
		break;
	}
	return SectionName;
}
```

5.13 Enemy health

Weapon.h

```
UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float Damage;

	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float HeadShotDamage;

private:

//Amount of damage caused by a bullet
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Weapon Properties", meta =
		(AllowPrivateAccess = "true"))
		float Damage;
	//Amount of damage when bullet hit the head
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Weapon Properties", meta =
		(AllowPrivateAccess = "true"))
		float HeadShotDamage;
```

Weapon.cpp

```
Damage = WeaponDataRow->Damage;
HeadShotDamage = WeaponDataRow->HeadShotDamage;
```

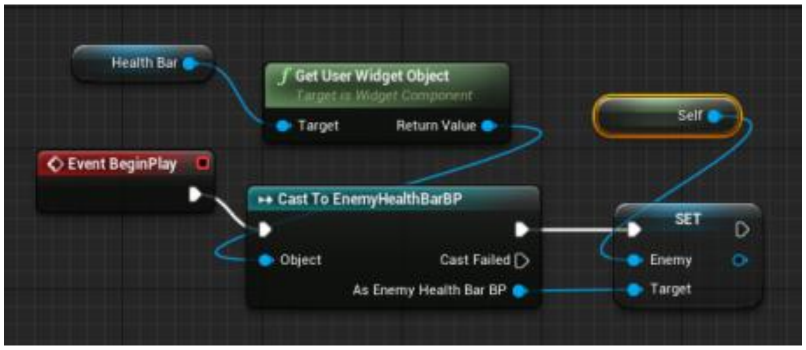
Enemy.h

```
public:

virtual float TakeDamage(float DamageAmount, struct FDamageEvent const& DamageEvent,
		AController* EventInstigator, AActor* DamageCauser) override;

FORCEINLINE FString GetHeadBone() const { return HeadBone; }

private:

//Current health of the enemy
	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Combat, meta =
(AllowPrivateAccess =
		"true"))
		float Health;
	//Maximum health of the enemy
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess =
		"true"))
		float MaxHealth;

	//Name of the head bone
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess =
		"true"))
		FString HeadBone;
```

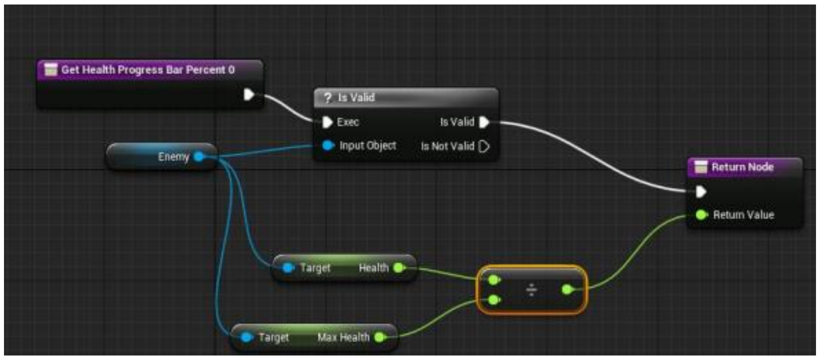
Enemy.cpp

```
float AEnemy::TakeDamage(float DamageAmount, FDamageEvent const& DamageEvent,
AController* EventInstigator, AActor* DamageCauser)
{
	//Set the Target Blackboard Key to agro the Character
	if (EnemyController)
	{
		EnemyController->GetBlackboardComponent()->SetValueAsObject(
			FName("Target"),
			DamageCauser);
	}

	if (Health - DamageAmount <= 0.f)
	{
		Health = 0.f;
		Die();
	}
	else
	{
		Health -= DamageAmount;
	}
	return DamageAmount;
}
```

Enemy Blueprint



Enemy Health Bar Blueprint



5.14 Character health

ShooterCharacter.h

```
public:

//Take combat damage
	virtual float TakeDamage(
		float DamageAmount,
		struct FDamageEvent const& DamageEvent,
		class AController* EventInstigator,
		AActor* DamageCauser) override;

private:

//Character health
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess =
		"true"))
		float Health;

//Character max health
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess =
		"true"))
		float MaxHealth;
```

ShooterCharacter.cpp

```
Health(100.f),
MaxHealth(100.f)

float AShooterCharacter::TakeDamage(float DamageAmount, FDamageEvent const& DamageEvent,
AController* EventInstigator, AActor* DamageCauser)
	{
		if (Health - DamageAmount <= 0.f)
		{
			Health = 0.f;
			Die();

			auto EnemyController = Cast<AEnemyController>(EventInstigator);
			if (EnemyController)
			{
				EnemyController->GetBlackboardComponent()->SetValueAsBool(
					FName(TEXT("CharacterDead")),
					true
				);
			}

		}
		else
		{
```

```
                        Health -= DamageAmount;
                }
                return DamageAmount;
        }


void AShooterCharacter::FinishDeath()
{
        GetMesh()->bPauseAnims = true;
        APlayerController* PC = UGameplayStatics::GetPlayerController(this, 0);
        if (PC)
        {
                DisableInput(PC);
        }
}
```

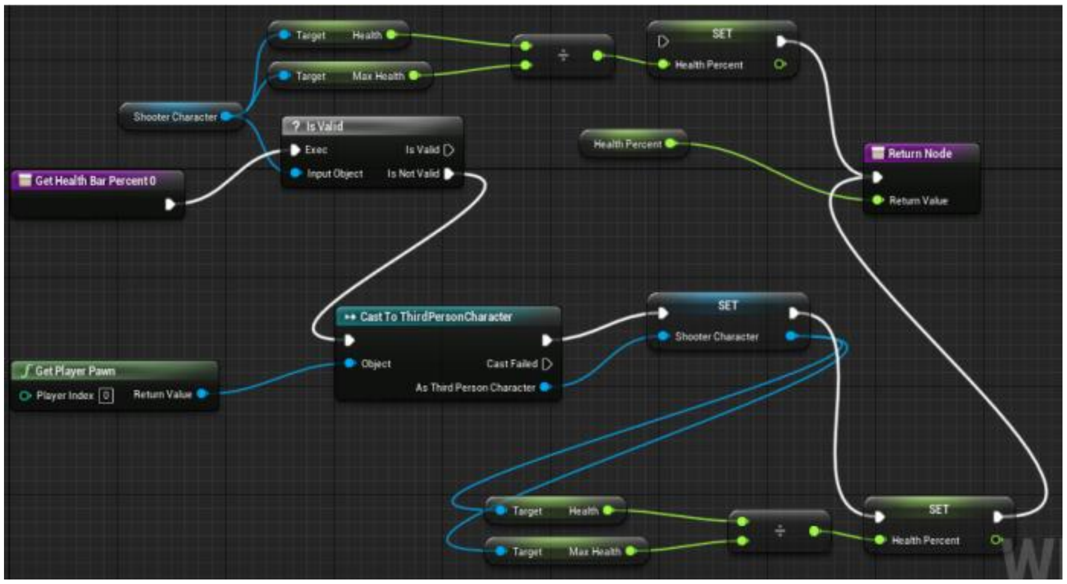
Enemy.h

```
private:

//Base damage for enemy
        UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess =
                "true"))
                float BaseDamage;


protected:

void DoDamage(class AShooterCharacter* Victim);
```

Enemy.cpp

```
BaseDamage(20.f)

void AEnemy::DoDamage(AShooterCharacter* Victim)
{
        if (Victim == nullptr) return;
        {
                UGameplayStatics::ApplyDamage(
                        Victim,
                        BaseDamage,
                        EnemyController,
                        this,
                        UDamageType::StaticClass()
                );
                if (Victim->GetMeleeImpactSound())
                {
                        UGameplayStatics::PlaySoundAtLocation(
                                this,
                                Victim->GetMeleeImpactSound(),
                                GetActorLocation());
                }
        }
}
```

Character health bar blueprint



Shooter animation montages

ShooterCharacter.h

```
//Montage for character death
       UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Combat, meta =
(AllowPrivateAccess =
              "true"))
              UAnimMontage* DeathMontage;

protected:

void Die();
       UFUNCTION(BlueprintCallable)
              void FinishDeath();
```

ShooterCharacter.cpp

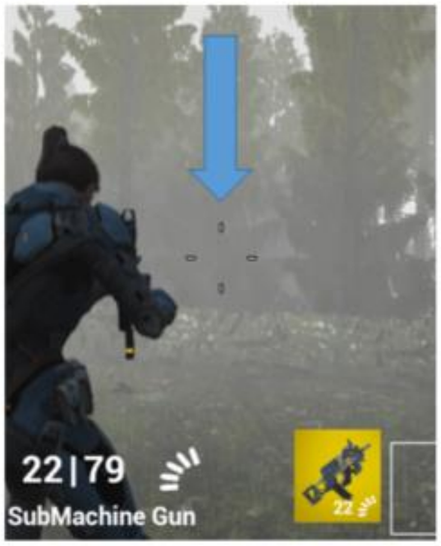
```
void AShooterCharacter::Die()
{
       UAnimInstance* AnimInstance = GetMesh()->GetAnimInstance();
       if (AnimInstance && DeathMontage)
       {
              AnimInstance->Montage_Play(DeathMontage);
       }
}
```

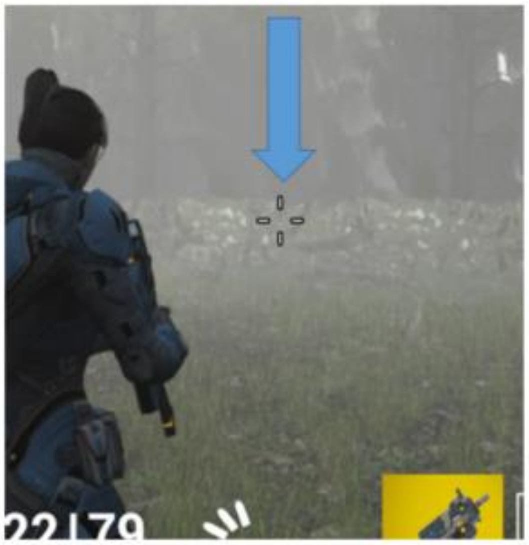
6. Testing

6.1 Crosshair spreading

Component tested	Purpose of test	Expected result	Tested by	Date tested
Crosshair spreading	To test how the crosshair is shown on the HUD and how it spreads when the character moves, crouches or aim.	Works as expected	Aaishane	22nd July 2022

Table 2: Component tested – Crosshair spreading

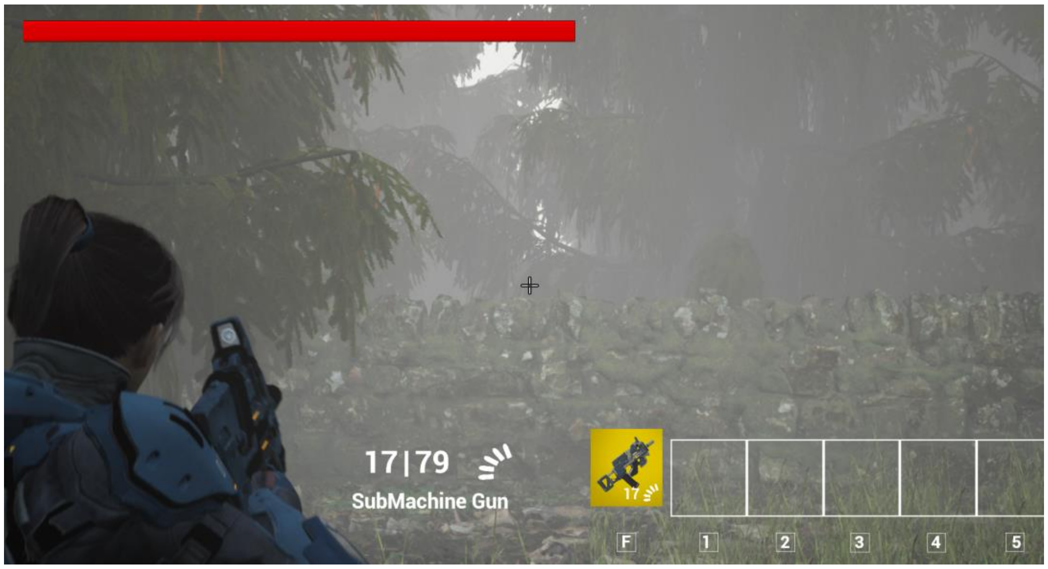




6.2 Crouching aiming

Component tested	Purpose of test	Expected result	Tested by	Date tested
Crouching aiming	To test if the character can crouch aim	Works as expected	Aaishane	24th July 2022

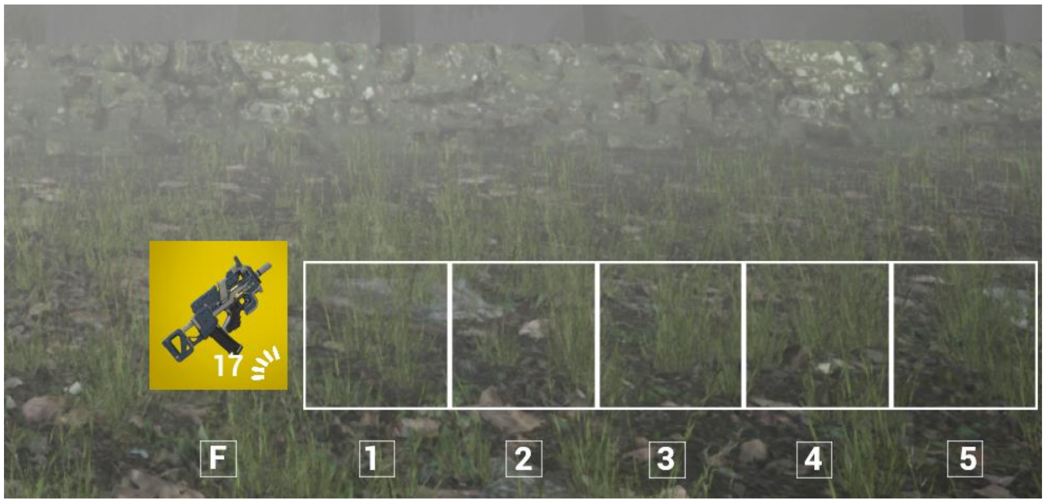
Table 3: Component tested – Crouching aiming



6.3 HUD Item inventory

Component tested	Purpose of test	Expected result	Tested by	Date tested
Item inventory		Works as expected	Aaishane	26th July 2022

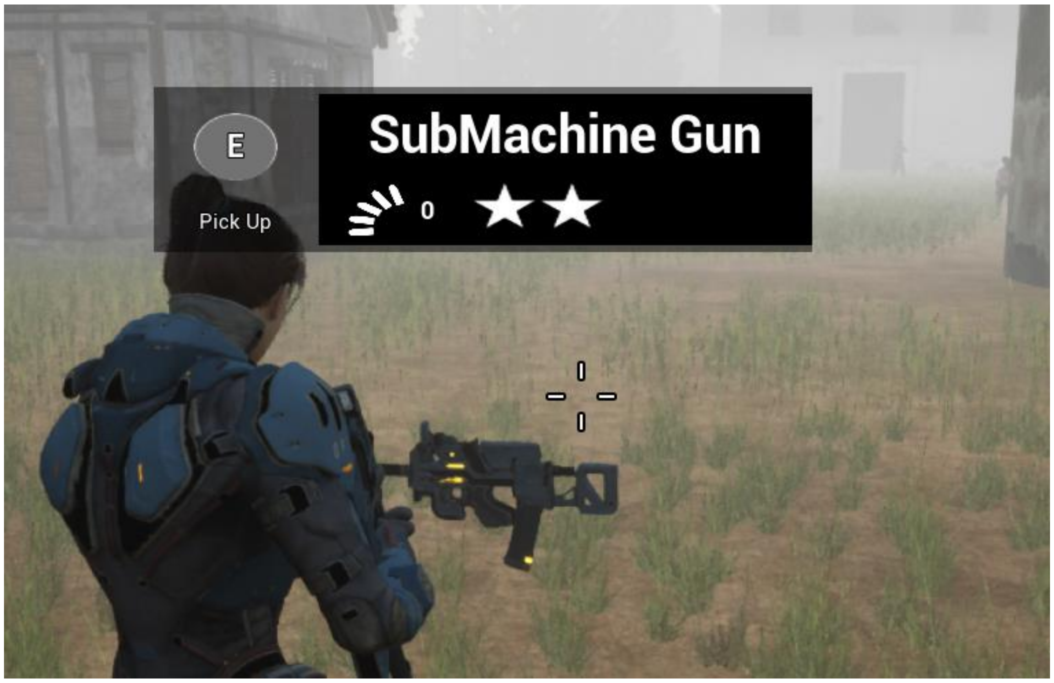
Table 4: Component tested – HUD Item inventory



6.4 Datable to change weapon rarity & damage

Component tested	Purpose of test	Expected result	Tested by	Date tested
Data table to change weapon rarity & damage	To test weapon rarity and damage	Works as expected	Aaishane	30th July 2022

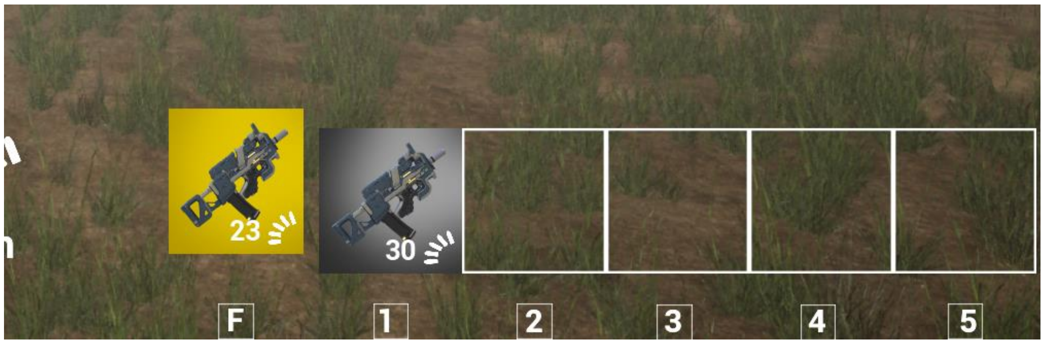
Table 5: Component tested – Datable to change weapon & damage



6.6 Weapon equip

Component tested	Purpose of test	Expected result	Tested by	Date tested
Weapon equip	To test if the character can equip the weapon he wants to use	Works as expected	Tavish	22nd July 2022

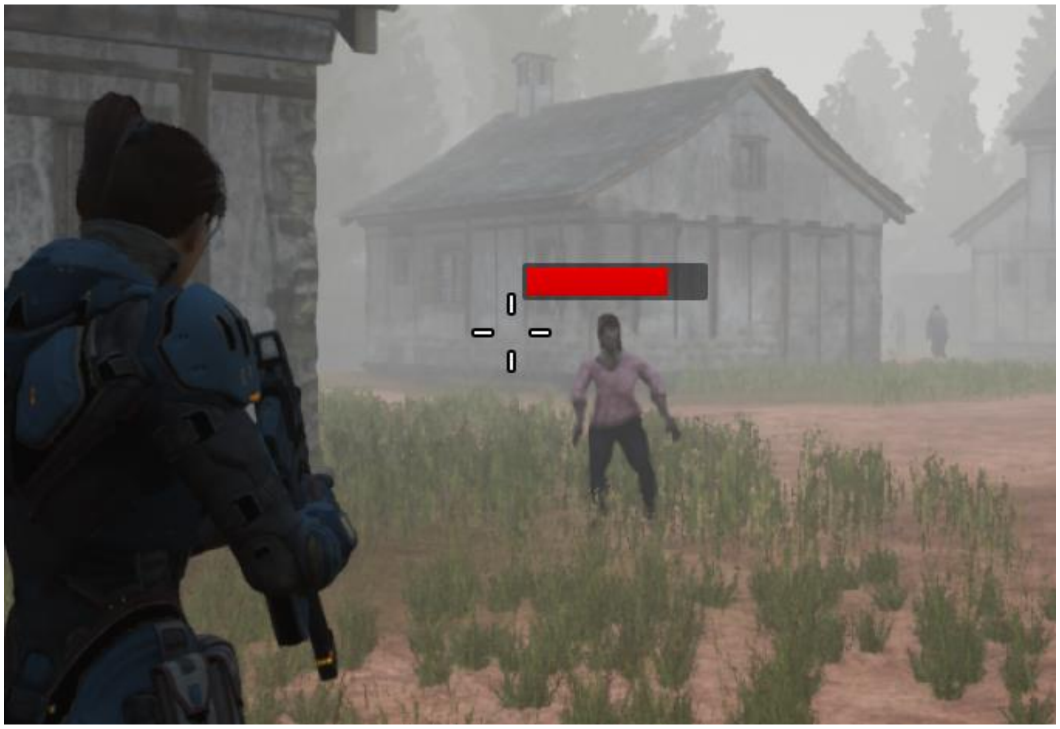
Table 6: Component tested – Weapon equip



6.7 Enemy movement

Component tested	Purpose of test	Expected result	Tested by	Date tested
Enemy movement	To test if the enemy is able to move	Works as expected	Ismaail	10th July 2022

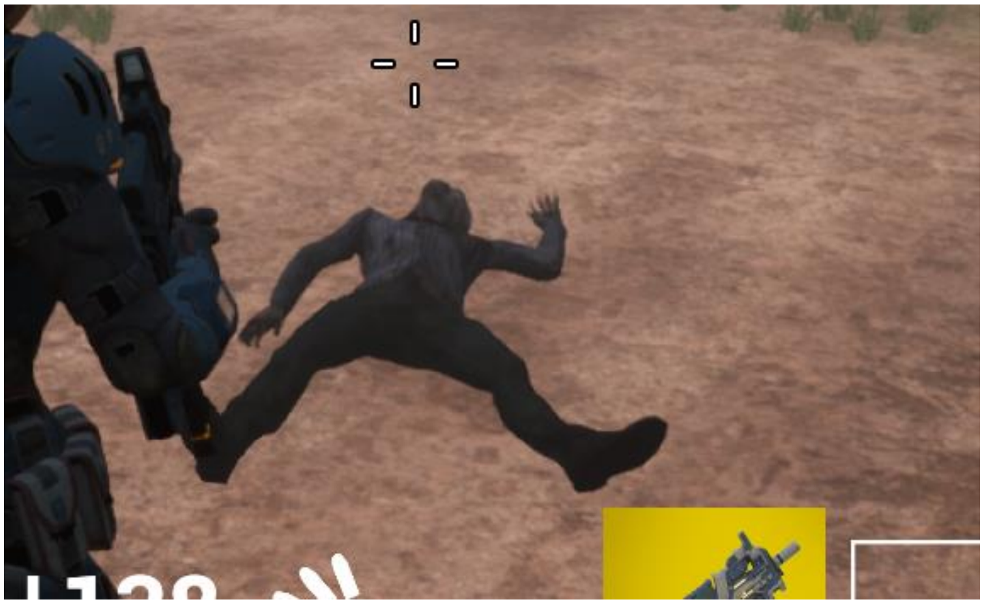
Table 7: Component tested – Enemy movement



6.8 Enemy attack and death

Component tested	Purpose of test	Expected result	Tested by	Date tested
Enemy attack and death	To test if the enemy is able to attack our character and die	Works as expected	Ismaail	15th July 2022

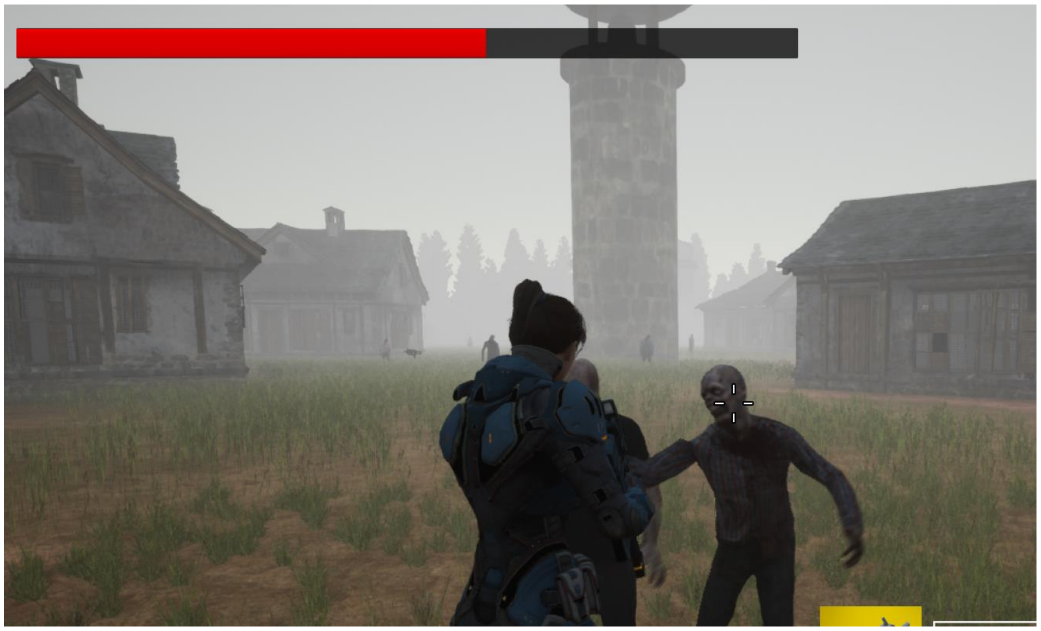
Table 8: Component tested – Enemy attack and death



6.9 Health HUD and damage

Component tested	Purpose of test	Expected result	Tested by	Date tested
Health damage	To test if health damage of character works	Works as expected	Ismaail	20th July 2022

Table 9: Component tested – Health HUD and damage



7. Conclusion

7.1 Achievements

7.1.1 Technical side

- Learned more about C++ coding skills
- Understood how to use public and private class
- Understood Object-Oriented Programming
- Learned how to use Unreal Engine
- Learned how to create a 3D game
- Learned how to put sound in a game
- Acquired knowledge of menu design and what elements to incorporate
- Developed the capacity to work concurrently on various tasks inside the same project
- Acquired the ability to think from both a developer's and a user's perspective
- Learned how to build and implement different modules together

7.9.2 Non-technical side

- Team members working together and communicating more effectively
- Acquired an understanding of the value of planning
- Acquired the skills to identify and address problem areas
- Improved our report-writing abilities
- Gained experience working in pressure-filled and unusual circumstances
- Greater time management abilities
- Have established a routine to write code that works correctly rather to just settle for working code
- Refined the skills of self-learning

7.2 Challenges and Problems Encountered

- The Unreal Engine Editor required a large of memory and some of us had to upgrade and completely change our laptops to a better performing one.
- Some zombies don't die.
- Endless time was needed to correct errors and bugs.
- Some functions did not function as expected and their functionality had to be changed.
- Our game is 30 GB and demand a lot of space
- A lot of time and money were invested.
- Some Unreal tutorials were complicated and hard to understand.
- Unreal Engine Editor is very complex to use.
- Editor crashed while compiling due to errors, back up data

8. References

8.1 References

Character Movement:

freeCodeCamp.org - Learn Unreal Engine (with C++) - Full Course for Beginners: https://www.youtube.com/watch?v=LsNW4FPHuZE [Last accessed July 1, 2022]

Devslopes - Unreal Engine Beginner C++ Tutorial: Building Your First Game: https://www.youtube.com/watch?v=1dl91ORwmy8 [Last accessed July 1, 2022]

Uisco - How To Make A Health Bar and Damage In Unreal Engine 4 https://www.youtube.com/watch?v=guHoFqJBLos [Last accessed July 1, 2022]

Uisco - How To Make A Main Menu In 9 Minutes Unreal Engine 4 Tutorial https://www.youtube.com/watch?v=RZlCqZhI9Nc [Last accessed July 1, 2022]

Awesome Tuts - Unreal Engine Character Tutorial - Animate And Move A 3D Character In Unreal Engine 4 https://youtu.be/DimZmTd5H44 [Last accessed July 1, 2022]

Unreal Sensei - Unreal Engine 4 Beginner Tutorial - UE4 Start Course. Available from:
https://www.youtube.com/watch?v=_a6kcSP8R1Y [Last accessed August 8, 2022]

Udemy 2022. Unreal Engine C++ The Ultimate Shooter Course [online]. Available from:
https://www.udemy.com/course/unreal-engine-the-ultimate-shooter-course/
[Accessed 1 July 2022]