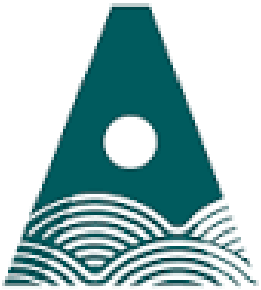

Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

GAME DEVELOPMENT

WITH

C++ and UNREAL ENGINE

B.Sc. (Hons) in Software Development

David Allen

2022

Supervised by: Gerrard Harrison

Department of Computer Science and Applied Physics

Atlantic Technological university (ATU)

Table of Contents

About This Project

Abstract

Gaming is a huge industry and with this, comes different types of software which is specifically designed to develop an array of different games. Unreal Engine 5 is new to the market, with an aim to create professional games by both professionals and amateurs.

The project is to create a basic game using software that I have not previously used before. I decided to use the Unreal Engine 5. The programming language used is C++, but I also have its own inbuilt Blueprint Visual Scripting that allows you to pull different functions and methods together without a deeper knowledge of C++. The combination of C++ and Blueprint Visual Scripting helps the user to unlock the full potential of the Unreal Engine software.

The Unreal Engine 5 is an incredibly diverse software package. On their own website, they describe the software as; 'The world's most open and advanced real-time 3D creation tool'. This means it can be used across multiple industries for a magnitude of uses, including, but not limited to, games, film, and television content creation.

The goal is to create a third person shooter, that incorporates player movement and shooting against enemy AI. Extra features such as health and damage will determine the winning or losing condition. We will examine the engine, interfaces and other essentials while developing a game and use C++ for coding.

The aim is to learn to use Unreal Engine, while developing my skills and understanding of C++. To create a game with all the knowledge I gained and to use the Unreal Engine to its full potential.

Author

The author of this project is David Allen. A 4th-year students studying for a Bachelor's of Science Honours Degree in Computing in Software Development in the GMIT Dublin Road campus.

Acknowledgements

The author would like to acknowledge Gerrard Harrison, my supervisor. I would like to thank Gerrard for all the time he put in, helping me with my project, for the tips and advice he gave throughout, for meeting me and for always keeping me on track.

1 Introduction

In October 1958 saw the release of a simple tennis game ‘Pong’, it is recorded as being the first Video Game development. Released by William Higginbotham, physicist it was the start of an evolution that is now in its 9th generation [1] And the evolution continues. Gaming in itself is now a multibillion-dollar industry, according to the gaming industry magazine, PC Games, it “will pass $200 billion in 2022”. [2]

If we take a look at one of the major gaming platforms: Steam, we can see that nearly 11 thousand games were released in 2021 (https://www.statista.com/statistics/552623/number-games-released-steam/) and solid growth expected in 2022. With this growth has come a significant investment in gaming software for development.

Epic Games, the company behind Unreal Engine has received more than 2 million in investment after they announced the launch of Unreal Engine 5. [3] Unreal Engine started off it demonstration of software by creating a shooter series known as Unreal. Unreal is a series of first-person shooter video games that powers the games, and is available for developers. [4] With games such as Fortnite, Street Fighter, Hellblade, Stalker and Mortal Online under Unreal Engine it is safe to say that it is one of the leading platforms to develop games on. As a person with a passion for gaming, when looking at environments to develop a game on, Unreal Engine piqued my interest and was a clear front runner for me, out of all of the gaming software available, and will be explored later in the review.

1.1 Idea

It is the aim of my thesis to introduce Unreal Engine in the context of a game developer starting out and understanding and examining the learning curve involved in the process of game development. We will examine the engine, interfaces and other essentials while developing a third person shooter game and use C++ for coding. C++ is an object-oriented programming language, known for its portability, clear structure and platform adaptability. Combined, we will create a simple gaming project from scratch and try to understand the process.

1.1.1 Game Engines

In its most simplistic explanation, a gaming engine is a developer's environment for creating games. [5] There are many components that feed into the engine and sit within its architecture e.g.

- artificial intelligence (AI)
- input
- rendering

- scripting
- collision detection

The above list is not exhaustive and a lot of the time, the components are programmed in the background without the developer's need for input. This leads to less time needed to develop games and is a win for the developers.

If you search Google for the top gaming engines in 2022 [6], Unreal Engine is returned as being the best in its field and one of the easiest for first-time game developers.

1.1.2 Technologies Used

- Unreal Engine 5.0
- Visual Studio
- GitHub

1.1.3 The Game

The game is to be a simple third-person shooter. The overall goal is to learn to use the Unreal Engine and C++ to develop a game, rather than focusing on the aesthetics and smaller details of the game.

The purpose is to make a fully working shooter game ending with you killing the enemy or losing as the enemy hunts you.

The game is set on a planet similar to our own. Players will have to navigate the character over mountainous scenes, around castle ruins and over hills trying to survive, while being hunted by alien soldiers. The soldiers are patrolling the planet searching for intruders to hunt and kill. The player has to kill or be killed, using only the weapon they start with.

The aim of the game is to survive, so if the player is spotted by one of the alien soldiers, the alien will start hunting them down. The player is able to outrun the soldiers also, taking into account their health and stamina levels. Stamina is rebuilt when the player rests, so if they are able to avoid being detected by the alien soldiers for long enough, they can avoid combat situations while their stamina builds back up, giving them a better opportunity of killing the soldiers when they are next spotted.

1.1.4 Objectives

Develop a game for an end user/player

Learn and embed a new programming language and its techniques - C++

Use a new software to create the gaming environment - Unreal Engine 5.0

1.1.5 Chapter Summaries

This paper is divided into chapters, each chapter is titled and describes an area of research relevant to the overall project. The table of contents list the chapters, their sub chapters and page numbers.

1.1.5.1 Introduction

In this chapter contains the context of the complete project, from the history of the topic through current day implementation. It examines the idea behind my project and the objectives I set myself for the project.

1.1.5.2 Methodology

In this chapter is about the choices that determined the outcome of the game. It describes the methodologies used: Agile and its use in the project.

1.1.5.3 Technology Review

In this chapter we take a deeper look at all technologies used with the development of the game. It will explain why whilst exploring the benefits and drawbacks of each such tech was used.

1.1.5.4 System Design

In this chapter: System Design, we will focus our attention on how the games was created using the technologies outlined in the Technology Review chapter and architecture of the project.

1.1.5.5 System Evaluation

In this chapter, I will discuss how I am testing and how I evaluated the project. System performance, my choices in prioritising features and real time problems that I came across.

1.1.5.6 Conclusion

In this chapter I will discuss how the aim and objectives of the project and how they were met whilst reviewing the overall outcome of the project.

1.1.6 GitHub Repository

The URL of my GitHub Repository is https://GitHub.com/allend4/Project.

The GitHub repo holds two main files that hold the entire project

Yr4Project

Config – contains configuration files which store the information related to the editor and the project.

Content – contains all the project's contents, such as assets to make the project. This is the folder mainly used as it will contain everything used such as meshes, blueprints.

Intermediate – contains temp files and folders required to work with the project.

Saved – contains autosaves, to log files and crash reports.

Source – contains the C++ files for the project. That’s includes .cpp source files and .h header files.

Documentation - contains any and all documentation related to the project.

How to run?

Everything was created within the Unreal Engine so it is relatively easy to run the game. The only thing needed is for the Unreal Engine 5.0 to be installed on your system. Within the Yr4Project is a

Yr4Project.uproject file. The project file is used to open the game in the unreal editor. Press plays for the game to simulate.

2 Methodology

There are various methodologies available to the software development process, waterfall, lean, Spiral to name but a few and the most commonly used at this time: Agile. When I started out developing the game, I did not have a full formed project plan. As I had not used Unreal Engine before and I had never developed anything in C++, I expected to come across some road blocks that may influence the functionality hence I needed to remain agile. With this in mind, I needed to choose a methodology that allow me to revisit the plan and make changes as needed. After researching the Waterfall, Lean and Spiral method I decided to go with the Agile Methodology as it allowed for a continuous iteration of development and testing. By using the Agile Methodology, I could remain flexible in my development while staying on track for development cycles. As I developed the project on my own, the need for some of the Agile features was not required. There was no need for a daily scrum meeting as I was in complete control of the development. Sprint cycles were also not needed as all project assets remained in my ownership.

2. 1 The Agile Methodology

The Agile manifesto [7] outlines the approach of Agile Software Development process.

The Agile methodology is a management style that allows projects to be broken up into smaller phases. There is continuous collaboration with all stakeholders and each stage ensures continues improvement. When the project starts, the team repeat the cycle of planning, executing and evaluation. This cycle is reiterated until the project is finished which allows for varying requirements to be delivered faster.

2.2 Using Agile

When you implement a Software Development Life cycle into a project, the expectation would be to have a team of developer's designers, engineers etc however as I was a team of one,, a lot of the traditional phases were not needed.

In a traditional project, the work would be divided into Sprints, with each Sprint taking somewhere between four to six weeks. This is normally outlined by the project manager at the start of the project when the planning meeting takes place and tasks are assigned based on the resources available and the scope of the project. These tasks will be prioritized and assigned to the team members. Each member will leave the meeting with a clear unserstanding of the projects bigger picture and the e

Plan: The sprint begins with a sprint planning meeting, where we had a meeting where we layed out components for the upcoming week of work. Once we were happy with this we then prioritised the work and assigned a task to each team member.

Develop: Design and develop what needs to be done to complete the task.

Test/QA: Complete thorough testing and documentation of results before delivering it to team members.

Deliver: Present the working product or software to team members and to our supervisor.

Assess: Gather Feedback from the team and from the supervisor and make changes accordingly.

Sprint Planning: The team and supervisor discuss the top priority user stories and determine which ones will be implemented in the next sprint

Sprint Backlog: Is the list of the user stories which are committed for the next sprint.

Sprint: This is a one to three week time frame where the user stories in the sprint backlog are developed by the team. During the sprint meetings its a team meeting where each team member discussed the completed work, the part each person is working on and if anyone needs help or has ran into issues in the work they were doing.

2.3 Technology Choices

2.3.1 Unreal Engine

Unreal Engine was created by Epic games in 1998 (Gen ?) and the original was used for first-person shooter (FPS) games. Starting out on a PC, it has evolved into a multiplatform, multi-user engine over the 24 years. According to the Times, “Unreal Engine is the second-most widely used video game engine, trailing only Unity, and is known for its depth of features and visual quality” [8].

For the small independent developer and for students like me one of the swaying points towards Unreal Engine is the fact that I can use it for free, as it is “free to download and use, with Epic taking a 5% cut on products created with it only after they earn over $1 million in gross revenue.”

Unreal Engine 5.0 recently released and promises to be a game changer in the world of game development. Some of the main aims was to learn a new language and use a software that I had never used before. Therefor the Unreal Engine ticked those boxes. A new game engine that I had never used before and implements using c++, meaning Id also have a language I wasn’t familiar with.

Unreal Engine also offers its own in-built blueprints, which is its own visual scripting system. One thing that makes Unreal Engine standout is it is developed in C++. Meaning compared to other mainstream platforms such as unity, which is developed in C#. Being programmed in C++ leads programs to perform faster due to C++ getting compiled directly into machine code, needing no intermediary translation. Meaning better speed and performance, which is importation for games and applications

2.3.2 C++

Industry standard language. Needing separate editors to program++ has been around for over 30 years, therefore many people can understand and there are lots of tolls and programs to assist in its use. One of C++ big advantages is its speed. It remains one of the gold standards for high

performance software and is used knowing it performs better than others. Being able to code in C++ gives you more options as you can make the game your own way, giving you access to everything in C++, compared to blueprints where it gives you limited functions.

C++ is one of the world's most popular programming languages. C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems. C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs. (W3schools)

According to the TIOBE (https://www.tiobe.com/tiobe-index/) Programming Community, C++ remains in the top 5 of the most used programming languages. The PYPL (https://pypl.GitHub.io/PYPL.html) Popularity of Programming Language Index is created by analysing how often language tutorials are searched on Google and C++ is also listed here in the top five.

One of the reasons for C++ remaining a popular programming language according to Schaffer (2022) [9] is because the language has many solid features and advantages, such as:

- **Exception handling**: Exception handling is built into C++. It's a tool that separates code that detects and handles exceptional circumstances that arise while running programs.
- **Function overloading**: Function overloading is the process of having two or more functions with the same name but with different parameters. This C++ feature allows you to define more than one definition for a function name or an operator in the same scope.
- **Memory management**: C++ supports dynamic memory allocation (DMA), which helps free up and allocate memory.
- **C++ standard library**: The C++ standard template library (STL) is filled with templates of ready-to-use libraries for various data structures, arithmetic operations, and algorithms.
- **Object-oriented**: Object-oriented programming (OOP) concepts allow you to treat data as objects and classes.
- **Multi-paradigm**: C++ is a multi-paradigm language. This allows you to choose a single approach or mix aspects of different programming paradigms (such as *generic*, *imperative*, and *object-oriented*).
- **Highly portable**: C++ is highly portable and is used for scripting systems apps which make up a significant portion of Windows, Linux, and Unix operating systems.
- **Versatile**: C++ is versatile and has a large job market.
- **Scalable**: C++ is great for resource-intensive applications because of its scalability and performance capabilities.

2.3.3 Blueprints

Blueprints is a visual programming language designed for Unreal Engine. It is tailor-made for Unreal Engine and is beginner-friendly in you don't need to be able to understand C++ to use it. However, although it can be handy for beginners to use, as they do not need knowledge of C++ in order to use it, it does limit what you are able to do with it, as there is only a certain number of options that are pre-created available. Therefore, having knowledge of C++ or developing your knowledge of C++ helps to widen the possibilities of use as you are able to call upon your own methods and functions created in C++. Some benefits I experienced were that there was no need to remember complex syntaxes or wording and it is easy to develop as you can right-click and search for a variety of functions which are readily available. This is helpful when using functions which are used often, as it saves time for the developer. It can be said from personal experience that anyone is able to understand

the visual representation of the code as it is and you are easily able to follow the functions as it is self-explanatory with the design and the way it laid out.

2.3.4 C++ vs blueprints

Blueprints is a method of stringing together node-based interfaces that represent events, functions, and controls. Unreal libraries contain limited events, functions and controls, which are prewritten using C++. Therefore, using a combination of both is the best of both worlds, as everything is coded in C++ still, but you have the benefits of quickness and ease by pulling from the blueprints to being able to customise by writing your own syntax. It gives you the convenience of using blueprints, without the limitations of using that alone.

2.3.5 Visual Studio

There are many different IDEs- integrated development environments- which contain a source code editor for typing in a textual programming language. Any of which would have been suitable for me to use. I decided to use visual studio as it is a powerful integrated development environment and is widely used by game developers. It is designed to smoothly integrate with Unreal Engine so I knew it would benefit me during this project. One of the advantages of using visual studios, is that you can benefit from the use of the in-built IntelliSense that can assist in your coding. IntelliSense is a code completion aid. This means features such as list members, parameter information, quick information and complete word allow you to learn more about the code you're using by keeping track of parameters, adding calls to properties and methods with minimal effort on the user's behalf. Another benefit of using visual studios is that it also works as a C++ debugger. A debugger is used by programmers to test and debug code under controlled conditions. This can help the user as C++ can be relatively complex, especially for a beginner.

3 Technology Review

3.1 Unreal Engine 5.0

Epics Unreal Engine 5.0 was released for production ready use on April 5 2022, supporting all existing systems. Unreal Engine 5 is viewed a s a game changer introducing various new features such as Lumen, Nanites and Megascans.

- lumen - Photo-Realistic dynamic Lighting.
- Nanite - Dynamic Level of Detail. Allows an insane level of detail in meshes that render instantly.
- Megascans - A Massive Asset Library – brixel bridge. Thousands of real-world assets converted into 3d objects for use.
- MetaHuman – framework that creates realistic human characters within minutes.

Unreal engine is new and full of innovation. It will allow the development of high-quality environments, with highly detailed models. Quality has been significantly raised all around. So, games can be created easier and quicker.

3.2 Visual Studio

It's a proven well-known editor. Easy to use and recommended by many developers as the ultimate IDE for C++ development. I had no issues with development alongside Unreal Engine. Unreal Engine C++ files automatically loaded within visual studio. C++ syntax highlights in various colours to make it easy to read and follow. Also the inbuilt C++ debugger helps to inspect the code.

3.3 GitHub

GitHub is a code hosting platform for version control and collaboration. It allows you to track and manage projects from anywhere. The code could be pulled and pushed as changes were made, with every update documented so that the change in code could be seen and explained.
Easy to use and allowed me to keep track of the progression of my project. Pre-built-in git ignore files for Unreal Engine stops files being committed that are not needed and would only waste space. My only issue with GitHub is that it does not like large files. I assumed the git ignore files would have managed this, but GitHub has a 100MB file limit. So, working with assets and trying to make the game realistic caused me problems due to their size. Not a big issue but it was inconvenient and made me make changes to the project such as using lower quality assets.

4 Game Design

When it comes to scripting, Unreal Engine offers two methods: C++ and Blueprint Visual Scripting to help create gameplay elements. The game is divided into two workstreams, game designer and programmer. . They each work in their chosen environments, normally Blueprint for the designer and notepad++ (or something similar) for the developer.

Both systems allow the creator access to the API and the framework classes, and they can work separately. The true power of UR5 is seen when both of these personas work together. The developer-building blocks in C++ and the designer implemented those blocks in their design.

The game was broken down into stages. As I go through each stage of the game, I'll be referring to different methods and functions. Everything mention is contained within Unreal Engines build in c++ library [10] and the Unreal Engine 5.0 documentation is the best place to go for all resources of learning to use Unreal Engine [11].

- Scene - Creating the 3D level in which the players character will be exploring.
- Character – set up and create characters
- movement – to enable a character to move based on the movement system.
- Camera – placing cameras to create a third person view.
- Animation - bring your character to life with built in animation tools.
- Shooting function – implement the mechanics for a shooting game.
- Health, Alive and Death system - set death after health hits zero.

- Enemy AI – create Ai behaviour
- Win/lose conditions – game mode created where you lose if your health hits zero but if you manage to defeat all enemies.
- Additional features – features expected to be put in game
- Improvements – to make better.

4.1 Scene

My initial aim was to create a forest effect. I started with an open plane and moulded the landscape, to sculpt mountains and rivers to make the landscape more interesting and appealing for the player to explore. However, this is where my initial problems started. As the level design was being created the aim was to upload each task completion to GitHub to document the project through stages.

Unreal Engine 5.0 has many new features, one of many is the inclusion of brixel bridge in the asset store. This allows access to mega scans which is a library of thousands of 3d and 2d assets based on real-world scans. These assets can be dragged and dropped directly into your scene for free. Trying to make the level look realistic I imported rocks and mountains. This leads to another feature of Unreal Engine 5 nanites. It is an amazing new feature, essentially it is a form of virtualized geometry, much like virtual texturing, although it can render scenes quicker and in higher quality.
With these two features I created an amazing scene to only realise that once uploaded to GitHub it created issues. Unknown to me GitHub has a maximum file size of 100mb. Therefore, any file that exceeds 100MB gets blocked. Instantly I realised there was an issue with file sizes. Although my aim was to create a realistic environment, I was also limited by the GitHub threshold. Unfortunately it was back to the thinking board for me. I had to rethink what I wanted my landscape to look like, understanding that I was now limited to which assets I could use, as it would impact my file sizes which would in turn, impact my ability to upload my files and my progress to GitHub. Through trial and error, the assets could not exceed medium settings to meet GitHub's 100MB constraint. Finally, I understood my limitations and could therefore move on and continue level development. Create a directional light that simulate the light being emitted from the sun. Also add ***SkyAtmoshpher*** which will create the atmosphere in the sky. Both together you can see the sun in the sky and the lighting becomes dynamic as shadow are based on the light direction in the sky as it moves.

Landscape and sky had now been set. I was able to create mountains and hills, and even a river. From a distance, the land looked like a mountainous region similar to the one you would find on Earth. But up close the land looked barren and empty. I decided to create a field of trees moving in the wind to give some texture to the environment and to make it look a bit more interesting for the player.

However, again found that I'd hit GitHub size limits once more. The forest was deleted and the game design went back into development. At this stage over 2 weeks had passed and instead of a detailed level, I had a barren and basic looking level. Too much time had been wasted on aesthetics that I had to move on, so the process was restricted. Now the main goal was to deal with the functionality of the game. There was no point in having a pretty game that didn't work. It was decided to have a game function correctly and could be played, this way progress could be made with the project and the aesthetics could be updated later if the opportunity arose.

4.2 Character

I needed to create a character next. This leads to the built-in options between a pawn and a character. A pawn is anything you can possess and be controlled by the player. Although, the character class is a pawn that inherits all from the pawn class but is a specialised class, in that it has character-like abilities built in such as enabling the character to walk.

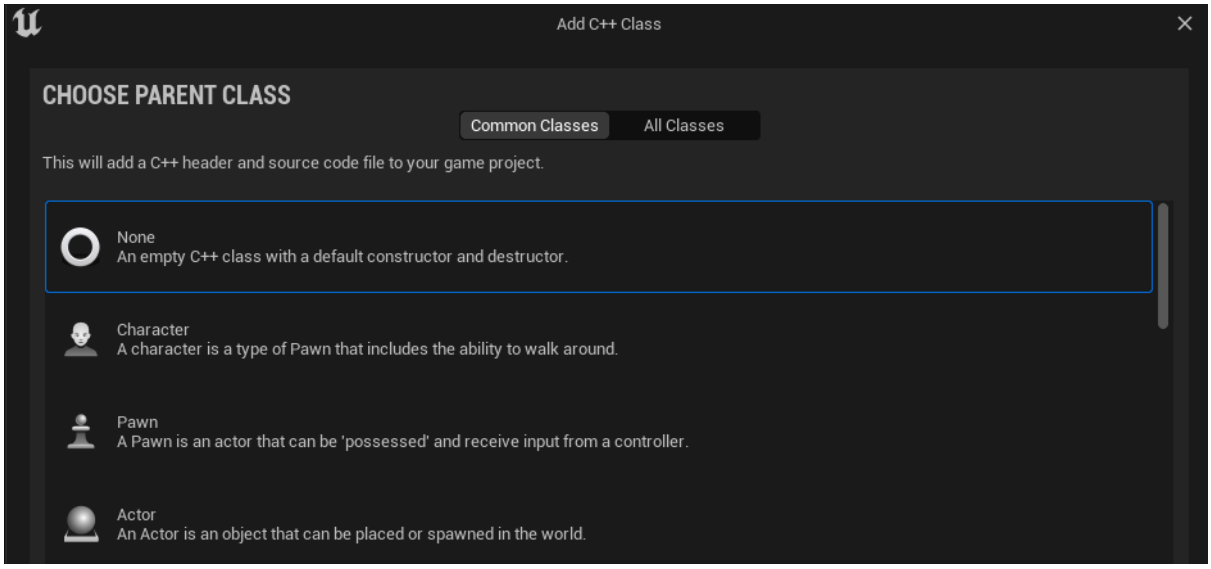

Figure 4.2.1 – Add C++ class

To do this you; create a new C++ class, character class Character. Builds a .h header file and .cpp C++ file. Header files are used for publicly declaring instance methods and variables. While the C++ file is used to implement those instances and methods.

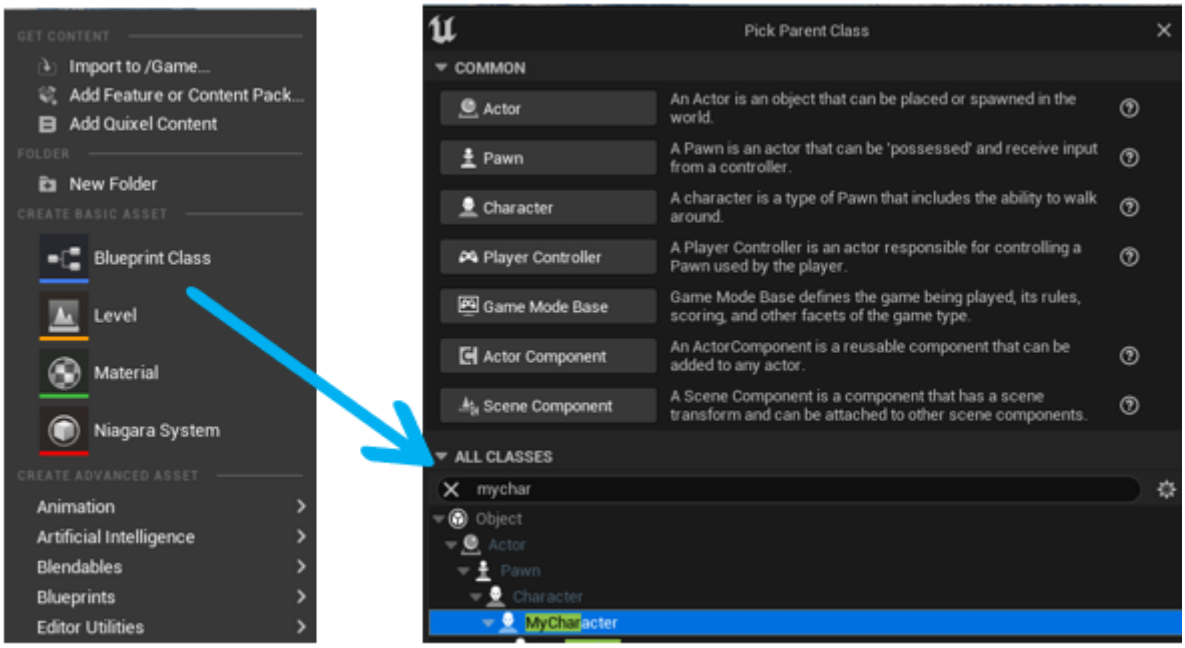

Figure 4.2.2 – blueprint parent class

Create a blueprint that inherits off the already created MyCharacter class. Open the blueprint of my character and the components available are Capsule Component, Arrow Component and Mesh. Capsule component which does rough physics, manipulation and collision with the terrain. Arrow component that suggests which way is forward for the character and finally the mesh, which is empty but awaits a mesh to be added. I my project I have a s soldier asset that I used in the skeleton mesh.

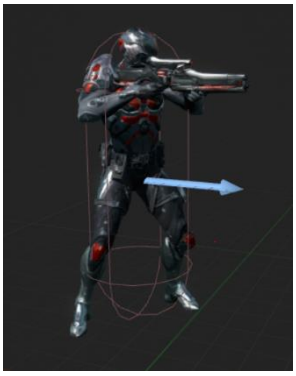

Figure 4.2.3 – MyCharacter Componenets

Adjustments are made to fit within the capsule and rotations, so the character is facing forward with the arrow. Also under components is Character Movement, this is responsible for giving the mesh character like movement, being able to walk, run, jump and more. Under details panel contains all the movement details and settings. That is why it is beneficial to use a character over a pawn, so that you don’t have to create all the movement code from scratch.

4.3 Movement

Unreal Engine is optimized to create games and aims to simplify. These are all already built into the movement script.

- ***AddMovementInput*** to move left to right, forward and back.
- ***AddControllerPitchInput*** to look up and down
- ***AddControllerYawInput*** to look left and right
- ***Jump*** to enable jump.

Start with creating a binding through Project Settings, Engine, and Input. Allows you to map keys by creating action mappings and axis mappings. Action mappings represent a button or key press that calls a method based on the key. While axis mapping allows for inputs that have a continuous range. Works well with controller triggers. For the purposes of this game being playable on pc online. I focused on axis mapping.

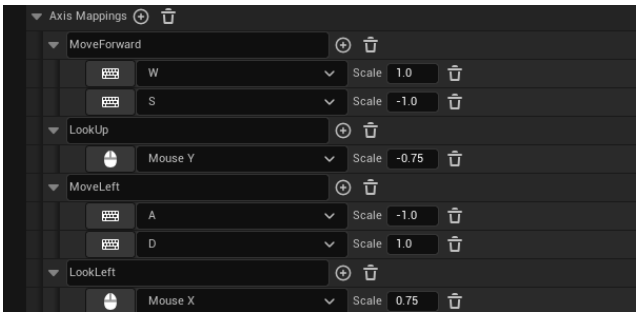

Figure 4.3.1 Axis mapping

Within action mapping can add a new mapping called forward. Bind this mapping to the keyboard letter w. there is no need to add another mapping for moving backward. As the mapping move in a positive and negative integer. Meaning you call also bind the keyboard letter S to forward mapping but assign it a negative value to move backwards.

```
// axis
PlayerInputComponent->BindAxis(TEXT("MoveForward"), this, &AMyCharacter::MoveForward);
PlayerInputComponent->BindAxis(TEXT("LookUp"), this, &APawn::AddControllerPitchInput); // pitch
PlayerInputComponent->BindAxis(TEXT("MoveLeft"), this, &AMyCharacter::MoveLeft);
PlayerInputComponent->BindAxis(TEXT("LookLeft"), this, &APawn::AddControllerYawInput); // yaw
// action
PlayerInputComponent->BindAction(TEXT("Jump"), EInputEvent::IE_Pressed, this, &ACharacter::Jump);
PlayerInputComponent->BindAction(TEXT("Shoot"), EInputEvent::IE_Pressed, this, &AMyCharacter::Shoot);
```

Figure 4.3.2 binding keys

Open the ***MyCharcater*** under C++ classes to load up visual studios, which is an IDE that allows you to edit code in C++. Bind the action to the player input. Calling the mapping named forward. Calling the function ***this*** the current target and wanting the ***MyCharacter::Moveforward*** to be called.
In ***MyCharacter*** heap file create a new function called ***MoveForward***. Create that implementation within the cpp file that calls the ***AddMovementInput***, which will get the forward vector the actor and multiply by the axis value. So, if it's one we move forward, but if it's -1 we move backwards as it becomes a minus vector with this single function. Following the same steps, you can create the move right function that moves the character left to right based on the 1/-1 vector. And similarly, look up and look right, although of instead of using the ***AddmovementInput***, you would work with ***AddControllerPitchInput*** and ***AddControllerYawInput*** for looking up, down, left, and right.
with this basic function linked to their key binding, the character is now able to move and look around the map freely.

4.4 Animation

using prebuilt animation sequences from asset packs. You can view all animations available for your skeleton. The skeletal mesh is the skin or surface model of the character. And the skeleton is the bones in the character that ties together the meshes and the animation. A single skeleton can have multiple meshes or skins, which each have multiple animations. All built on top of the skeleton.

The animation blueprint asks for a suggestion skeleton. Like the meshes and animation, the blueprint is linked to the skeleton. This was you can use all the animations available for the skeleton with all meshes available for that skeleton.

The animation blueprint editor contains a section called AnimGraph. AnimGraph short for animation graph is where you can create animation specific logic for your character.

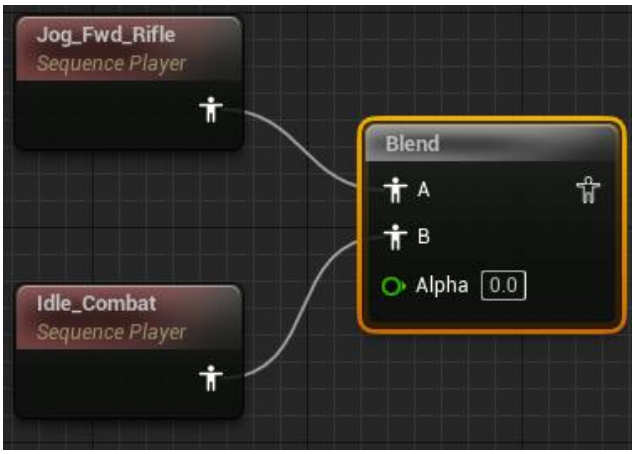

Figure 4.4.1 Blend animations

With the blend node, you can change between animations. By changing the alpha, it determines the animation to play.

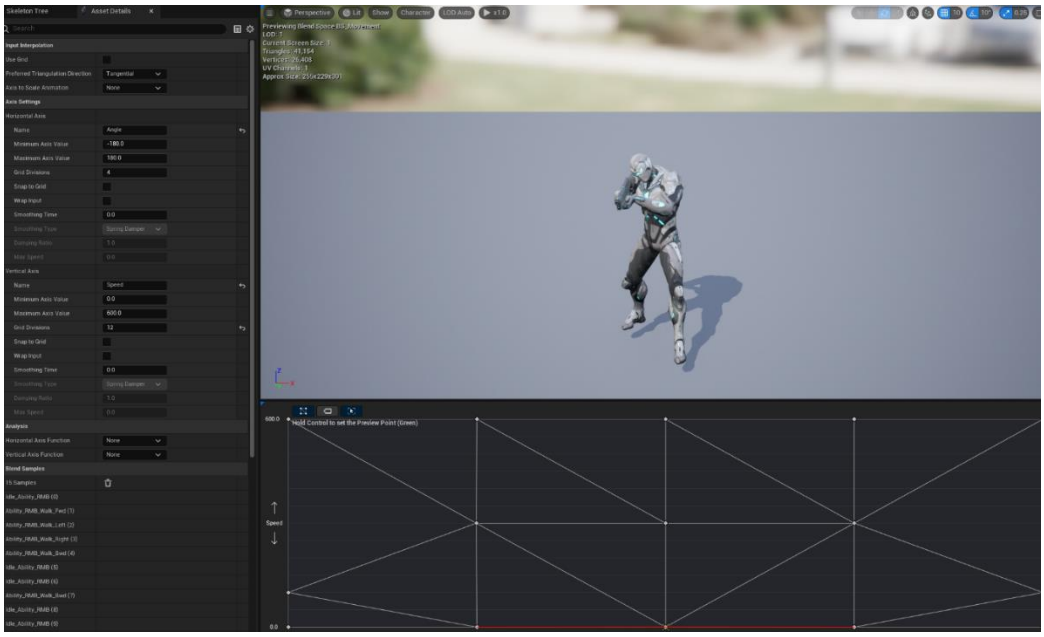

Figure 4.4.2 Blend space

Blend space as seen above allows you to create transitions between different animations and smoothly move between them. Create a 2D blend space based on your chosen skeleton called ***BS_movement***. Two dimension due to the forward/backwards dimension and the left/right dimension.

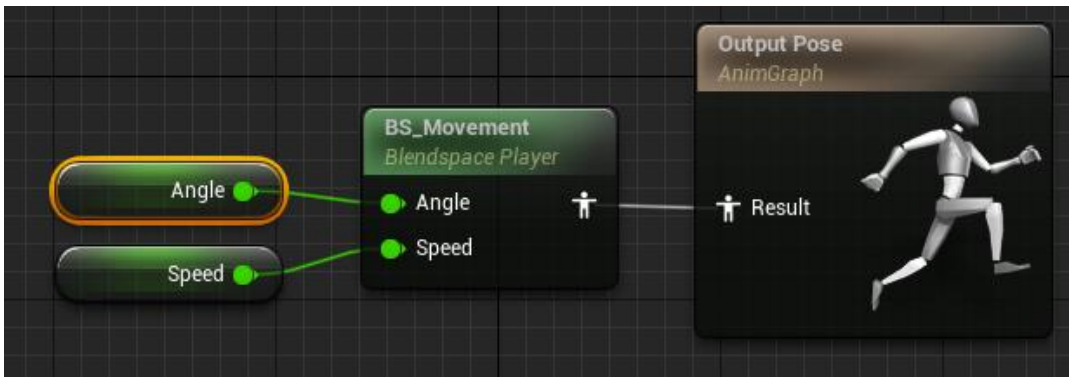

Figure 4.4.2 Animgraph

Inside your animation blueprint ***ABP_Mycharacter***, open the AnimGraph and add in our recently created movement. You can create variables speed and angle. Speed to adjust the speed of the character between walking and jogging, to the angle of the movement. Compile and save.

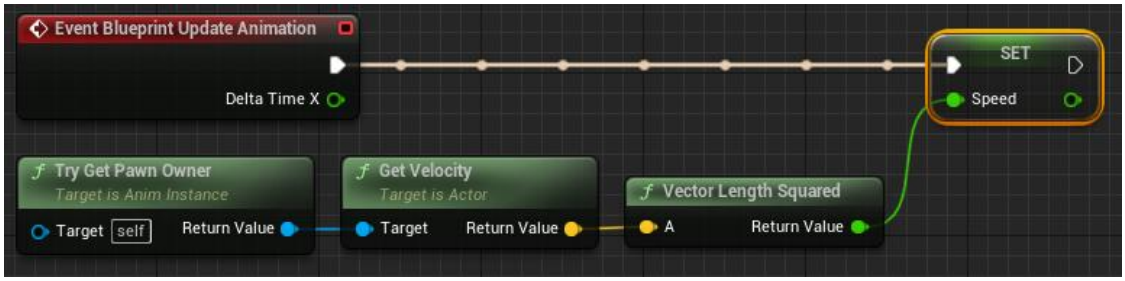

Figure 4.4.2 BluePrint animation

Get velocity of the pawn, use the vector length to set the speed, while calling it from the blueprint animation.

4.5 Camera

Things like the camera are a reason why blueprints are so convenient. No need to hard code it. You can just open the ***MyCharacter*** Blueprint. Add the component Spring arm and camera, making the camera a child of the spring arm. Adjust the spring arm to the character model depending on where you want the camera placed. The reason the spring arm is used is to automatically control how the camera handles situation where it becomes obstructed, such as backing your character into a wall. This will always keep the character in view with no obstruction.

4.6 Shooting function actor

You want the gun to shoot. You create a new C++ Actor Mygun as it will be an object placed in the world. Open Mygun.h and create a pull trigger method and create the implemented function.

Within the Mycharacter.h create a new action Shoot () and implement the function. In this function, you get the gun to call the trigger. As previously instructed the key binding will need to be created within the project settings and input so that when the action mapping left click mouse button is pressed it will call the shoot function.

4.7 Health, Alive and Death system

For damage to be dealt, there needs to be something that can be damaged. From the content browser clip and drag in ***BP_MyCharacter***. Once the character mesh loads it will automatically inherit the AI Controller.

If you load the game, you can shoot the enemy although no damage is taken. So, damage needs to be introduced and is a two-stage process. To send and receive the damage. These are the two main types of damage that exist in unreal:

FPointDamageEvent : damage subclass that handles damage with a single impact and source direction. As stated in the title point damage from a single point. So fits the description more of a gun.

FRadialDamageParams : parameter used to compute radial damage. More for wide spread damage such as grenades.

Within the gun.cpp create ***FpointDamageEvent DamageEvent*** . Under ***Gun..h*** create another ***Uproperty***. ***UProperty*** variables are declared using C++ with additional descriptors. Declaring the float damage of 20.

Apply the hit code to the Actor. Checking for a null pointer. If the actor is not null than enemy can take damage. Now damage can be dealt but the damage is not recognised,

Open ***Mycharacter.h*** and declare the ***TakeDamage*** method and implement. The implementation opens in ***MyCharacter.cpp***

Use ***DamageApplied*** to decrease the health by the damage applied. Use ***Fmath.min*** to take the damage applied off the health. And return the ***DamageApplied***. This means once the health hits its minimum of zero, it will not go negative.

Now we have the damage dealt and the damage applied. Once an enemy is hot the health reduces until it hits zero. But nothing happens. We want the character to die when its health reaches zero. Therefore, we need an animation to indicate the indicatethead. Open n ABP_MyCharacter. Within animgraph create a new Boolean, blend poses by bool. Play death forward animation to connect to the blend poses by bool node. If positive it calls the death animation, but if it's false it does not.

Create a new IsDead variable Boolean within the variables section and connect it up to the active value. If it determines the character is dead it will play the death scene otherwise nothing will happen.

4.8 Enemy AI

Create new C++ class, search all classes and find AIController. Once MyAIController is created it will generate and create the needed c++ files.
Now create a Blueprint that is a child class of the MyAIController. Open the blueprint of myCharcacter and under details, pawn, you want to change the Ai controller class to your newly created BP_MyAIController. Compile and save.

Now that BP_MyAiCOntrolller is showing up in the game it means any change or modifications to this file will also show up.

Open MyAIController within visual studios. In the heap file create protected: virtual void BeginPlay();

Protected - accessible by friends of their class, and in the case of protected members, by friends of their derived classes.

Virtual – used to modify a method and allow for it to be overridden in a derived class.

override - redefine a function.

Create the implementation in the C++ file MyAIController.cpp .

Super:Beginplay()

Super - It is used inside a sub-class method definition to call a method defined in the super class

Can use an instance from GameplayStatistics, which needs to be called for use
#include “kismet/GameplayStatistics.h”

UGameplaystatics:: calls getplayerpawn function which returns a pointer to a player pawn by taking in a worldContect Object and playerindex.
apawn* playerpawn = Ugameplasystatics::GetplayerPawn(getworld(), 0);

Now use the setfocus(playerpawn); which will cause the enemy to focus on the player. Instantly you will notice in game that the enemy follows the character pawn.

windows – placed actors. Search for Nav mesh bound volume. This will create a box that everything inside will be checked how navigable it is. Make it visible so that you can see the boundary. This boundary will cut out unnavigable paths such as walls etc leaving only a path that the enemy can follow.

But you need to link the AI to the boundary. Open BP_MyAiController. Under componentes there is a path following component that iOS is responsible for creating paths and allowing you to follow them automatically. The function does not need to be built in as it's already there.

Back in MyAiController,cpp, add the function movetoactor(playerpawn, 200). So, it will move to the character pawn within a distance of 200.

Load the game and if you set foot with the nav bounded area, the enemy will move to character location. Although once there will stay there.as once they reach you, they give up on chasing you.

So to add to this you can create a tick function that will call on every frame of the actor. Create a public: virtual world tick (float deltaSeconds); and implement it. Within the function put Apawn and move to actor methods, compile and play. Now the enemy checks every frame if you are nearby. If you are it will constantly update to follow you. Although if you leave the boundary, it does nothing but stand still.

Line of sight.
Within the tick function, we will want to check the line of sight. If a character is in line of sight, set focus to character and move to. Otherwise, clear focus and stop chasing.

Behaviour trees and blackboards.

At the moment the AI is very basic. To make the AI more complicated we need to move onto behaviour trees and blackboard. So, to create some sophisticated AI behaviours is using behaviour trees. Right-click and under artificial intelligence create a behaviour tree BT_EnemyAI and also a blackboard BB_enemyAI. Allows the creation of structured behaviour as trees.

Blackboard is the memory of the AI, where you set the properties of the game. And the behaviour tree uses those properties and uses them to decide what it's going to do.

Within the MyAicontroller need to reference the behaviour tree in the blueprint so it knows what behaviour tree to run.

Within the BP_myAIController under details select the AIbehaviour to your behaviour tree.

4.9 Win/lose conditions

Need to create a state where you have won or lost. Open StartGameMode.h. create pawnKILLED(APawn* Pawmkilled). As usualu create a basic implementation to open in StartGameMode.cpp.

Create new class KillThemAllGamemode that inherits from StartGameMode,

4.10 Additional features

As the development was focused on functionality. The aim was to fit in as many features as possible instead of working on visual effects. Here are some of the addition features worked on in the project and their stage of completion.

Mini map (Completed): the mini map was relatively easy to complete. It revolved around creating another camera that viewed the character from above, looking down. This was streamed to a new window and appears as if a mini map is on screen. Completed.

Blood splatter (Not Complete): Blood splatter was a work in progress. The aim was that the character would recognise when they received damage. A separate UI screen was created with blood splatter that would flash up on screen every time the character received damage. The Ui screen would only appear for a fraction of a second before fading away. The issue I came across is that I could have it on constantly or not at all. Needed time to work on it into the damage taken code. Ran out of time.

Character Selection (Not Complete): Character selection was a work in progress. The purpose was to load a UI screen that's showed you the various skins or meshes available for your selection. Once chosen the game would save your choice and load in the correct mesh to the character. Was taking too long to implement and was put on the back burner for a future update.

Sound (complete): whether it was adding shots to the gun or a backing track to the game. It was easy enough to complete. Search with actors for Ambient Sound and drag it into the world. You can select a sound clip so that is plays music in the background. Simple.

Main Menu / Pause Menu (Complete): No game would be complete without a menu. Create a basic UI with Buttons. Each button can have an on clicked event. Therefor the quit button quits the game, while the play button removes the UI widget while adding the character and level to the viewport.

Setting Menu (Complete): Create a Ui with buttons and make each button have a clicked event. The three buttons min, medium and max are for the computer setting. Meaning you can pre configure the setting for the game. On clicked event executes console commands that are obtained [12] within the unreal documentation. Post processing effects that define the overall look and feel of the scene.

4.11 Improvements

With time being wasted in the beginning on attempting to make the game look realistic, as well as Problems with GitHub max file limitations causing assets to be scrapped, a more basic scene was created and the updated aim was to focus on functionality, as so much time had been wasted creating an environment that couldn't be used.

Improvements were to be made once the game was fully functional. I was hoping to go back and make various improvements such as improving gameplay visuals, from making the environment look more interesting and realistic, to adding additional features such as a blood splat screen that detects when you receive damage, a select player screen where you are able to select the skin/mesh you wanted your character to play as, to health bars above the enemy player so you are able to monitor their health status while in combat. I also wanted to update the menus and logos.

Due to delays within the project and time constraints, these tasks and features were never able to be completed. This could therefore be used for future updated releases of the game, which would improve the gameplay experience for the player.

5 System Evaluation

5.1 Testing

The main type of testing that occurred with the game was in editor testing (play and simulate). Play in editor allows you to play the current level directly from the editor so that you can test gameplay functionality, including player controls and events triggered by the player in a controlled environment.

The game would first have to be saved and built. Visual studio would build the game before it would play. If they game would not compile you could view the debugger to see what issues were highlighted within the code.

Using editor testing, every time something was added to the game it could be loaded and trailed. It allowed instant tweaks and would show how the game changed as a result. This type of testing was constant trial and error. Every tweak made to the game would be re-compiled, built, and simulated. Any issues that occurred could be instantly fixed. It was a very convenient method that allowed testing of everything to see what occurred.

Additionally, I ran manual test plans intended to check the functionality developed. Through requirements and test cases you can determine the parameters needed. With a series of checks you compare the expected result with the actual result to see if it passes or fails. If the test passes, you can move onto the next test case, but if it fails it gives you time to correct the expected response.

The test cases require thought to see if actions performed will satisfy the software requirements. Each step can be as detailed as I want, meaning I could test exactly what I wanted to see if I got the desired result.

5.2 Limitations and issues

5.2.1 GitHub

As the game was being developed there was a couple of limitations/issues that were found. The initial issue was not noticed for some time. I spent time creating a realistic forest sandbox level. Although there was no issue with the game itself, the problems occurred using GitHub. GitHub has a maximum upload file size of 100MB. I never realised the issue until after the level was completed and I attempted to push it all to GitHub. Once I did this, non-stop errors occurred related to max size limits. Once I knew what the issue was, the forest was deleted, and all the mountains and hills had to be redone with lower-quality assets. Through trial and error, I could manually view the size of each asset to determine if it could be used. Everything had to be replaced and I ended up with a basic scene with no forest. Over two weeks had been wasted and it was decided that I should focus on the functionality of the game rather than the aesthetics. Having a level that looked good, but you could not do anything on was not a game. Where a game with good functionality but poor visuals could always be updated at a later date. Therefore, the project changed focus and functionality won over visuals.

5.2.2 animations

Another problem was the animations. Unreal Engine 5 provides tools that enable you to manage imported characters, skeletons, and animations, but the animations are pre-built into your asset. For example, if you download a character asset, it comes with various meshes and animations. But it is not easy to create your own animation. As an example, I used the wraith skeleton asset. It came with

multiple animations available such as walking, running, and jumping. I wanted to incorporate crouching into my game. Setting up this feature is easy enough until I realised there was no animation included to go along with it. So, my character was able to crouch and stand, but there was no transition between the meshes. No animation to match the movement. I found it particularly annoying, as I had created a feature, but it would be unusable.

You can import other animations from other meshes. I thought this could be a reasonable solution. Although every animation is built for a particular mesh. When I found and imported a crouch animation, I could place it on my skeletal mesh, and it did work but not flawlessly. As it wasn't made for my particular mesh it resulted in stretching and an unusual movement made with the character.

5.3.3 Glitches

Unreal Engine 5.0 is brand new and with any software, glitches and errors can occur. This issue occurred frequently and is known as an issue with the unreal editor, waiting to be patched. Although there is a simple workaround.
The issue is from time to time as I'm editing the code and adding new features, the game wont compile. It states issues such as 'child of blueprint does not exist'. So, it somehow does not pick up the inheritance within files, even though you can physically see them, and they do exist in the project.

I ended up having to Googled [14]the issue to see that it's a common issue that occurred in UE4 and has been carried over to UE5. The easy fix is to go to the project files and delete the content within the Binaries and Intermediate folder. When you attempt to open the project again it will rebuild from scratch and all the inheritance is recognised once again. It's a silly little bug that occurred quite often. It was quite inconvenient and took up a lot of time but luckily there was a simple solution that fixed the error every time.

6 Conclusion

The Unreal Engine was a great engine to learn. With the release of Unreal Engine 5 its makes things easier to create and I can see why it is so successful and is one of the main game engines out there.

6.1 Outcomes

Develop a game

Learn a new language C++

Use a new software Unreal Engine 5.0

Create a fully functional game

6.2 Retrospective of this project

The purpose of the dissertation was to develop a game using an unfamiliar method, which was achieved. Unreal Engine is a major player in the game engine market and even with little C++ knowledge, t allows you to build fantastic games. Overall, the creation of the game went well. I created a third person shooter where you were being hunted by the enemy. Using AI, the enemy characters would hunt you down. It was kill or be killed. The game was not overly complicated, and I

was happy with all the little extra features built into the game such as the mini map and changing between third person and first person.

Visuals were an issue and were pushed aside to focus on game functionality. In reality, this was a wise choice and enabled a working game to be created. Although I do feel aesthetical it could look better and level design could be a lot more complicated. The main issues I had with making this game, were due to having to also use GitHub to upload my work and show my progress. If I hadn't had to use GitHub, these problems probably would not have come up, and I feel as though my game would have been a lot more detailed, especially aesthetically, as I would not have had to worry about file size issues and could have created the landscape I had pictured when developing my game initially.

Using Unreal Engine was a learning curve and looking back now, there would have better ways of coding the game, ways of making characters more complex, and landscapes more interesting. However, my aim was to gain a better understanding of C++, which I definitely do now. I did rely partly on the visual blueprints but with a better knowledge of C++, I could add in more features and make them run smoother if I were to make the game again or make any future updates to the game. C++ is similar to other languages; however, the syntax is slightly different. I feel after making this game, and by using Unreal Engine, that I have a good knowledge of C++ and I can understand now why it is still a very popular coding language, despite being over 35 years old. As the saying goes, 'if it isn't broke, don't fix it!' if the coding language works and works well by many people all over the world, why change it or use something different?

6.3 Improvements

Practice makes perfect. During my 4 years of studying software development, I have used many different coding languages and programs, throughout a variety of different assignments. However, this was my first-time using C++ and the Unreal Engine. I learned a lot about the language and how to apply it. As with any new skill, I feel I could only improve on what I created. With a better understanding of the process, language used and program, I feel there's much room for improvement in all aspects of the game.

7 Bibliography

[1] *Home video game console generations.*
https://en.wikipedia.org/wiki/Home_video_game_console_generations.
(Assessed on 30/07/2021)

[2] *Game industry revenue 2022.*
https://www.pcgamesn.com/games-industry-revenue-2022.
(Assessed on 01/08/2021)

[3] *Epic Games investment.*
https://uk.pcmag.com/games/139737/sony-lego-group-each-invest-1-billion-in-epic-games
(Assessed on 01/08/2021)

[4] *Unreal Wiki.*
https://en.wikipedia.org/wiki/Unreal_(video_game_series)
(Assessed on 02/08/2021)

[5] *What is a game engine.*
https://fullscale.io/blog/what-is-game-engine/
(Assessed on 04/08/2021)

[6] *Best gaming Engines.*
https://www.incredibuild.com/blog/top-7-gaming-engines-you-should-consider
(Assessed on 05/08/2021)

[7] *Agile Manifesto and Agile Principles.*
(Agile Manifesto and Agile Principles, http://agilemanifesto.org
(Assessed on 07/08/2021)

[8] *Unreal Engine 5 launch.*
https://time.com/6164332/epic-unreal-engine-5-launch/
(Assessed on 05/08/2021)

[9] *Is C++ still a good language to learn for 2022.*
https://www.educative.io/blog/learn-cpp-for-2022 /
(Assessed on 07/08/2021)

[10] *Unreal Community.*
https://unrealcommunity.wiki/unreal-cpp-d702003t
(Assessed on 10/08/2021)

[11] *Unreal documentation.*
https://docs.unrealengine.com/5.0/en-US/.
(Assessed on 11/08/2021)

[12] *post-process-effects.*
https://docs.unrealengine.com/5.0/en-US/post-process-effects-in-unreal-engine
(Assessed on 15/08/2021)

[13] *post-process-effects.*
https://docs.unrealengine.com/5.0/en-US/post-process-effects-in-unreal-engine
(Assessed on 15/08/2021)

[13] *classes disappear issue.*
https://forums.unrealengine.com/t/my-c-classes-disappear-after-reopening-the-project-why/535990
(Assessed on 20/08/2021)