National College of Ireland

BSc in Computing

2015/2016

Graham Robinson

x12486282

graham9robinson@hotmail.com

Captain Ecks

Technical Report

Project

Table of Contents

Executive Summary

The game being detailed throughout this technical document is called "Captain Ecks". The player will take control of the captain as they attempt to return to their ship by fighting through a number of enemies. The player will need to manage abilities and loot as they upgrade their character in order to face bigger challenges and out score their friends. Created using unreal engine, Captain Ecks uses blueprints to adapt to the player’s choice of difficulty level, determining spawn rates and health percentages that will in turn have a great impact on how you will approach the game. This project includes key elements such as;

- **Artificial intelligence** in the form of two different types of enemies that will actively pursue and attack the player.
- **Character abilities**. As the game progresses players will be able to purchase abilities to use during gameplay, once an ability has been activated a short “cooldown” time will need to take place before that ability can be performed again.
- There is a **Loot system** within Captain Ecks, during gameplay the player can collect coins which are then converted into “Loot”, this loot can be spent in the shop on upgrades such as better weapons, more health, new abilities, etc.

Captain Ecks provides a fun and challenging pirate adventure, with abilities and a loot system adding to the strategy involved with playing the game.

1 Introduction

The objective of this project is to create and develop a fully functional game based on combat and loot. The player will take control of Captain Ecks, and will fight their way through hordes of enemies while simultaneously collecting loot to acquire upgrades for their character.

Throughout game play the player can upgrade their damage and health to do more damage to enemies and to reduce damage taken respectively.

The main features of the game will be as follows, full character control (walk, run, jump, attack, block etc.), enemies of varying difficulty (common enemies, bosses etc.), immersive pirate themed islands, sound effects, and functioning economy in terms of both loot spawns and scaling upgrade pricing.

The player will be faced with the task of fighting through hordes of enemies and navigate through the pirate themed islands, once they have reached the final island by acquiring enough souls, they will unlock the soul gate and the game's boss enemy will spawn.

The player will need to keep an eye on their health bar (how much damage they can take before dying) and their cool-down time (abilities can only be used every 10 seconds or so.) while traversing the islands in an effort to take down the games boss character.

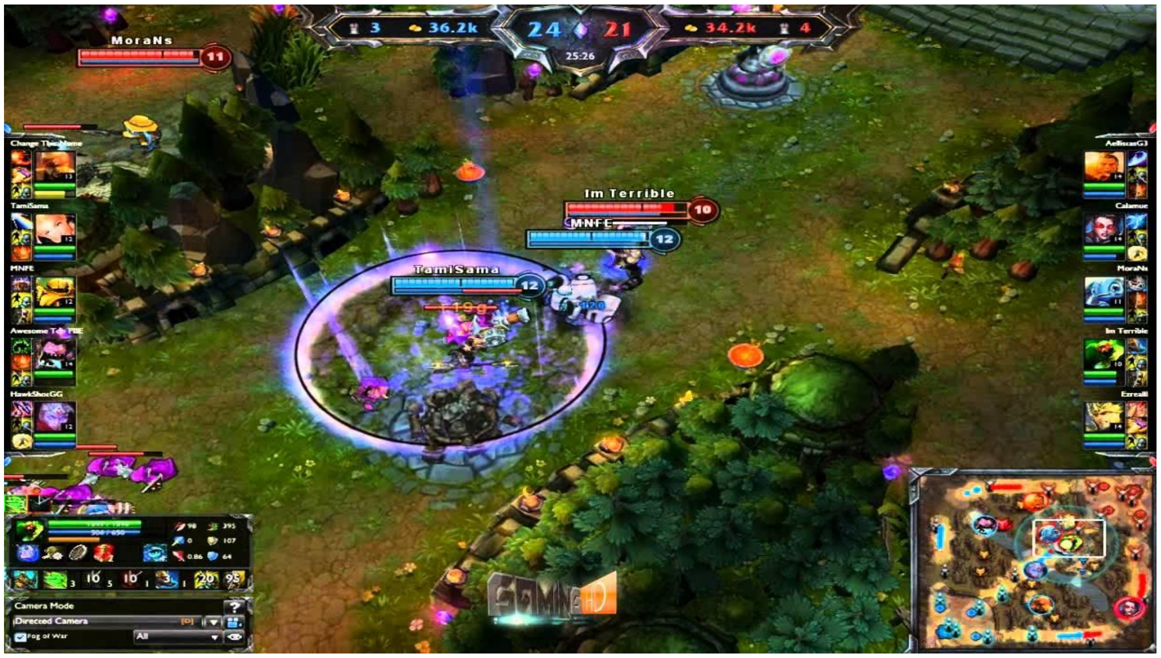
1.1 Background

The game industry has grown tremendously over the past 20 years; many computers owe their biggest breakthroughs to the gaming industry which provided graphics cards, faster CPUs, and 3D graphic accelerators.

The game industry now rivals the film industry in terms of both sales and profit and continues to smash records. The bestselling console of all time is the PlayStation 2 which sold over 155 million units since its release in the year 2000 while the bestselling video game of all time is Mine craft which has sold upwards of 70 million copies across many various platforms including playstation3, PlayStation 4, Xbox 360, Xbox one, android, IOS, PC, etc. these are only the top two examples with other consoles and games selling just as impressively.

Video Games have grown in popularity so much that some games are now considered sports or "e-sports". These games are usually competitive online multiplayer games such as league of legends or Dota 2, but single player games have also been included in the e-sports line up. These games are played by professionals in worldwide tournaments for incredible cash prizes with the fourth international Dota 2 championship in 2015 having a record breaking prize pool of $10.9 million. It is estimated that in 2013, 71,500,000 people watched competitive gaming world wide, even BBC televised the most recent League of legends world championship.

League of legends is a massively multiplayer online (MMO) game where two teams of 5 or 3 fight against each other in order to progress to the enemy nexus and destroy it. The game uses an isometric view as shown below;



The reason I am including this in this document is to show the method of the isometric camera view that will be utilising in Captain Ecks.

Captain Ecks is a third-person, pirate themed, isometric action game. Third-person video games have been very popular since the late 90’s with many play station one games adopting the camera angle. Some of those games include, crash bandicoot and Spyro. The isometric view that this game is undertaking is a sub-genre of the third person view that is rapidly growing in popularity with many indie games. Isometric games are very popular in both mainstream blockbuster games and games bought through app stores such as IOS and the android play store allowing much more position awareness then the standard third person point of view on account of its zoomed out feel and diagonal point of view.

I have always been interested in games and throughout the years video games have evolved to a much more polished and realistic level, with some cinematic scenes looking as lifelike as actors on a set. I have played many games across a lot of consoles and have played games of all types of genres including, first-person shooters, third-person, top-down, 2D side-scrollers, etc. I chose to do my project on this topic because not only is it something I have a huge interest in but I feel it could also evolve into a really good and interesting game.

Gaming has covered nearly every topic imaginable but I feel like there is relatively little content based around pirates in the medium, even less using the isometric camera method, therefore I feel there is a genuine gap in the market for a game like Captain Ecks.

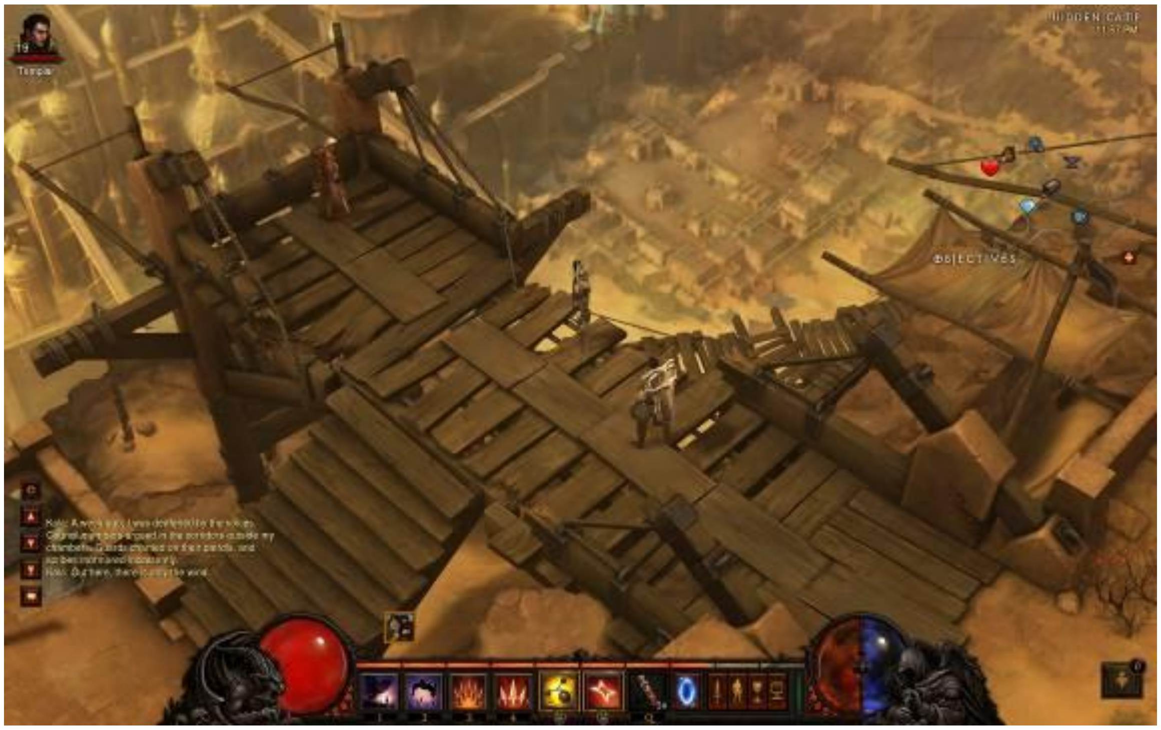
1.2 Research

I began my research by looking for games that shared the isometric camera angle and other than league of legends which was previously mentioned, I found one other popular example;

1.2.1 Diablo III

Diablo 3 is and action oriented role playing game developed and published by Blizzard Entertainment. It is the third instalment of the franchise and was released worldwide on May 15th 2012. The game was quick to rise to success, accumulating 3.5 million sales in the first 24 hours alone, and growing up to 6.3

million sales in its first week. The primary attributes of the character in this game depends on what items the user decides to equip to their character, with millions of combinations of items the degree of customisation is almost endless. While the item and currency system within this game proves interesting to me in terms of how I can improve Captain Ecks, the main reason I am mentioning Diablo 3 is because throughout the entire game the camera angle is at a fixed isometric view. Below is a screen shot from Diablo 3 that shows a similar angle to the league of legends screenshot from earlier;



This view presents the player with far more information than the traditional third person view and allows more freedom in terms of level design. It allows the player to see all of the events occurring from every angle, in front, to the side, and behind the player's character.

The second area of my game I wanted to research was the pirate theme. While I found a large number of games that made use of an isometric view I did not find any that had the same theme as "Captain Ecks". Through this information I realised that there is a gap in the market for a game like this and should I decide to develop the game any further, may find an audience with individuals interested in both pirates and RPG games that are displayed through an isometric camera angle. The research carried out on the pirate genre of video games led to one particular game;

1.2.2 Assassin's Creed IV: Black Flag

Assassin's Creed IV: Black Flag was released on October 29th 2013, it is the sixth major instalment of the Assassin's Creed franchise, this game was researched more for its theme and setting as opposed to its actual gameplay, mechanics, and functions. The game is set in a fictional version of history with real world events; however it follows a fictional pirate protagonist called Edward Kenway. The main elements of this game that I was interested in researching was its character design and level design as I wanted to see how I could make a game feel like it was set in a pirate themed world and what core features I would need for that to be a reality.

Below is a picture of the main character of the game, Edward Kenway. Early in the development process when I was considering creating a character using blender from scratch I used this image as a reference as to how I would structure the clothing of my character. As time became an issue it became a more realistic idea to import a character from mixamo, however this image is still being used as a reference for what types of weapons will feature in the game and also how a character would hold such weapons.

The next image shows the type of location typically found in Assassin's Creed IV: Black Flag. This image will be used as a constant reference when level design becomes a bigger priority as it shows just what kind of objects are found in the genre, such as barrels, trees, and houses. While the ocean is not the games main terrain it is always visible while outdoors, and I feel this is a subtle element that really intensifies the sense that this game is heavily influenced by the pirate genre.

Through these two examples I now have a stronger sense of how I want to develop this game in a way that is both unique and compelling. By following outlines set by previous successful video games and by borrowing elements that I belive were a success, I hope to create a game that can be easily defined as a pirate themed action adventure.

1.3 Aims

The scope of the project is to develop a Pirate themed adventure game that allows the player to walk, run, fight enemies and buy upgrades. The game will incorporate a loot system that will allow the player to collect coins while exploring the world and at the player will then be given the opportunity to spend their coins on upgrades from the shop. The player will be given a choice of difficulty level once they start the game, the three difficulty options will be, easy, medium, and hard. Some of the differences dependant on difficulty level include; Number of enemies, strength of enemies, and frequency of enemy spawns from loot sources such as barrels or chests. The user will have to manage things such as health, cool downs, and loot. The game will incorporate saving and loading although the way in which this will be included is still open to changes.

1.4 Technologies

The game is created in unreal engine, which is a very powerfully and popular game development engine used by both amateur developers and major game developers. There are many libraries in unreal engine that can be availed of to improve the aesthetics of any project. I have made use of these libraries for both the enemies in my game and also the sword that the captain carries which are both available from the open source Infinity blade character and weapons bundle.

Adobe fuse is a program developed by mixamo that allows users to easily design and build 3-D characters that can then be imported into programs such as blender and unreal engine. Fuse also allows users to easily attach animations to any skeleton or rigging. Mixamo Fuse was used to create the main character of the game.

2 System

2.1 Requirements

2.1.1 Functional requirements

The functional requirements of Captain Ecks are as follows;

- Start game
- Save Game
- Load Game
- Choose Difficulty
- Tutorial
- Pause/Options
- Exit Game
- Acquire Loot
- Fight Enemies
- Shop

Start new game

The user should be able to begin a new game when starting the program.

Save Game

The player must have the ability to save their progress as it will not always be possible for a person to play the game in full from beginning to end in one sitting without pauses.

Continue Game

The player should have the option to pick up where they left off and resume a previous game session to prevent the necessity to restart the game each time they begin playing.

Choose Difficulty

The player will be presented with a choice between three difficulty levels, Easy, Normal, and Hard. This choice will determine factors such as the rate that enemies spawn, the amount of enemies there are and the strength of enemies to name a few.

Tutorial

The Game will present the player with a very short tutorial at the beginning of the first level to allow the user to get to grips with the controls and the core elements of the game such as loot sources and enemies.

Pause/Options

The game will allow the user to pause the game at any time, it will be through the pause menu that the player will be able to change options such as sound volume or screen resolution.

Exit Game

This function will be accessed through the pause menu, the user will simply be redirected to their desktop and the game will close.

Acquire Loot

Coins will be dispersed throughout the level and there will be a counter that will keep track of how much loot the player has collected. Coins will spawn from loot sources such as barrels or chests.

Fight Enemies

The user will have to fight through enemies to progress by using all of the tools at their disposal.

Shop

There will be a shop feature that will allow the player to spend the loot they have collected on new weapons or character enhancements.

2.1.2 Changes to the functional Requirements

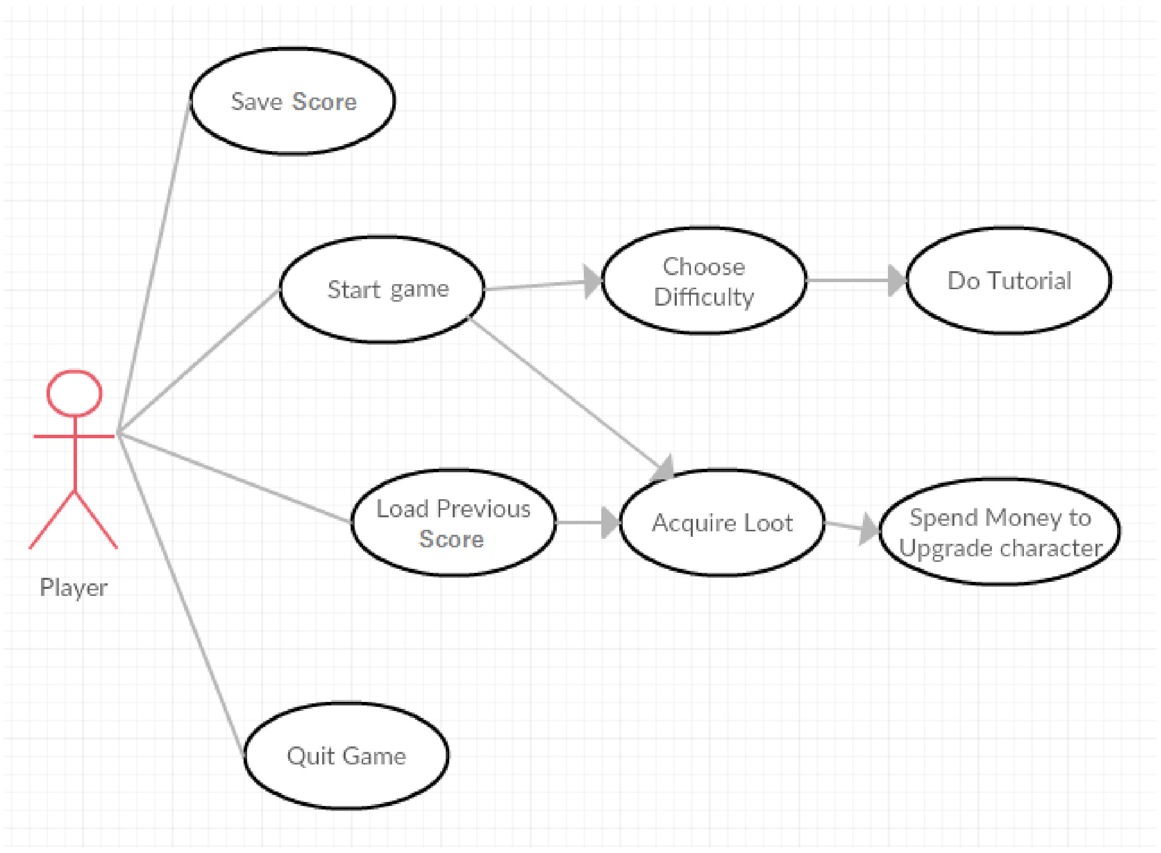
Throughout development it is common for the requirements of a game to change and evolve as the game nears completion. While many of the initial requirements have remained intact a couple of them have been transformed into other requirements, such as the saving and loading of the game. Initially it was intended to allow the player to save their progress and quit the game, then upon reopening the game they will be presented with three save slots allowing them to choose the game they wish to resume, this idea was overtaken by the idea of having a leader board in the game which changed the dynamic of the game in a way. Instead of allowing the player to just save their game at any time and reload it whenever it suits them, the game is much more challenging when the user is afraid to die within the game world, to create this tension there must be a punishment or consequence for dying. In the popular video game franchise "Dark Souls" whenever the player dies they respawn but instead of coming back from death with everything they had at the time of their death, they respawn with nothing and every enemy that they killed comes back to life. This consequence to death forces the player to make important decisions while they are playing the game in order to keep both their loot and their progress safe and also prevents the game from being too easy for the player to complete. Instead the save and load function has been applied to the score that the player sets once they have completed the game, when the player completes the game a score will be generated for them based on how they played and that's score and the player's name will be saved to the games records, their score along with all of the previous scores set by other users will then be loaded into the game whenever a player wishes to view the game's leader board. Therefore the functional requirements Save Game and Load Game have transformed into the functional requirements, Save Score and Load Score.

There has also been an addition to functional requirements in the form of an ability. The player will have an ability at the start of the game that can be used as a once off high damage attack that can instantly kill enemies, however once used the ability must go on a cool down period which can last up to 10 seconds.

As I mentioned before changes are common and in some cases inevitable when developing a game, I believe that the changes made benefit both the game play and the complexity of my project, and am pleased that the scalability of my project allowed for me to evolve these functional requirements with little resistance from the other functions of my project, this is an encouraging sign for the further development of my project.

2.1.3 Use Case Diagram

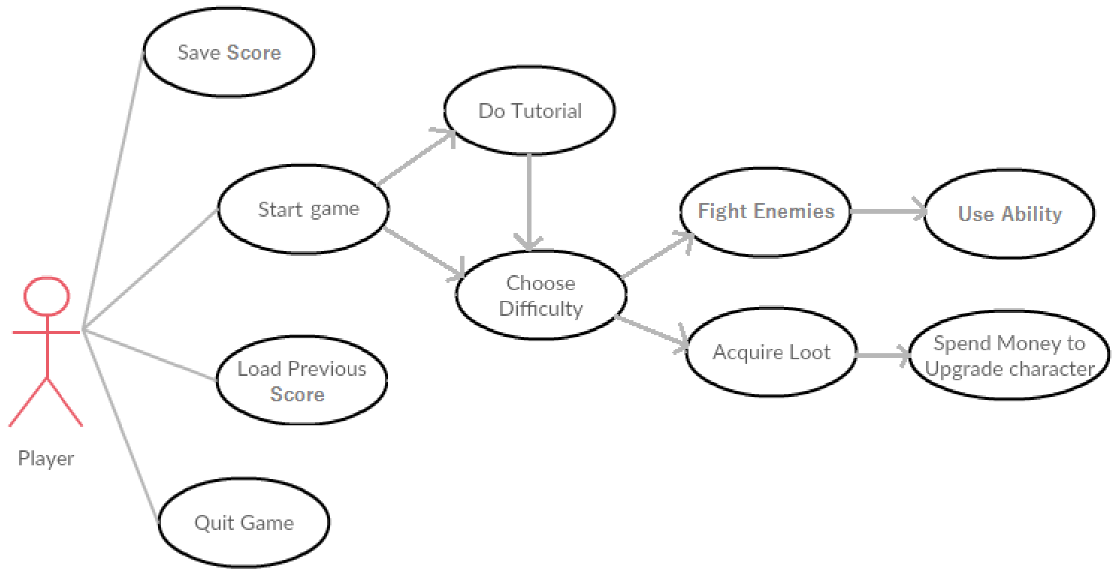
The Use Case Diagrams below provide an overview of all the functional requirements involved with my project and also shows how they have evolved throughout the development process.



Use Case #1: Initial Use Case before any changes were made

Use case #1 above shows the initial interpretation of what the functional requirements would have looked like if there were no changes made throughout development.

Below is the current use case for the project that takes into account all of the changes and requirement evolutions that took place during the development process.



Use Case #2: Current Use Case after changes have been made

2.1.4 Requirement 1 Start game

2.1.4.1 Description & Priority

The player begins a new game session; this requirement is the most important requirement and must be completed before any other requirements can take place.

2.1.4.2 Use Case

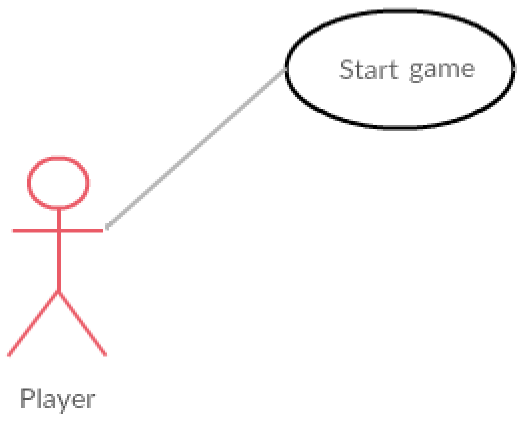
Scope

The scope of this use case is to allow the player to run the program and begin the game.

Description

This use case describes the process of beginning the game with all the base stats and tools.

Use Case Diagram



Flow Description

Precondition

The system is in initialisation mode.

Activation

This use case starts when a player has started the game

Main flow

1. The system identifies the Player
2. The player chooses to start the game from the menu
3. The system begins the game

Termination

The system presents the game to the user

Post condition

The system goes into a wait state

2.1.5 Requirement 2 Do Tutorial

2.1.5.1 Description & Priority

This requirement gives the player an option whether to play the tutorial level or not.

2.1.5.2 Use Case

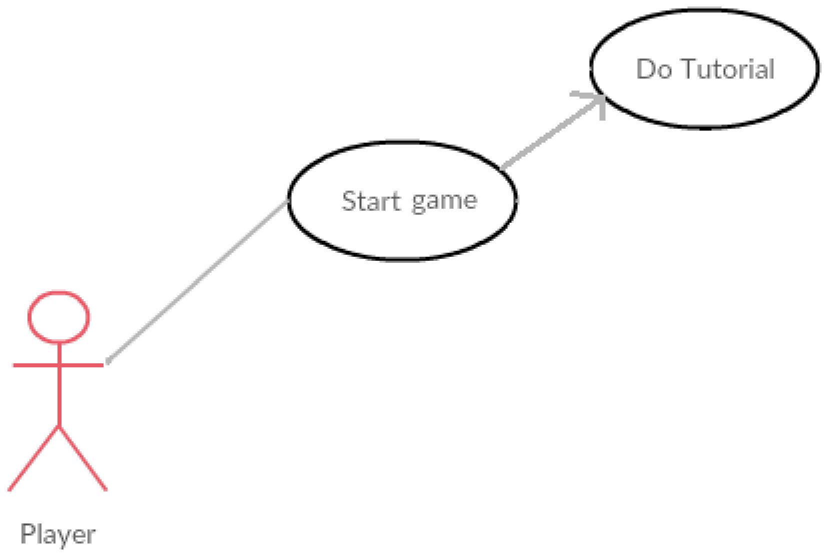
Scope

The scope of this use case is to allow the player to partake in a tutorial level in order to come to terms with the game's controls before beginning the main game.

Description

This use case describes the process of performing a simplified set of tasks to allow the user to get the grasp of game play elements in a short secondary level.

Use Case Diagram



Flow Description

Precondition

The player has started the game.

Activation

This use case starts when a player has chosen to play the game and selected "Play" from the main menu.

Main flow

4. The player has chosen to play the game
5. The system presents the user with a choice of whether to play the tutorial level or not.

Alternative Flow

A1. If the player chooses no then the choose difficulty use case will begin

A2. If the player chooses yes then the tutorial level will begin

Termination

The system presents either the tutorial level to the user (if they selected yes) or the main game to the user (if they selected no).

Post condition

The system begins the real game

2.1.6 Requirement 3 Choose Difficulty

2.1.6.1 Description & Priority

The player chooses the difficulty of the game from the following choices; easy, Medium, and hard.

2.1.6.2 Use Case

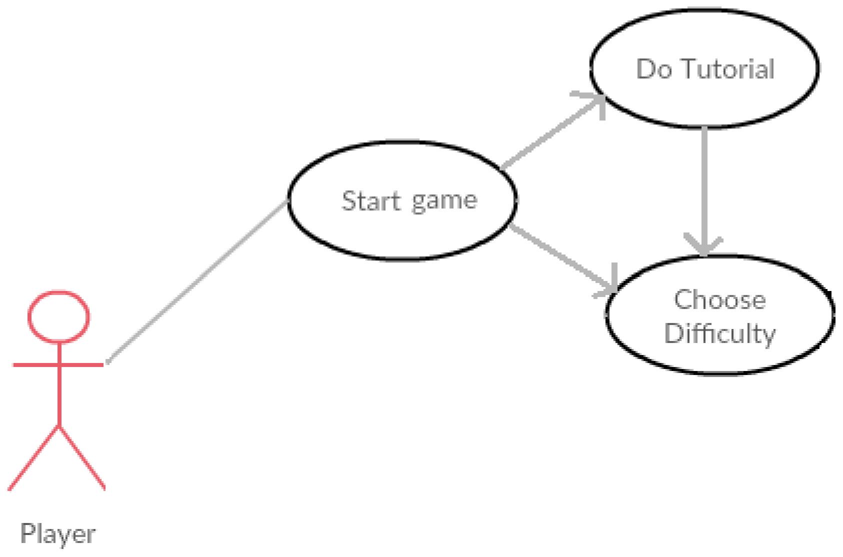
Scope

The scope of this use case is to allow the player to have control over how challenging they want the game to be. The difficulty level will impact things such as the amount of damage the player does or the amount of health their enemies will have.

Description

This use case describes the choice the player must make to determine the difficulty of the game.

Use Case Diagram



Flow Description

Precondition

1. The player has decided not to do the tutorial.

2. The player has completed the tutorial

Activation

This use case starts after the "Do Tutorial" use case has been resolved.

Main flow

6. The system identifies that the player has completed the "do tutorial" use case.
7. The Player chooses the desired difficulty level.
8. The system begins the game using the selected difficulty level.

Termination

The system presents the main game to the user.

Post condition

The system goes into an wait state to await further input.

2.1.7 Requirement 4 Fight Enemies

2.1.7.1 Description & Priority

This Use Case allows the player to fight and defeat enemies. This use case provides the core element of game play in the game and is therefore a very important use case.

2.1.7.2 Use Case

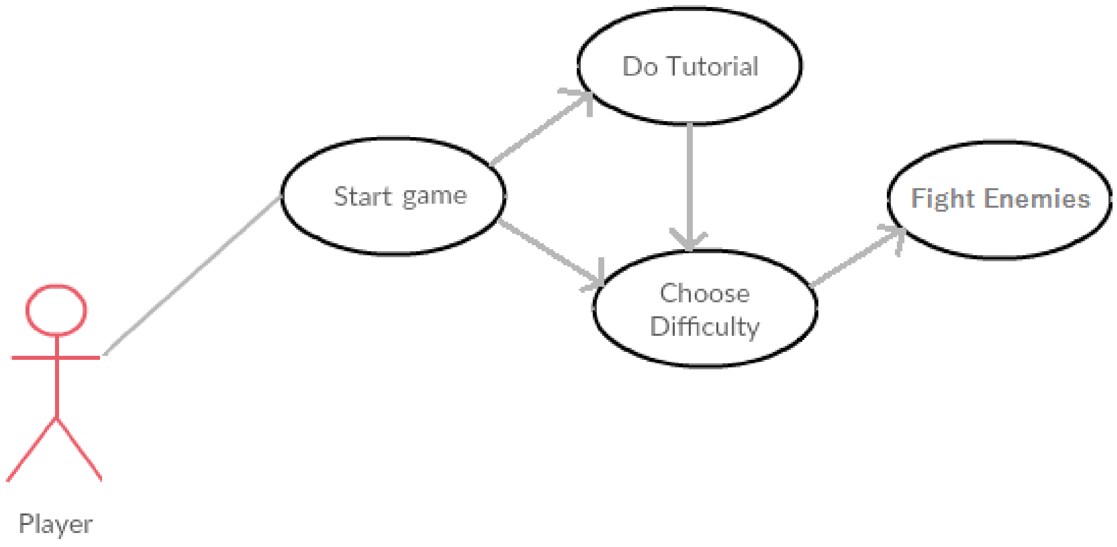
Scope

The scope of this use case is to allow the player to attack enemies while also allowing the enemies to attack the player.

Description

This use case describes the process of fighting with in game enemies.

Use Case Diagram



Flow Description

Precondition

The system is in a wait state after the game has been started.

Activation

This use case starts when a player has approached an enemy as they will automatically chase and attack the player.

Main flow

9. The player is spotted by an enemy
10. The enemy chases the player
11. The player begins to fight the enemy until one of them are destroyed.

Termination

Either the player dies or the enemy is defeated.

Post condition

The system goes into a wait state to await further input from the user.

2.1.8 Requirement 5 Use Ability

2.1.8.1 Description & Priority

This Use Case allows the player to use a high damage attack called an ability.

2.1.8.2 Use Case

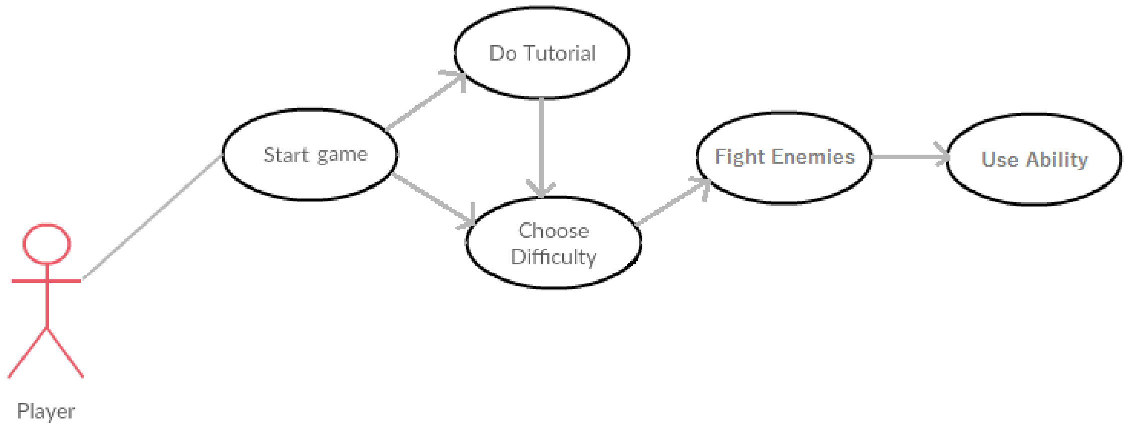
Scope

The scope of this use case is to allow the player to use an ability that can instantly kill enemies at the cost of a small cool down.

Description

This use case describes the process of the player using an ability

Use Case Diagram



Flow Description

Precondition

The system is in a wait state

Activation

This use case starts when a player has clicked on the button that activates their ability.

Main flow

12. The system identifies that the Player has clicked on the ability button.
13. The ability is carried out.
14. The ability goes on cool down.

Termination

The ability is carried out and any enemies hit are destroyed.

Post condition

The ability goes on cool down and cannot be used for a few seconds. The game returns to a wait state.

2.1.9 Requirement 6 Acquire Loot

2.1.9.1 Description & Priority

The player has the ability to collect coins while exploring the world; these coins will be counted and monitored to be used when visiting the shop.

2.1.9.2 Use Case

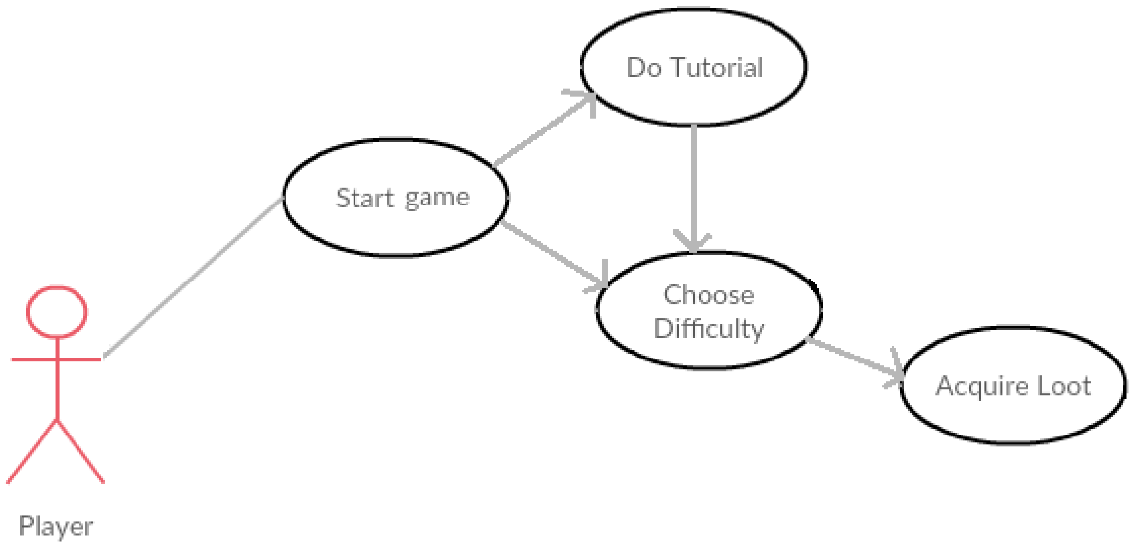
Scope

The scope of this use case is to allow the player to collect coins to be spent on upgrades at a later time.

Description

This use case describes the process of collecting coins.

Use Case Diagram



Flow Description

Precondition

The game has begun.

Activation

This use case starts when the player has come into contact with a coin..

Main flow

15. The player comes into contact with a coin
16. The player's coin total is increased
17. The coin is destroyed and play resumes

Termination

The coin is collected and the player's "Loot" amount is increased.

Post condition

The game displays the current amount of coins that the player has and the game resumes play.

2.1.10 Requirement 7 Spend Money to Upgrade Character

2.1.10.1 Description & Priority

The player can spend any coins collected from within the store screen and purchase upgrades for their character.

2.1.10.2 Use Case

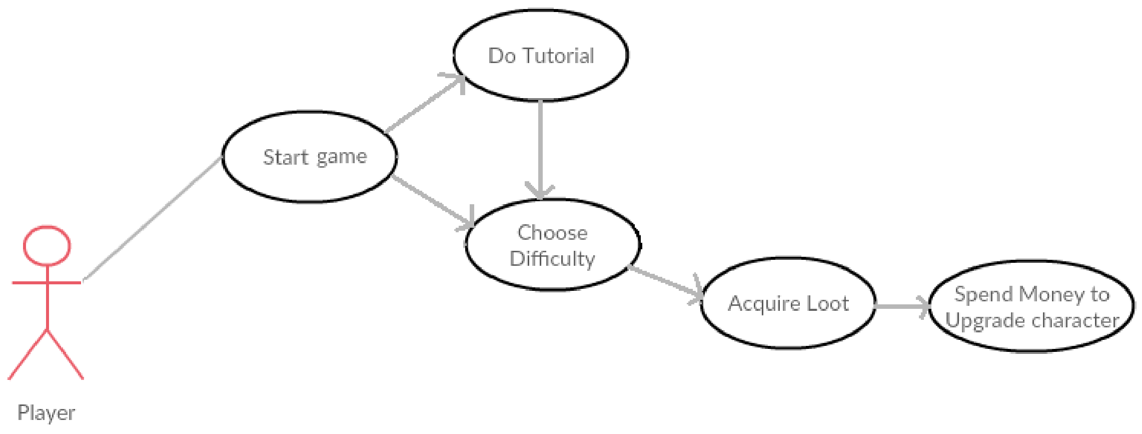
Scope

The scope of this use case is to allow the player to purchase upgrades for their character using coins collected in game.

Description

This use case describes the process of spending coins that were collected on new tools and upgrades for their character.

Use Case Diagram



Flow Description

Precondition

The player has collected coins while playing the game and opened the shop menu.

Activation

This use case starts when a player has opened the shop menu in-game.

Main flow

18. The system presents the shop menu
19. The player chooses which upgrades to purchase
20. The system resumes the game with the changes/upgrades applied

Termination

The player closes the shop menu

Post condition

The system resumes play with the new upgrades applied to the character

2.1.11 Requirement 8 Save Score

2.1.11.1 Description & Priority

The players progress must be saved in some way in order to provide a competitive element among friends and to promote replaying the game once it has been completed once.

2.1.11.2 Use Case

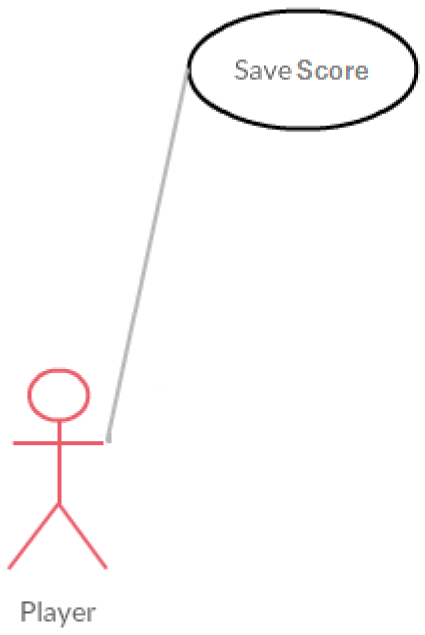
Scope

The scope of this use case is to allow the player's score to be saved to the game and the end of the game session.

Description

This use case describes the process of saving the player's score and allowing the save file to be called upon at a later date through the "Load Score" use case.

Use Case Diagram



Flow Description

Precondition

The game has been completed by the user and a score has been set.

Activation

This use case starts when a player has set a score after completing the game.

Main flow

21. The system identifies the Player has set a score by completing the game.
22. The system then saves the player's score.
23. The system also saves the player's name.

Termination

The player's score is saved to the Score.sav file within the game files.

Post condition

The player moves onto the leader board page.

2.1.12 Requirement 9 Load Previous Score

2.1.12.1 Description & Priority

This Use Case allows the game to retrieve the score variable from a previous game session in order to populate the games leader board with the scores set by past players. This use case is very important as it adds a competitive aspect to the game.

2.1.12.2 Use Case

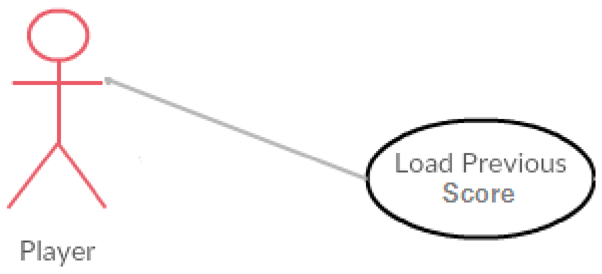
Scope

The scope of this use case is to allow the game to load statistics and variables from a previous game session and to use them to display a scoreboard in the current game session.

Description

This use case describes the process of using variables from a previous game session.

Use Case Diagram



Flow Description

Precondition

The player has called for the leader board page to be displayed.

Activation

This use case starts when a player has opened the leader board page as the game automatically fills the leader board with the names and scores of previous players.

Main flow

24. The system identifies that the Player has opened the leader board page
25. The system then loads the names and scores of previous players onto the page
26. The system player can then view the now populated leader board.

Termination

The system presents the loaded variables to the user

Post condition

The player can then return to the main menu of the game.

2.1.13 Requirement 10 Quit Game

2.1.13.1 Description & Priority

The player needs to have the ability to quit the game at any time.

2.1.13.2 Use Case

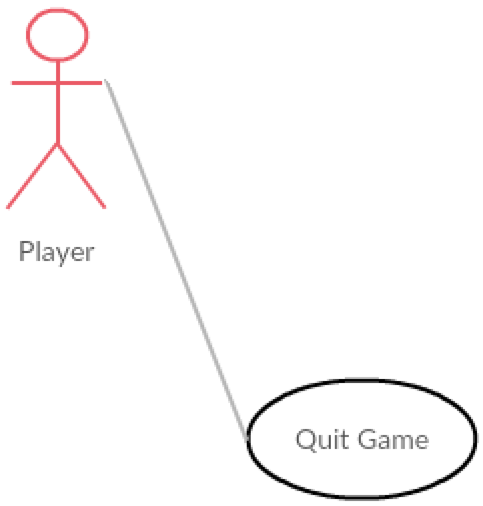
Scope

The scope of this use case is to allow the player exit the game.

Description

This use case describes the process of stopping play and exiting the game state.

Use Case Diagram



Flow Description

Precondition

The game pause menu is open.

Activation

This use case starts when a player has selected "Quit Game" from the pause menu.

Main flow

27. The system identifies the Player has selected "Quit Game".
28. The system terminates the program
29. The player is returned to their desktop

Termination

The game is terminated

Post condition

The system terminates, the user is returned to their desktop and play ceases.

2.1.14 Data requirements

The game allows the users to save the score that they set and therefore the user must have the necessary memory space required to store the Score.sav file free, which is roughly around 1 KB. The game runs on unreal engine and therefore requires the host computer to be able to run the engine, the minimum computer specifications to run unreal engine are as follows;

Operating System:	Windows 7/8 64-bit
Processor:	Quad-core Intel or AMD, 2.5 GHz or faster
Memory:	8 GB RAM
Video Card/DirectX Version:	DirectX 11 compatible graphics card

Unreal engine can be run on devices that have specifications under the recommended statistics; however this would most likely result in a loss of performance or visual quality.

2.1.15 User requirements

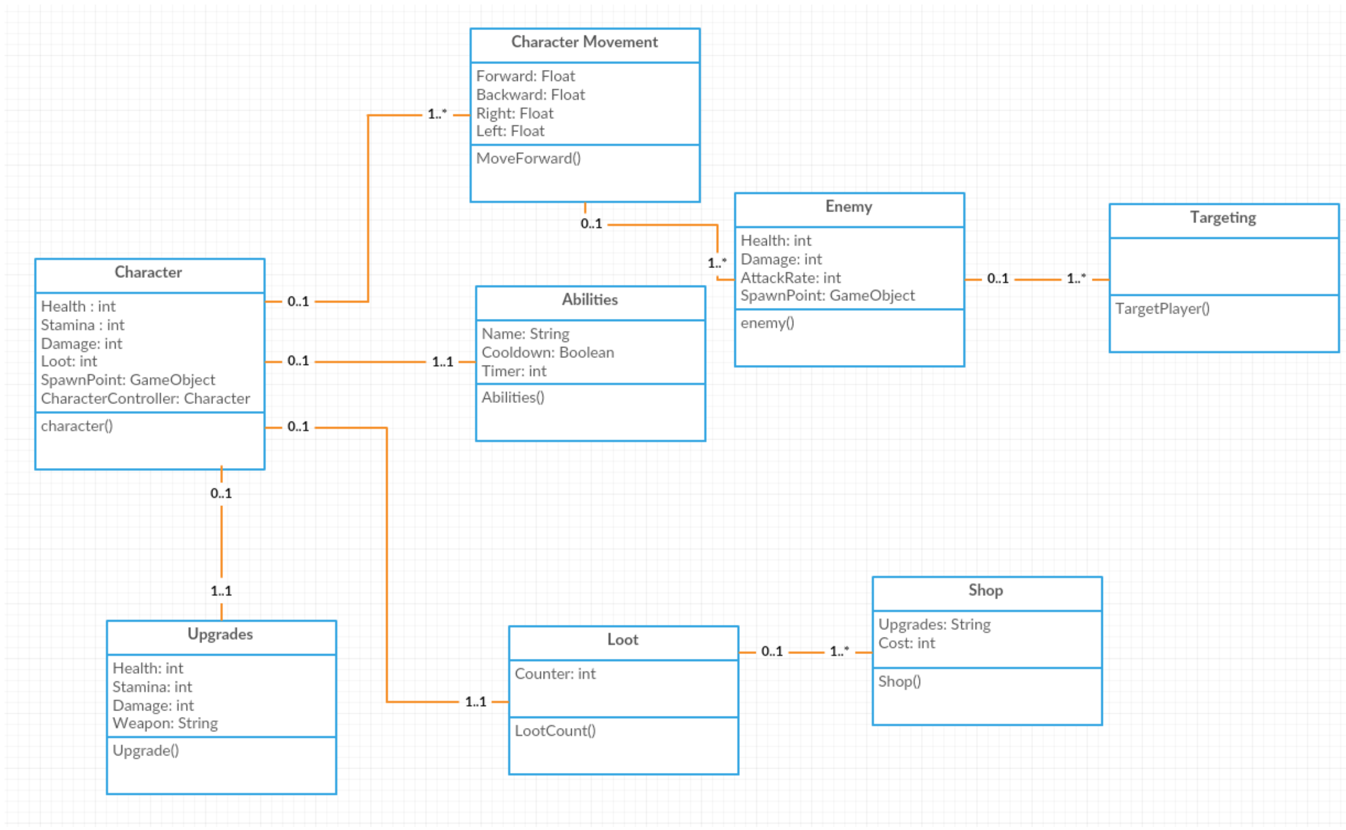
The main objective of this game is to allow the player to explore a virtual world and acquire loot to spend on upgrades in order to defeat stronger enemies. In order for the player to feel fully immersed in the game play of this game they must have a laptop or computer that is capable of running Unreal engine 4 without any issues. There is no need for the user to have internet access while playing the game, just for the initial download of the game itself.

2.1.16 Usability requirements

The games menus must be easily traversed with minimal options to prevent the user from feeling overwhelmed by the paths they can take. This must be accomplished by providing simpler, clear, and well mapped graphical user interfaces that will interlink to provide a seamless menu system. It must also be possible for the user to return to the screen that they just came from in the event of them clicking the wrong button by mistake. The UI should also be aesthetically pleasing and follow a general theme; in this case the theme is "Pirates". There is also a tutorial level present in order to aid the user which will guide them through their first steps in the game to allow them time to come to terms with the games controls in a controlled environment.

2.2 Design and Architecture

The class diagram below shows how the most important aspects of my game relate to each other:



It begins with the character class, this class holds values such as Health, Stamina, Damage, Loot etc. it is in this class that I will keep track of the various stats the player character possesses.

The first class the user will experience when playing the game is the "Character Movement" class, the player will be able to move around in all directions, jump, and attack while playing the game. The user will have access to an ability at the start of the game, which is a high powered attack capable of killing all enemies that get in the way, the ability goes on a short cool down after it is used and therefore requires its own class to keep track of how long the cool down is at any given time. The character then needs to communicate with this ability class in order to find out if they are capable of using the ability at that time.

There will be enemies in the game as can be seen by the "Enemy" class, these enemies actively search for the player at all times and once they have spotted the player they will begin to chase them down until they are within range to attack, therefore the enemy class needs to communicate with a "Targeting" class in order to track, follow, and attack the player as shown in the class diagram above.

There will be a separate class for both the shop and upgrades, the shop class will contain variables for the types of upgrades available and will also contain the cost of each upgrade. Each of the upgrades available will scale up in cost with repeated purchases. The upgrades class will contain all upgrades that the player currently has equipped such as increased damage, heath or ability power, which is why the player character must be able to communicate with the Upgrades class in order to put into effect any upgrades that have been purchased by the player.

Finally Loot will have its own class in which the system will keep track of how much loot the player currently has available and will also keep track of the amount of coins that the player has picked up since they first started the game. The first variable will increase as the player collects coins from the world and decrease as the player purchases upgrades from the shop, and the second variable will not decrease as the player spends coins keeping a total figure hidden from the player to be used at a later date when calculating scores.

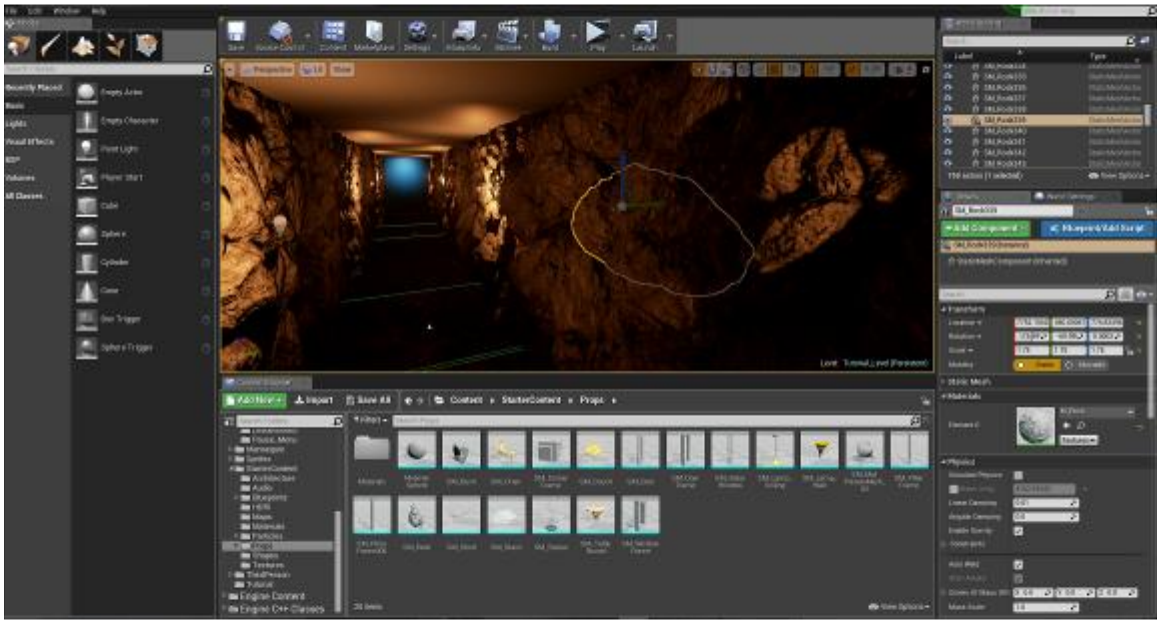
3 Implementation

This game was developed using unreal engine 4 and therefore makes use of the main coding technique used within unreal engine, blueprints. Blueprints are a visual type of coding and are produced by linking many different nodes together. Below I will outline the most important aspects of this project and in the more complicated circumstances, will provide relevant screenshots to coincide with the explanation of how I implemented those functions.

3.1 Environment

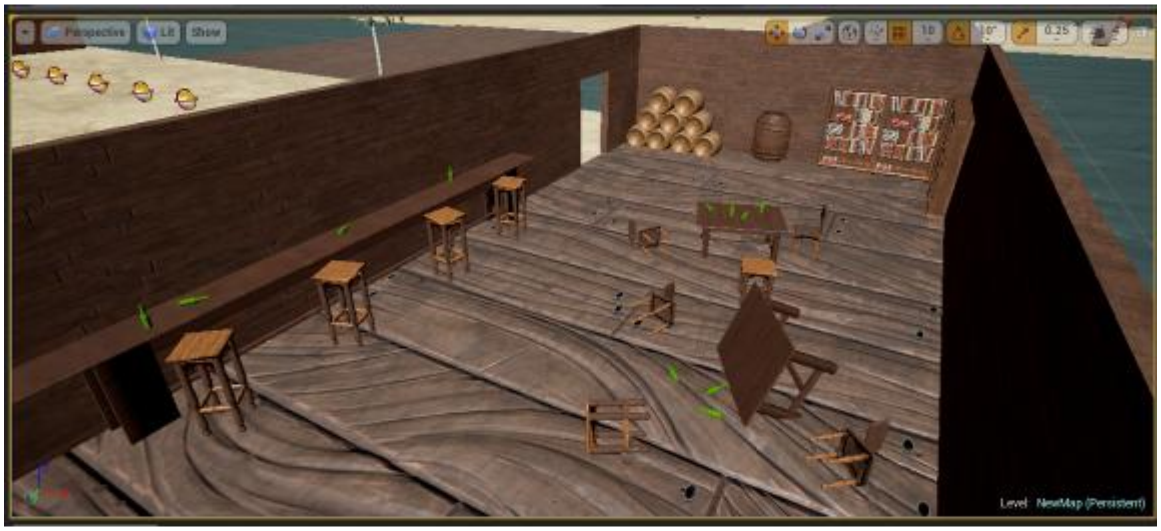
There are two environments in this game, the tutorial level and the main level. At the beginning of the game the player will be given a choice to play the tutorial or to move straight onto the game. If the player chooses to play the tutorial then they will be brought to the tutorial level that looks like the inside of a cave.

It takes many various assets to create these environments and everything that can be seen throughout each level has been individually place by hand using three different techniques, Translate, Rotate, and Scale. By making use of these three techniques I was able to take an asset and manipulate it to fit into the game world to give the player a sense of immersion. As you can see below each rock in the cave wall was placed individually and then was scaled or rotated in a way to best fit with the other rocks in the wall;



The tutorial level is purposefully linear in order to give the player a clear and simple indication as to where they need to go, along their way through the cave they will be prompted with hints that will teach them the basic controls of the game and upon reaching the end of the tutorial level the main game level will be automatically loaded.

The environment of the main level is very different to the tutorial level; it is outdoors and a lot more open for exploration. The level consists of a number of islands all on top of a sea of water with various objects and building scattered around the map. There are many small aesthetic additions that help make the game feel as immersive for the player as possible, for example when the player enters one of the building in the map they are presented with a thrashed bar scene with tables and chairs thrown over and bottles scattered all around;



This scene shows the full effect of how translation, rotation, and scaling can be put to good use to create an immersive scene. To create this scene I first brought the assets onto the scene (e.g. table, chair, bottle), I then applied a material to the asset to give it the right colour to best represent the object, finally I chose the location that the object should be in the game and set its rotation and scale in order to help it fit into the scene. This process was carried out for each and every object in the entire game.

3.2 Character

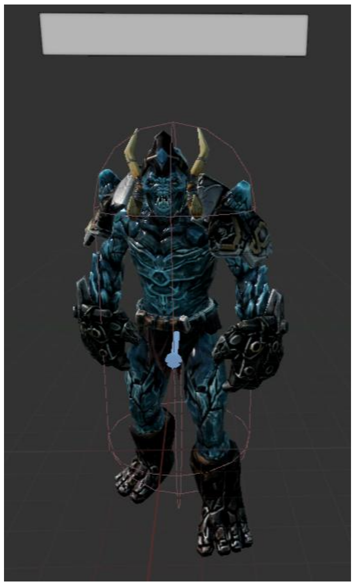
There are three major aspects to the main character; Mesh, Skeleton, and Animation. The mesh of the character is the overall look of the character, it is through the mesh that the colours, shapes, and collision settings of the character is found. The mesh of the main character was created through mixamo fuse while the meshes of the enemy characters, along with their skeletons can be found through unreal engine itself.

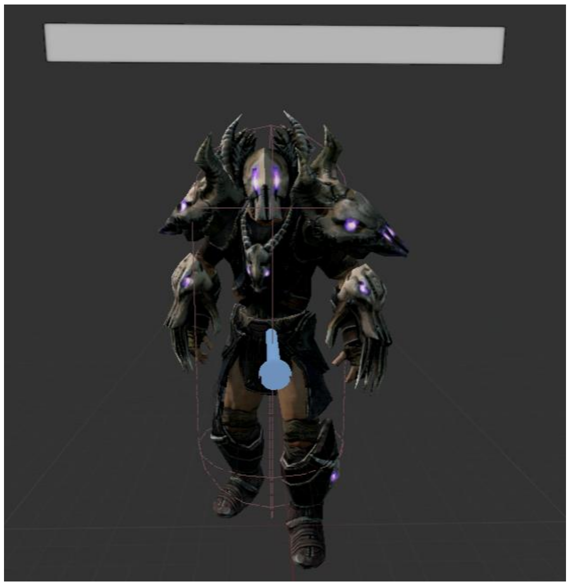
After the character mesh has been imported to the project a skeleton must be assigned to it. It is impossible to animate a mesh if it does not have a skeleton of some sort. It is through the skeleton that we assign sockets to our character. Sockets are used to attach object to the character such as a sword or a gun. To do this I first had to select which bone in the skeleton that I wanted to attach the socket to, in the case of the captain's sword the bone was the RightHand bone which is located in the palm of her right hand. After the socket has been created it is possible to preview the sword when placed into the socket, it is now that I had to align, scale, and rotate the sword in order for it to look natural in her hand. When the socket has been created and the sword has been properly put in place in the preview I had to actually put the sword into her hand, to do this I first had to drag the sword onto the scene I then had to create an instance of the sword in the level blueprints and then through the "AttachActorToComponent" node I selected which socket to connect the sword to and linked it to the sword object instance and also a player character instance.

The final step of the character is the animations. Animations can be called upon when an action is carried out, for example when the player shoots their gun the shooting animation is called, but not all animations should require input from the user, such as when the character is idle and waiting for input. To do this I had to create a blend space. A blend space is a type of blueprint that merges animations together based on things such as how fast the character is travelling, this blend space gives the player a smooth transition between the idle animation (when the player is not moving anywhere), the walking animation (when the player is moving slowly in a direction), and the running animation (when the player is moving at full speed). Another method of adding animations to the character is through an animation blueprint which I used to add the jump animation to the character. Animation Blueprints can be used in many ways and for many different reasons, in my project I used them to check and see if my character is falling and if so it will play the jump animation to give the effect that the character is actually falling through the air. By checking the characters velocity and location it is possible to see whether or not they are falling through the air and if they are I set the Boolean variable "IsINAir" to true while that is true the falling animation is played until it is found that the character is no longer falling.

3.3 Enemies

There are two types of enemy in the game, Normal enemies and the games Boss enemy. It is upon defeating the boss enemy that the player completes the game and sets a score to compete on the scoreboard, which will be discussed in a moment. Both enemy types can be seen below;

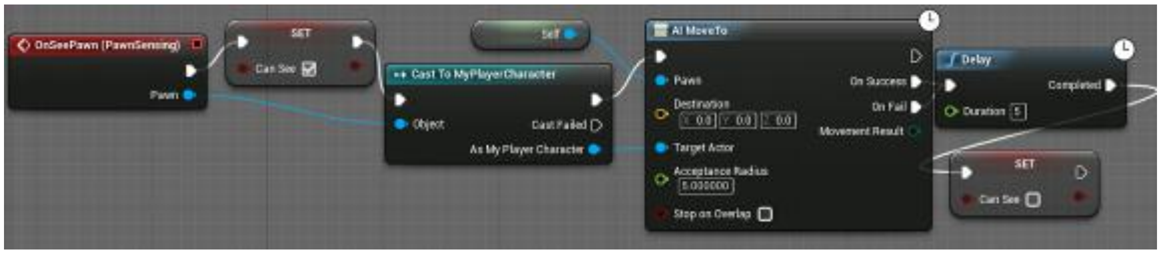

Standard Enemy

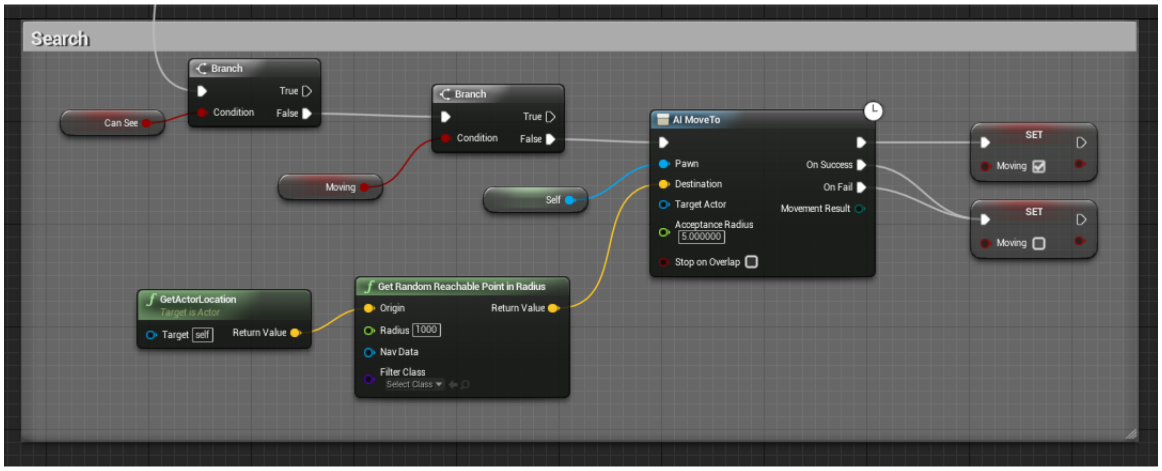

Boss Enemy

The tow sub-classes of enemy work very similar to each other in terms of artificial intelligence, however the boss enemy is far more relentless in its search for the player with a much wider search range, it also has higher health and damage compared to normal enemies making it a huge challenge that requires the player to use all of the tools at their disposal and to put what they have learned throughout the game to the test.

The first thing that needed to be accomplished when creating the enemy AI was to constantly check to see if the player character was visible to the enemy. To do this I had to create a search function that would check and see if the enemy could see the player through the "Can See" Boolean variable. If the enemy can see the player then they will move to the player's location and if they could not see the player then they would pick a random position within their searchable range and travel to that location instead, eventually through constantly switching from random point to random point they will find the player.

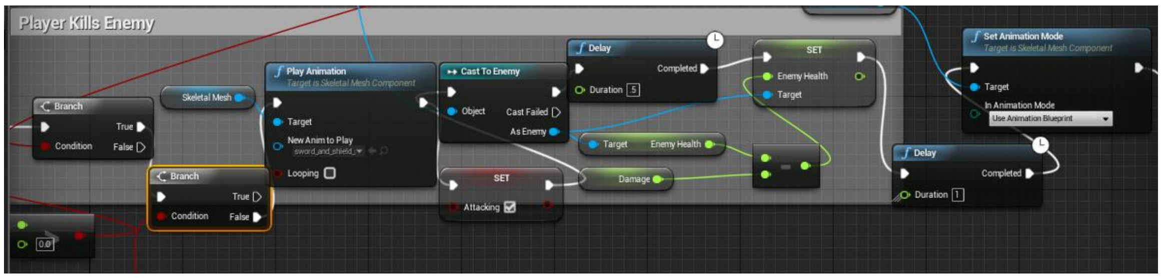
Once the search function was completed I then needed to find a way to accurately assign a value to the "Can See" variable. Within the viewport of the enemy character a pawn sensing component is added, the pawn sensing component allows functionality to be run only once the character has come within a certain range of a pawn e.g. the player character. As soon as the enemy has come within range of our player we set the Boolean "Can See" to true. Next we tell the enemy to move to the player's location which is how the enemy chases the player, finally the variable "Can See" is set to false. The reasoning behind the "Can See" Boolean is to ensure that the enemy does not get stuck in one position and stop searching for the player, by setting it to false we force the enemy character to pick a new position and constantly be moving.





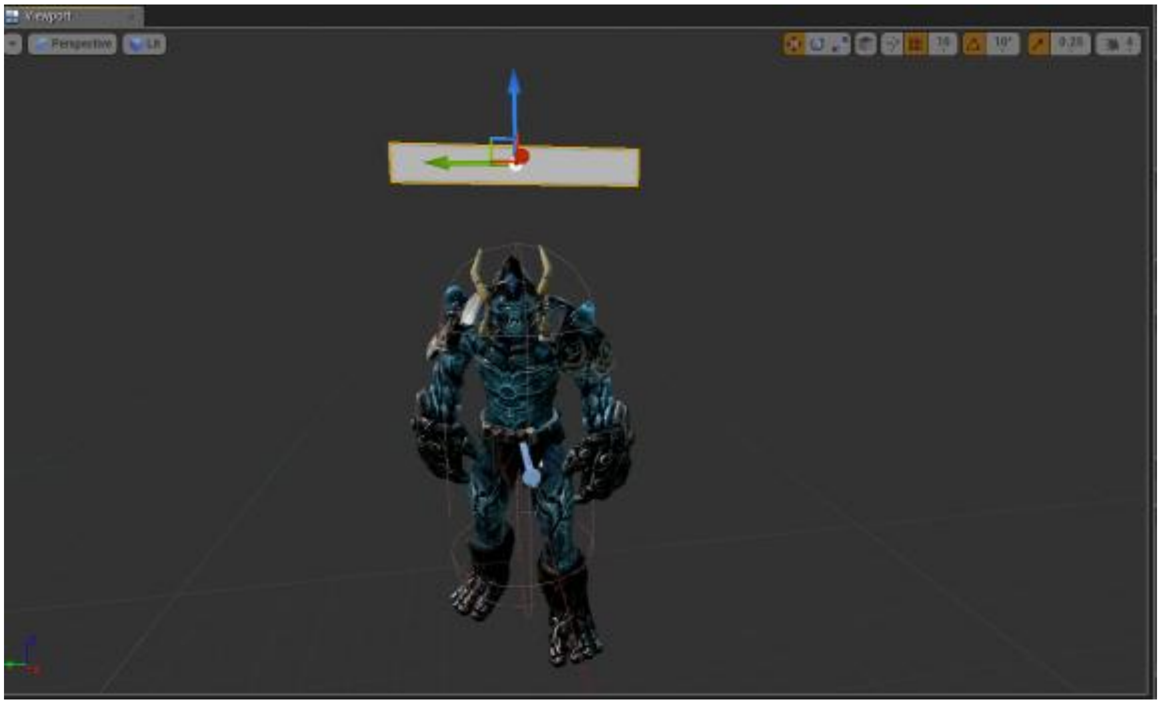
The enemy characters must be able to cause damage to the player in order for the game to be challenging and so my next task was to find a way to do this. I started by creating an "OnComponentBeginOverlap" node in my character's blueprint, this node waits until the mesh of the player character has come into contact with the mesh of the enemy characters and if it does a function will be carried out. In this case the function that is carried out is that the player's health will decrease by a certain amount every 1.5 seconds that the player is in contact with the enemy character. The amount of health that the player will lose is determined by the enemy's damage variable which is set once the player has selected a difficulty level to play the game on.

Once the enemy was able to cause damage to the player the final aspect of the enemy was to allow the player to cause damage to them. To do this I used the same "OnComponentBeginOverlap" node I used before, however I only made it possible to cause the enemies harm when the player clicks the attack button once they are within range. Below is the code that damages the enemy;



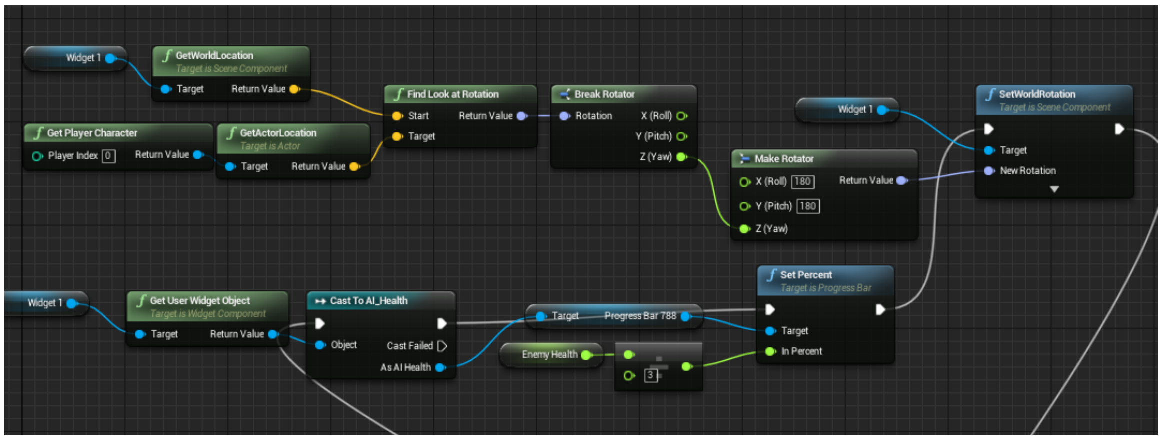
Once the player is both in range of the enemy and clicks the attack button then the player character will send to the enemy character and retrieve the enemy's health value, it will then decrease the enemy's health by the player's damage variable (which is determined by both difficulty level and the amount of upgrades the player has), after that the player character will then send the new enemy health value to the enemy character and it is there that the enemy will check to see if they are still alive, if not the enemy character will be destroyed.

I wanted it to take more than one attack to kill an enemy in order to make the game difficult but I knew that if it was going to take more than one attack to kill the enemies then I would need to display to the player an indication of how many more attacks it will take to defeat the enemy. The best solution I thought of for this problem was to display individual health bars for each enemy above their heads so the player can see which enemies are close to death and which enemies will take a little bit more time to kill. To do this I used widgets and placed them above the enemy's head in the viewport of the enemy's blueprints below;



Once the health bar was in place above the enemy's head I then needed to force it to accurately represent the amount of health that the enemy had left, to do this I made a reference to the health bar widget in the enemy blueprints and then used a node called "Cast To AI_Health" which allows me to access the variables from the "AI_Health" Widget which is where the progress bar is created and stored. I then set the percentage of the health bar to be filled by dividing the enemy's health by 3.

Once I had the health bars all working individually and keeping the correct health values of each enemy I had another problem which was that health bars kept their rotation fixed in terms of where the enemy character was facing, which made it difficult at times to see the health bars properly from the player's point of view. To combat this I decided to keep the health bars facing the player character no matter which way the enemy decided to face. Below is the code that allowed me to carry this function out;



As shown above, once the health of the enemy character is set the health bar is then passed onto a node called "SetWorldRotation" which, as the name suggests, sets which way the health bar is facing within the game world. I then got the location of the player characterthrough the "GetActorLocation" node and then by matching it to the health bar rotation I was able to set the rotation of the health bar to always face the player character allowing them to constantly see how much health each enemy has with ease.

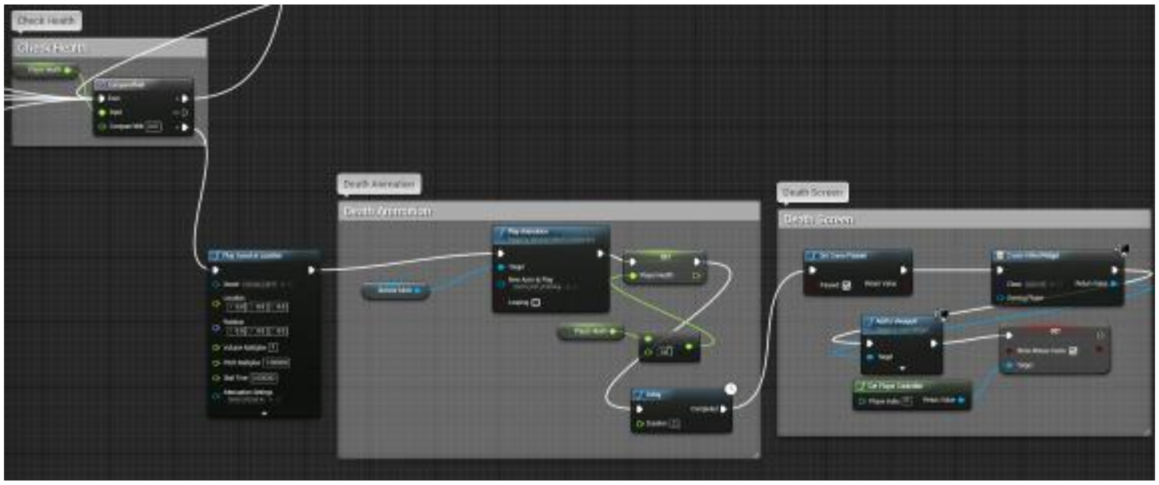
3.4 Health Regeneration

The player's health is the most important stat that must be kept above zero or the player will die and will be forced to restart the game. In many games a player's health steadily and slowly regenerates. I wanted to incorporate this into my game and to do this I needed to check the value of the current player's health at all times and if the player is not in combat with an enemy and is in a safe spot within the game their health will slowly increase to just below 100%.

The main issue with this feature was that it was very difficult to prevent the player's health from passing 100% and filling the health bar more than it should, this became even more of an issue when I introduced character upgrades to the game as the base value of health could then be altered by the player. To combat the problem I needed to put a function in place that could constantly check to see if the player's current health value was between 0.1 and the player's total health which is determined by this small formula; ((the amount of times health has been upgraded) * 20) + 100. As each upgrade increases health by 20 points and the players begin with 100 health points these two things must be taken into account in order to get an accurate figure for the current health points. While the player's health is between 0.1 and the player's max health their health will increase by 10 points every 2 seconds.

3.5 Player Death

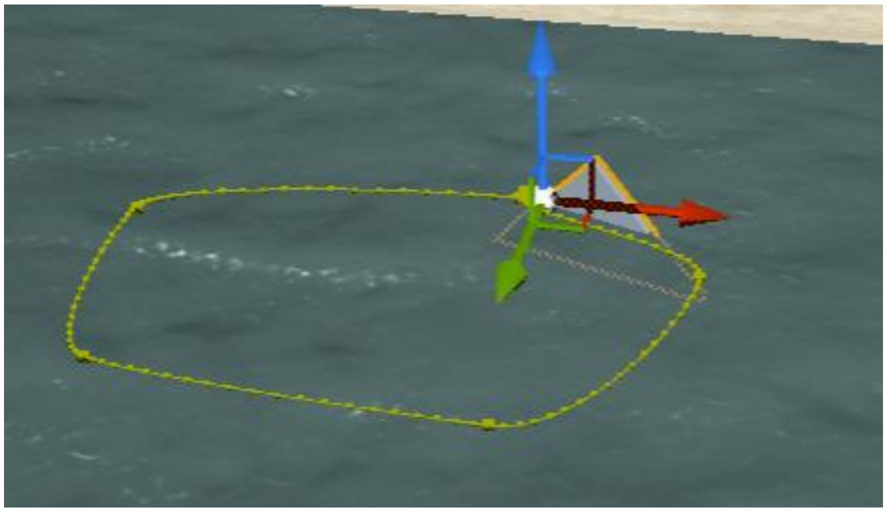
Below is a small example of the blueprints of the player character, it illustrates what happens when the player has died. As you can see the variable "Player Health" is constantly put through a compare float function to determine whether or not the player is still alive. As soon as the player's health falls below 0.01 a short death animation is played showing the player fall to the ground, then in order to prevent health regeneration affecting the animation "Player Health" is decreased by 100. After the animation has had time to fully play out a death screen is displayed giving the player the opportunity to *respawn at the beginning of the level. There are two different ways that the player can die in Captain Ecks, the first and most obvious way is to take too much damage from enemy characters and to allow the player's health to fall below zero, and the second way occurs if the player jumps into the shark infested water beneath them. Each death brings the player to a different death screen both of which giving the player a chance to respawn and retry the level.



**Respawn is a gaming term used to describe the act of a player being brought back to life after death.*

3.6 Sharks

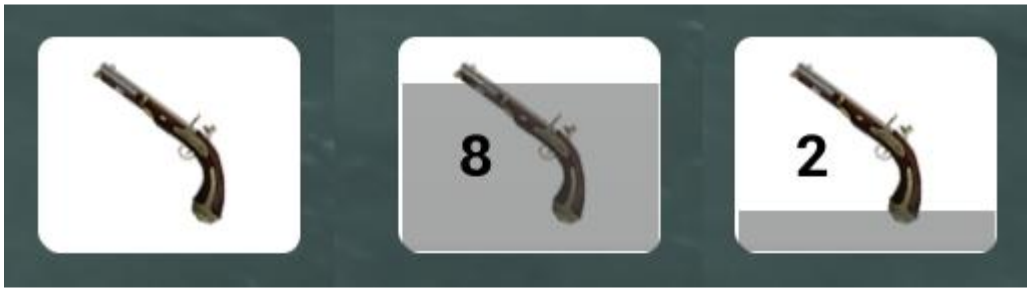
The sharks in the water were created using a simple shape and matinees. Matinees allow you to add movement to your game such as cut scenes or movable objects, through key frames. The way in which matinees work is that they allow you to set key frames and set the position of an object in each cut scene, the matinee then merges the positions together filling in the gaps with generated movement that gives the effect of a smooth singular movement of an object. As you can see below the shark is a simple triangle shape partially hidden beneath the surface and using a matinee I have created a simple path for the shape to follow;



Along the path it is easy to see where the key frames are placed (indicated by the 5 larger yellow squares along the path). By setting the position of the shape at each of the 5 key frames a path is created that in turn allows to shape to seem like a shark is swimming in the water.

3.7 Ability

The player has a "Gun" ability in the game. An ability is a special high damage move that can only be used once every few seconds and therefore the player must be cautious in choosing when is the best time to use their ability or to save it for a harder battle. The player's ability is constantly represented on the in game HUD (Heads Up Display) in the bottom right corner. Once the player has used their ability by click on the right-click button of the mouse they will shoot their gun and the ability will go on cool down, preventing the ability from being used until the cool down is completed. The cool down of the ability is also represented on the HUD in the form of a grey overlay that progressively disappears each second as the cool down counts down as shown below;



As you can see above the first image on the left shows the ability off of cool down and available for use, the image in the middle shows that the ability has been used recently and will need a further 8 seconds of cool down to be used again, and the final image on the right shows the ability further into its cool down with only 2 seconds left until it can be used again. You can also see how the overlay slowly disappears as the ability comes off of cool down.

In order for the ability to work I had to check to see if the ability was off cool down using a variable called "Gun Cooldown" which was set to be zero at default. If "Gun Cooldown" equalled zero I would then check to see if the player was currently attacking and if they were not attacking i would then proceed onto the ability function. If "Gun Cooldown" did not equal zero the game would not proceed to the ability function and a sound of an empty gun click would be played to signal that the ability is not yet off cool down.

Once the "Gun Cooldown" equals zero and the player is not currently attacking we can then start on the gun function by setting the Boolean variable "Shooting" to true preventing the player from moving until the ability has completed and by playing the shooting animation of the character. The character has two "Sockets" which are what is used to hold weapons. One socket is on the character's hip and the second is in her hand, it is by alternating between these two sockets at the correct time that give the effect of the character reaching for the gun on her hip, picking it up with her hand, shooting and then replacing it back on her hip. To do this I broke down the animation into three stages "Pick Up", "Shoot", and "Put Down".

During the "Pick Up" stage of the animation the gun remained on socket 1 on the characters hip then a delay of 0.821 seconds is set in motion before the Boolean variable "Holding Gun" is set to true which in turn signifies that socket 1 must be released and that socket 2 in the characters hand must be called into effect. We are now in the "Shoot" stage of the animation in which the character will actually fire the gun, after socket 2 is activated a further delay of 0.936 seconds is required before the bullet is fired after that a delay of 2.356 seconds is then put in place before socket 2 is deactivated and socket 1 is re-activated, putting the gun back onto the character's hip and entering the "Put Down" stage of the animation. Finally a delay of 0.241 seconds is needed in order for the animation to finish playing out and the Boolean variable "Shooting" is set to false so the player can then move around freely once more.

After the animation of the ability is completed the only thing remaining is to create the bullet that will be shot from the gun and kill enemies. I wanted to bullet to have a sort of cone like area in which damage would be applied which is common in isometric games like this one. The reason this method is preferred to in terms of shooting in isometric games is that it is far more difficult to aim that it would be in a first person shooter where the camera is directly behind the player. To do this I created the bullet object as a cone and flattened it as to not affect any object that were above the player. I then set the bullet to be invisible in game so the player would not see the cone when they used their ability. Finally I had to create a collision for the bullet to destroy any enemy or barrel it came into contact with.

3.8 Loot System

The game revolves around the collection of loot. The player cannot upgrade their character unless they have acquired enough loot to do so and therefore the loot system in the game must be fully functional. To do this I needed to start with 2 different variables, "Total Number Of Coins" and "Number Of Coins". The difference between these two variables is that the first one counts the total amount of coins that the player has collected regardless of how many coins the player has spent on upgrades; this will be used at a later stage when the player's score is being calculated. The second variable counts the current amount of coins the player has after they have spent some coins on upgrades, this is the figure that will be presented to the player while they are playing the game so they know how many coins they currently hold and whether or not they have enough coins to purchases upgrades.

To create the coins I first made a paper flipbook asset within the coin blueprint which would act as the coins mesh. To create the flipbook I needed to upload 6 pictures of the coin to the flipbook;

The flipbook takes these pictures and alternates through them to give the effect that the coin is spinning. After the coin looked like it was spinning I created a sphere collision around the coin in order for code to be added to it.

Once the player comes into contact with the sphere around the coin, the coin will be destroyed and both the "Total Number Of Coins" and "Number Of Coins" variables will be incremented by one, a sound will also play to signify that the coin has been picked up.

3.9 Loot Sources

There are many loot sources spread throughout the game in order to provide a chance for players to acquire enough loot to spend on upgrades for their character. These loot sources come in two forms, chests and barrels.

Barrels are the more common type of loot source and can give only 1 coin at a time. Chests are rarer loot sources and can give the player 10 coins at once. All loot sources have the potential of spawning an enemy instead of loot and so the player must be prepared to fight even when opening a barrel or a chest.

The way the loot sources work is as follows; Once the player character has collided with the loot source the object will be destroyed (in the event of a player character colliding with a chest however the chest will be replaced with an opened chest) and either loot or an enemy will be spawned above the loot sources location. The way that the game determines whether or not to spawn loot is provided by a "Coin Ratio" variable. The system will generate a random number between zero and 100 and it will then check to see if that chosen number is less than or equal to the "Coin Ratio", if it is less than or equal to the coin ratio then loot will be spawned, if however the generated number is larger than the coin ratio an enemy will be spawned and the player will have to fight their way out. The value of the "Coin Ratio" variable is decided once the player has chosen a difficulty level.

3.10 Difficulty Level

Difficulty level will have an impact on game play, the user will be presented with three choices (easy, medium, and hard) and depending on that choice certain variables will be changed including; Player Damage, Enemy Damage, and the ratio at which coins will be spawned from loot sources.

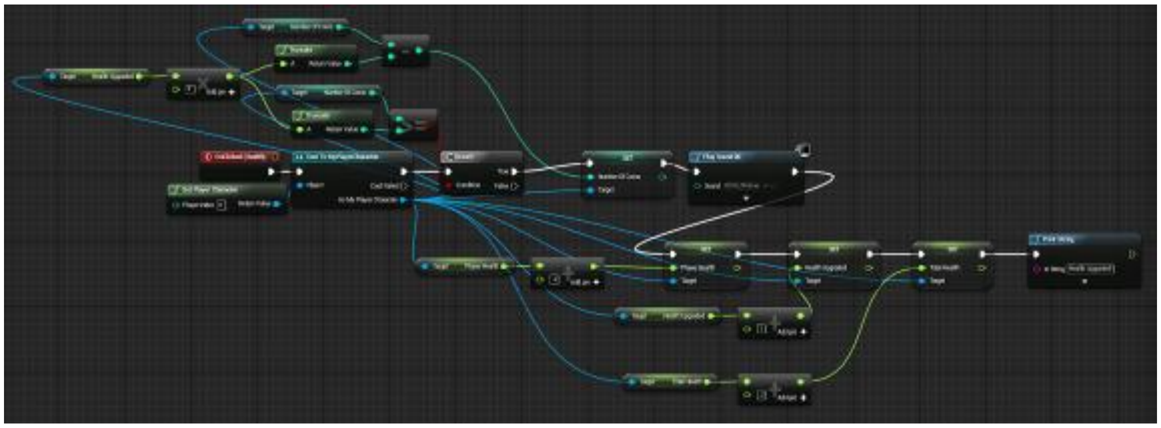
This function is activated as soon as the player chooses a difficulty level to play with, firstly a reference to the player character is needed in order to gain access to the variables required. After a reference to the player character has been successful we then begin to adjust the chosen variables, after the variables have been changed the system is then un-paused and both the difficulty level menu and the cursor are removed from the screen so the player can begin playing the game unimpeded. The player will be given the chance to alter the difficulty level of the game whenever they die and are forced to restart from the beginning. This function will be explained in greater detail in the "Difficulty Level" part of the Graphical User Interface section of this document.

3.11 Character upgrades

The player has the option to upgrade their character at all time by pressing "I" during game play. This will pause the game and the player will be brought to the "Shoppe" page that will display how much loot the player currently has and also all of the potential upgrades and costs of them upgrades available.

There are three upgrades available to the player. Health upgrade, Damage upgrade, and Ability upgrade. Once the player has clicked an option the system will check to see if they have acquired enough loot to make the purchase, if true and the player has enough loot then a coin sound will play and the player's health for instance, will be increased by 20 points, the variable "Health Upgraded" will be increased by one (this variable serves to keep count of how many times the player has upgraded their health) and a string will be printed to the screen that reads "Health Upgraded" to provide confirmation that the player has made the purchase. The player's loot will decrease by the cost of the upgraded and the player will then be able to purchases any other upgrades they can afford.

Below is a screenshot of the blueprints involved with upgrading the player's health;



The "Health Upgraded" variable is used to increase the upgrade cost in order to scale it with each purchase making it more expensive each time a purchase is made. The cost of the upgrade is determined by this formula; ("Health Upgraded") * 5, so each time a purchase is made and "Health Upgraded" is increased by one the total price of that upgraded to be purchased again increases by 5 coins. The same principals are put in place for the other two upgrades available to the player. When the player upgrades their damage they will do 20% more damage to enemies and when the player chooses to upgrade their ability the cool down on their ability will go down by one second, the cool down of the ability is capped to stop upgrading when it is at 3 seconds to prevent the game from becoming to easy.

3.12 Soul Gate

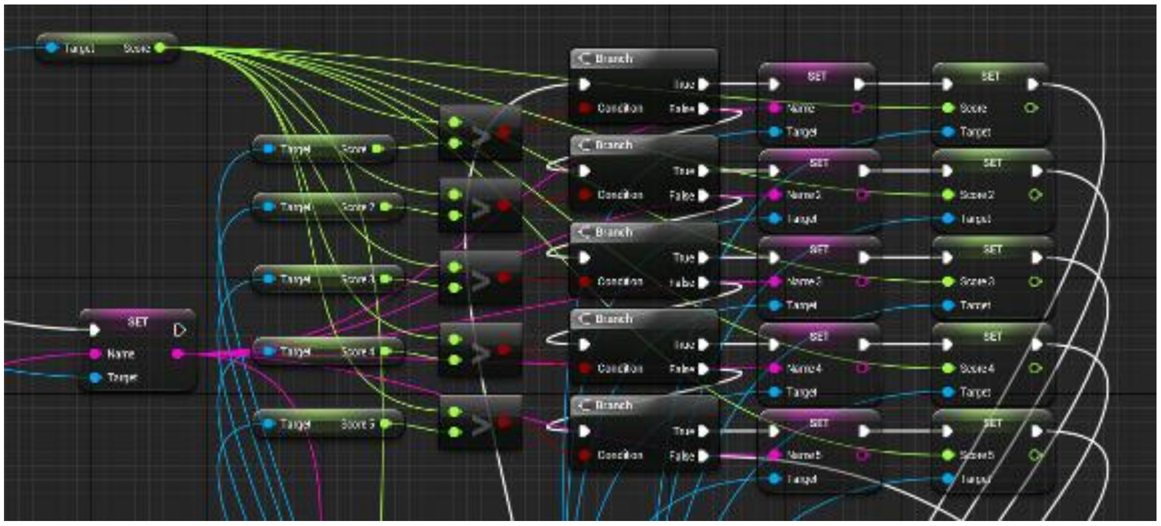
The only way to complete the game is to defeat the boss enemy however the game would be far too easy if the player did not need to carry out any activities before being allowed to face the boss, therefore I added in a feature called the "Soul Gate" which blocks off the boss area. Once the player approaches the soul gate a large message will appear on the screen saying "You need 10 souls to pass through the soul gate", the player will then need to go off and defeat 10 enemies in order to advance. The amount of souls that the player has collected will be visible in the top right hand corner of the screen at all times to let the player know how many more enemies they will need to defeat.

Once the player approaches the soul gate after acquiring the appropriate amount of souls a message will appear on the screen that says "Souls Accepted" and the soul gate will open allowing the player to progress. When the player enters the boss are the boss enemy will spawn and the player will need to defeat the boss enemy to complete the game and set a score.

3.13 Save Score / Load Score

Once the player has defeated the boss enemy and completed the game they will automatically be brought to a screen that will calculate their score and ask them to enter their name. The player's score is determined by using the following formula; ((Total amount of coins collected) + (Number of enemies killed * 2) * Difficulty Multiplier) * 100. So if the player has collected 20 coins throughout the play through, killed 3 enemies and was playing on medium difficulty then the formula would look like this; (20 + (3 * 2) * 1.5) * 100 = 3900. That score along with the name that the player enters will be saved to the game files and displayed whenever any person playing the game views the leader board.

The procedure to save the score and name of the player is a very complicated one. First I needed to force the game to check if a save file is present if there was a save file present that file is then loaded to the game, if however there is no save game present I would need to create a new Save Game Object in which to store the variables. Before anything could be saved I first need to create a save game blueprint, it is in this blueprint that I declared all of the variables that would need to be saved. Seeing as I only list the 5 highest scores on the leader board page I needed to declare 10 variables, 5 names and 5 scores. The game would then take the score that the player has set and determine where on the leader board it should be placed (in spot 1, 2, 3, 4, or 5) according to the blueprints below;



By comparing the newly set score to each of the existing scores on record it is easy to determine where the new score should be placed, if the new score is higher than the old score in spot 1 then spot 1 is replaced to display the new score and the new player's name, if it is not higher than the old score in spot 1 it will then be compared to the old score is spot 2 and so on until either an old spot is replaced and displays the new score and name or until the game has determined that the new score is not high enough to be displayed on the leader board.

After all the scores have been put in their correct spot all of the scores and associated names are then saved to the game files to be loaded when the process begins again.

The leader board can be viewed either by completing the game and entering your name or through the main menu in the "options" page.

The implementation of my project is continued in the Graphical User Interface Section as most of the menus and GUIs in my game have functionality behind them and therefore it would make sense to talk about both the aesthetics of these GUIs and their functionality simultaneously.

4 Testing

The project has been constantly tested throughout its development. With each function created I thoroughly checked to see if it was functioning properly before moving onto a new function. The testing initial began with white-box testing as I would critically evaluate the code behind the functions created to not only see if they would function but also to see if they could be improved upon.

There were various parts to my project that required repeated testing, one of which was the saving of the player's score at the end of the game. Once the game saves the user's score it is sent to a file within the game files called Score.sav, the way to refresh the scores to their default value is to delete this file. Each time a score was set I needed to check to see if the game was actually saving properly, to do this I created a leader board page within the pause menu for easy access and to prevent the need to complete the game to view the current leader board, once the scores were being saved I needed to complete the game a number of times getting various different scores in order to ensure that the leader board was returning the scores in the correct order. Many alterations were made to the save feature's code throughout the testing period as I found that it was not working as intended. Initially the save feature was only saving the most recent score and discarding any scores saved beforehand, on evaluation of the code I found that it was a problem with the Save State's variables as it could not distinguish between the new score and an old score, by creating more variables for the save state I was able to combat this problem.

The easiest way on ensuring a function is working properly is to print a string to the screen whenever a process is completed. This was extremely useful in the character upgrades section of the game. When I first began to develop the character upgrades function of the game there was no indication the exact value of the player's health, damage or the cool down of their ability, this made testing the character upgrades very difficult as I had no way of knowing whether they were functioning or not. In an effort to make testing simpler I added the actual health value to the game play HUD widget and also added just how long the cool down of the player's ability will take once they have shot their gun. With these new changes it was easy to see that the health and ability upgrades were functioning as intended, however the damage stat was not as easily displayed because it is a hidden variable that is not displayed to the player; instead during testing I forced the blueprints to print a string of the player's damage stat to the screen and then to keep printing it whenever the damage stat is upgraded by the player, for testing purposes each upgrade was set to cost no coins at all as it allowed me to quickly test each upgrade. Once the damage stat was visible and being upgraded with each purchase I removed the printed string command from the upgrade blueprints and moved onto another section of the game to develop.

The main aspect of the testing carried out during development was to ensure that the player was presented with enough tools to actually complete the game. In order to complete the game there must be a minimum of 10 enemies present on the map at any time regardless of difficulty level; they must also be able to acquire enough coins to adequately upgrade their character. This was carried out by both me and outside testers who attempted to play through the game without any background knowledge of the game.

I tested the game myself by playing through it from start to finish many times in a bid to find any problems or issues present in the code and or functionality of the project, while my testing was very useful and had a very positive impact on the outcome of my project, I realise that I myself could not carry out all of the testing required for this project to be a success. Therefore I carried out some customer testing below using subjects that had no understanding of the inner workings of my project. This black-box method of testing is very useful to find errors in the functionality of the project as the testers do not have a pre-emptive idea of what should occur when they fulfil a certain task and therefore can express how the project either met or did not meet their expectations and standards.

4.1 Customer testing

Customer Testing is a vitally important aspect of every game as it gives an accurately depiction of what is expected by the people who will be playing your game in terms of both visuals and functionality. I found 10 individual testers with no background knowledge of the game's code to take part in a short play through of my project and to then fill out a small survey to express their feelings of what areas of the project need improvement and what areas they felt were successes. The results of this testing is evaluated below with all of the testers remaining anonymous.

10 individuals carried out the testing of my project and all filled out a short survey which is included in the appendix section below under *"Other Materials"*.

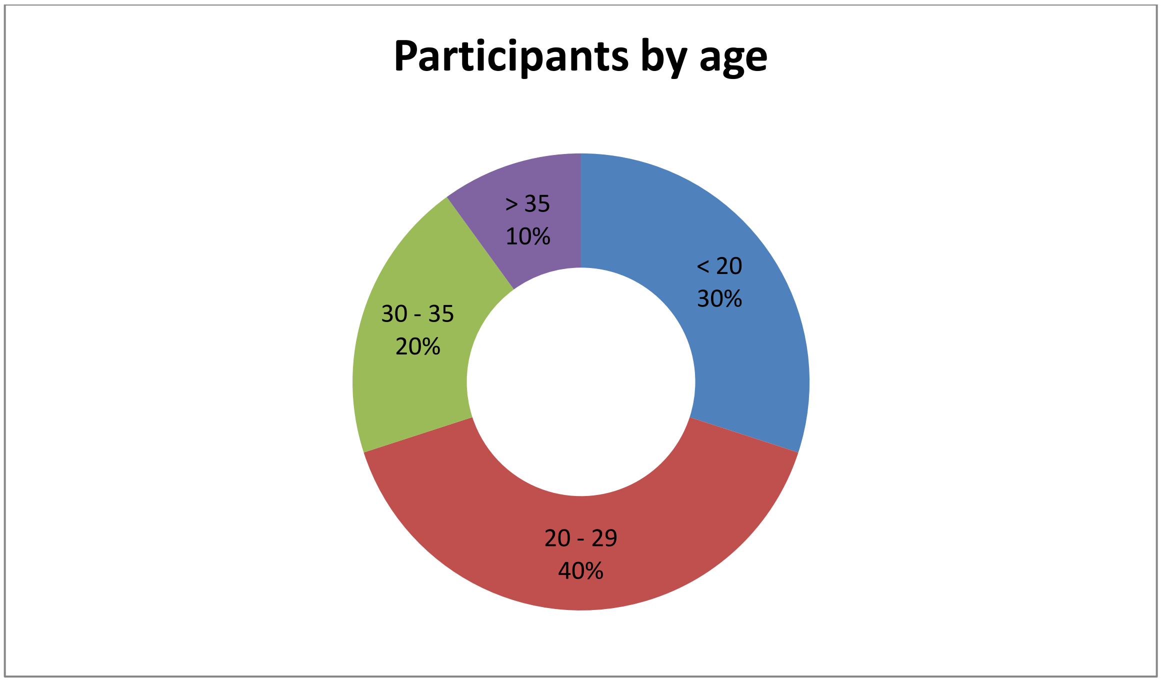
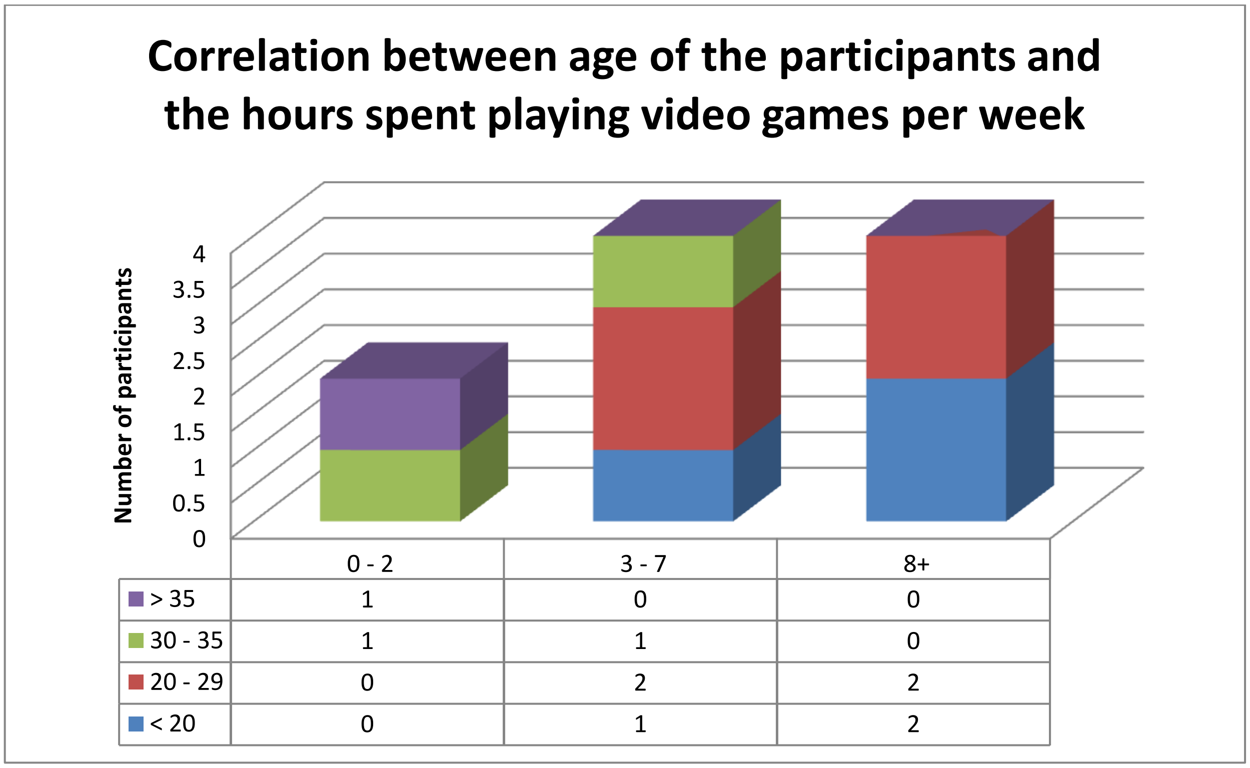


Figure 1, Chart of participants sorted in terms of age

There were four options for the participants to select the age group that they belong to on the survey, they were; below 20, 20 - 29, 30 - 35 and above 35. Of the 10 participants there was 1 person above 35, 2 people between the ages of 30 and 35, 4 people between the ages of 20 and 29, and finally there were 3 people under the age of 20 as illustrated by the chart in figure 1 above.

Each of the participants were then asked how many hours a week they spend playing video games, as expected there was a wide variety of results from each age group with some people playing 0 - 2 hours of games a week and others claiming that they spend more than 8 hours playing video games each week.



	0 - 2	3 - 7	8+
🟪 > 35	1	0	0
🟩 30 - 35	1	1	0
🟥 20 - 29	0	2	2
🟦 < 20	0	1	2

Figure 2, Participants age / Hours spent playing games

The data found in figure 2 above shows that younger audiences are more likely to play video games in their free time than older participants. It was also evident in the survey results that younger participants had an easier time coming to grips with the controls of the game, however this could be down to the participants prior experience with games rather than down to the actual game tutorial itself, this theory was reinforced while evaluating the survey results as the only people who had trouble coming to terms with the games controls were people who answered that they played between zero and 2 hours of video games per week.

One of the questions present on the survey handed to testers after they had played the game read; "Did you find any bugs or errors during your play through of the game? If so what were they?" out of all the 10 participants who took part in the testing of my project, one of them found an error. This particular person was in the 20-29 age group and played between 3 and 7 hours of games per week.

The error that they found was that there was a certain area of the level where they could fall into the water without dying. This was a big issue as once the player has fallen into the water there is no way to get back out because they are supposed to die and be greeted with a death screen. In this instance it was an easy fix, the trigger-box that was placed beneath the surface of the water did not stretch far enough to cover the entire map, this is most likely due to me adding areas to the map and forgetting to enlarge the hidden trigger-box that tells the player they have died. Once I had read the results of the surveys and this error was brought to my attention I quickly altered the trigger-box and tested it myself to ensure it was functioning properly and that the player could not fall below the surface of the water without dying.

Some of the participants added suggestions to their survey as to how I could improve upon the game in the future. Some participants suggested adding another level to the game that maybe would look different to the level being used at the moment, this could easily be done as all of the functionality behind the new level would have already been created, the only reason there is not another level in the game is due to the time restraints in place for the final year project. One other suggestion was the addition of a second ability which, much like the first suggestion, is not a matter of functionality but more a matter of time. The functionality behind a second ability is already present within the first ability but seeing as I wanted to demonstrate many different functions in my project I saw it more beneficial to spend my time on new functions rather than spend it on the same function a number of times.

The customer testing carried out on this project thought me a lot of things about my game that I did not notice. It showed me how different people choose different paths when faced with the same challenge and that a great game tends to these needs by providing different ways to fulfil each task. This is an area that I will look into in greater detail later on in the "further development" section of this report.

5 Graphical User Interface (GUI) Layout

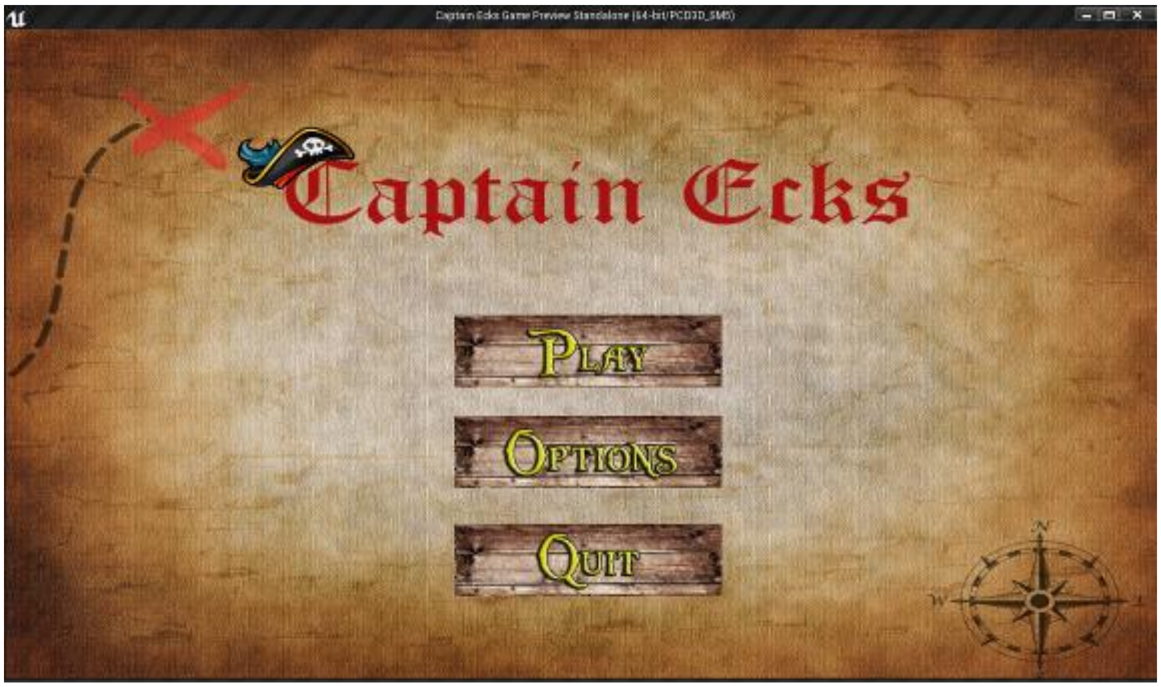
There are 12 screens in Captain Ecks all interconnected to provide a smooth and easily traversed graphical user interface. I will be discussing each of these 12 screens in detail below;

- Start Menu
- Options Menu
- Resolution
- Tutorial Option
- Tutorial HUD
- Gameplay HUD
- Difficulty Level
- Pause Menu
- Shop
- Death Screen (Shark/Enemy)
- Score
- Leaderboard
- Quit Menu

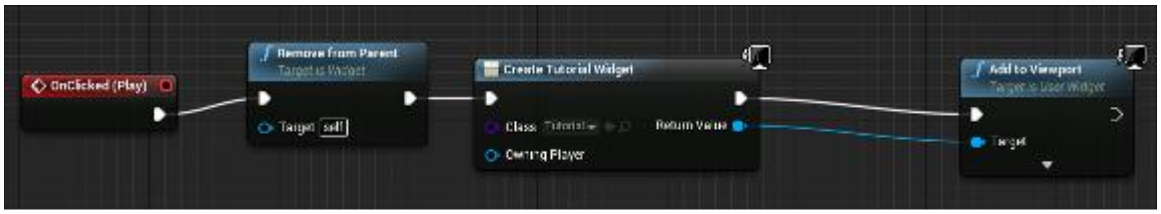
Each menu was created via a drag and drop technique that allows the programmer to add things such as buttons, images, and text to a widget. All of the background images and buttons present on these menus were created by myself through photoshop and sound clips were editted using audacity. Unreal engine allows you to add functionality behind menus through blueprints much like the other functions of my game explained in the implementation section above. Below I will go into detail of what each GUI does and how I implemented the functionality behind them.

5.1 Start Menu

The start menu will be presented once the player has turned the game on, it is the very first menu the user will see.



The start menu displays the title of the game and provides the player with three options, Play, Options, and Quit. If the player chooses the "Play" button then they will be redirected to the "Tutorial Option" page which will ask the player whether or not they would like to play the tutorial level before beginning the main game. Below is the small sample of code that deals with redirecting the player to a different widget (Menu);

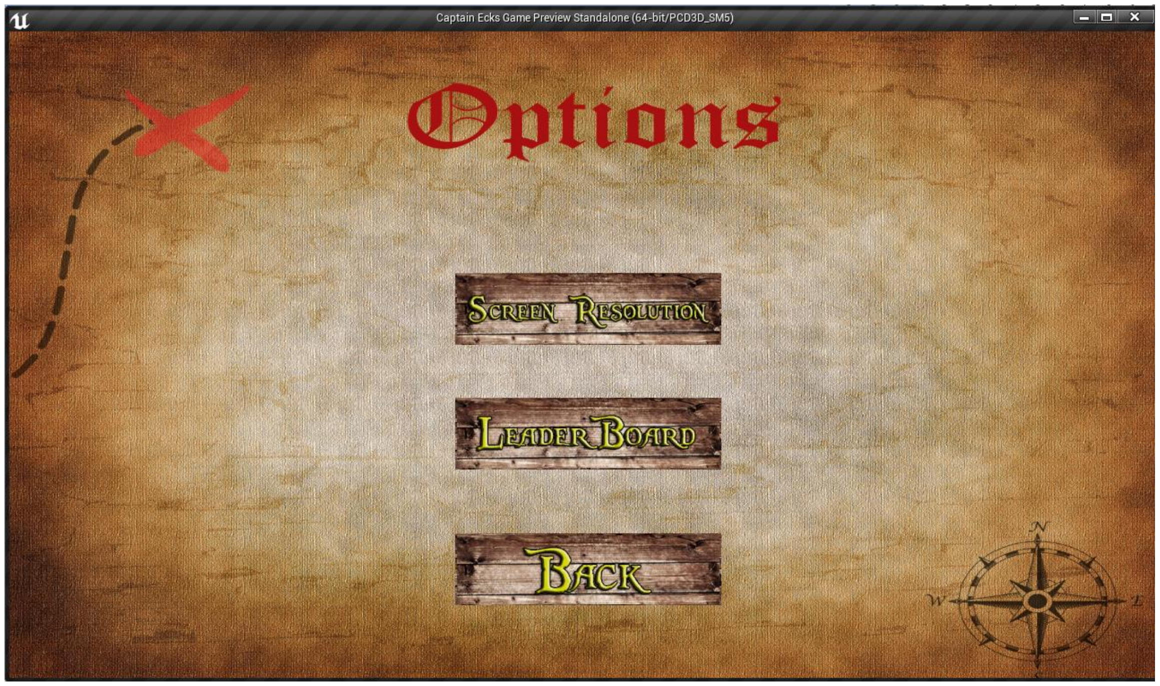


Once the player has clicked the "Play" button, the current widget is removed from the screen and replaced with a different widget. This simple method of switching widgets is used a lot through out the menus of my game which is why they are so easily navigated by the player.

The second button acts in exactly the same way as the first, once clicked the player will be redirected to the "Options" page, and finally the "Quit" button will redirect the player to the "Quit Menu", all of these other widgets will be discussed in detail within this section.

5.2 Options Menu

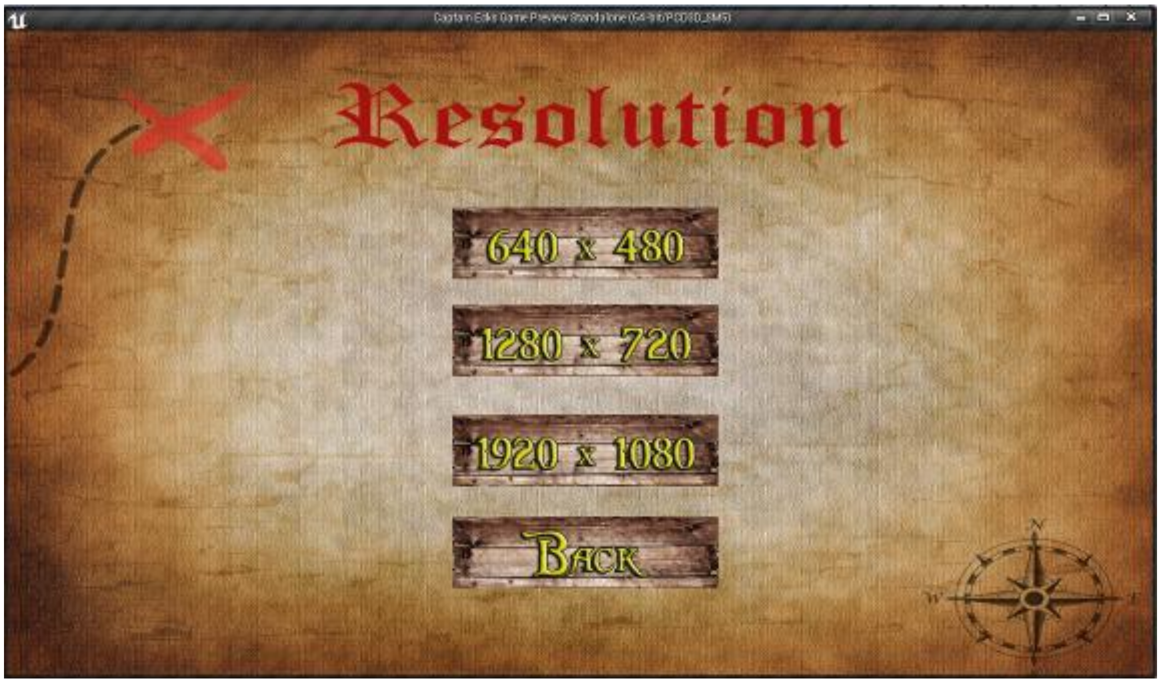
The options menu will be present in both the start and pause menus. The options menu acts as a bridge menu between the start menu and the option that the player would like to see, allowing them to choose to see the "Screen Resolution" page in which they can alter the screen resolution that the game is played in, or they can access the leaderboard page and view who is top of the leaderboard at present and what score they would need to beat in order to become "Captain".



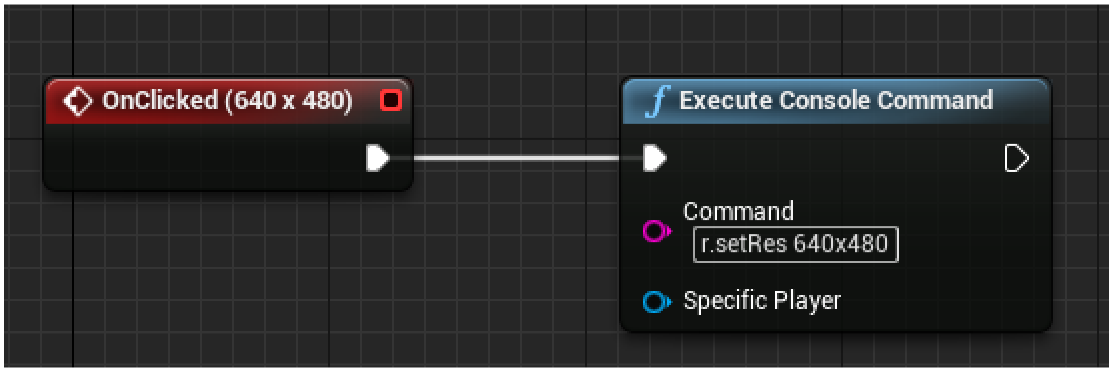
The final option of the "Options" menu is to return to the start menu for the player to either begin the game or exit the game.

5.3 Resolution

Once the player has reached the resolution page they will be given a choice of 3 resolutions in which to display the game. The three most commonly used resolutions for games are, 640 x 480, 1280 x 720, and 1920 x 1080.



The way that the game changes the screens resolution is through a console command in the widgets blueprints.

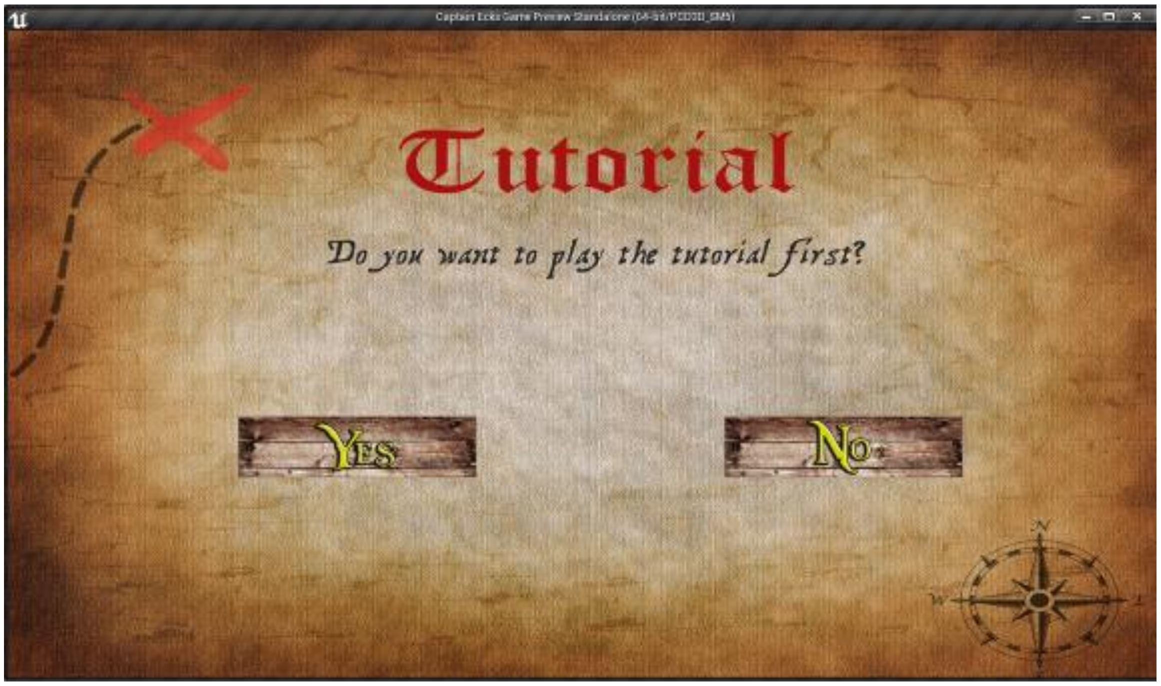


r.setRes 640x480

The command to be executed is as follows; "r.setRes 640x480". Once the system recognises this command the screen is then shrunk or enlarged according to the resolution entered and the player can then exit the reolution page and return to the "Options" menu.

5.4 Tutorial Option

Once the player has selected the "Play" button from the start menu they will be brought to the "Tutorial Option" page, it is from this page that the player will make the decision of whether or not to play the tutorial or skip the tutorial and progress straight into the main game.

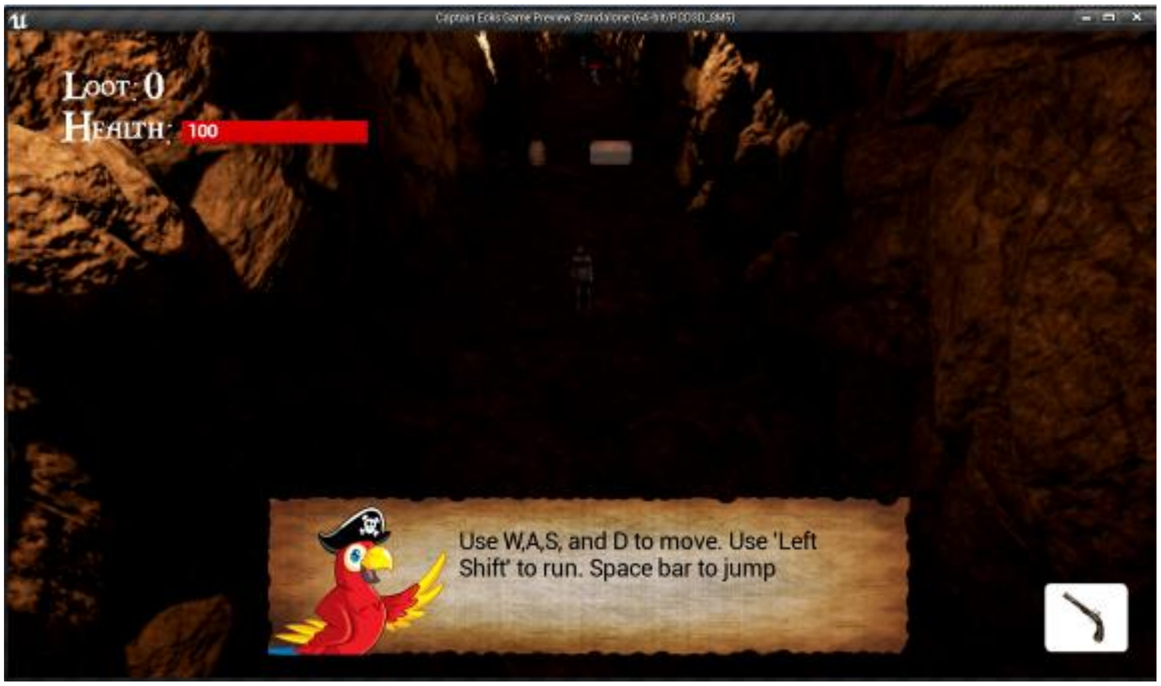


The functionality behind these two buttons are very similar, whit one important difference. If the player chooses yes that they do wish to play the tutorial then the widget will be removed from the screen and the tutorial level will load. If the player chooses no then the same functionality will commence, removing the widget from the screen except this time the main level will load instead of the tutorial level.

5.5 Tutorial HUD

The term HUD stands for Heads Up Display, and it comprises of all of the elements shown on the screen as the player is playing the game such as, health, abilities, loot, etc. In this game there are two different versions of HUDs, the tutorial HUD and the Gameplay HUD.

The tutorial HUD as shown below illustrates to the player various elements including, loot amount, health stat, the player's ability, and the in game message prompt that will guide the player through the tutorial.


Captain Eoks Game Preview Standalone (64-bit/PCD3D_SM5)
Loot: 0
Health: 100
Use W,A,S, and D to move. Use 'Left Shift' to run. Space bar to jump

The message on the prompt is activated as the user steps into what is called a triggerbox. Triggerboxes can be programmes to carry out a function either when a player has entered the triggerbox or when a player has exited a triggerbox. In the tutorial level the messages visible in the message box are changed as the player progresses from one invisible triggerbox to the next. To do this I made the HUD widget constantly call for the message box area to be populated by a variable called "Tutorial Text" from within the player character. I then set the value of the variable in the level blueprints depending on which triggerbox the playe last entered, Which forces the GUI to update the message box each time the player enters a new triggerbox which allows me to provide the player with relevant step by step instructions on how to progress the level when they are needed.

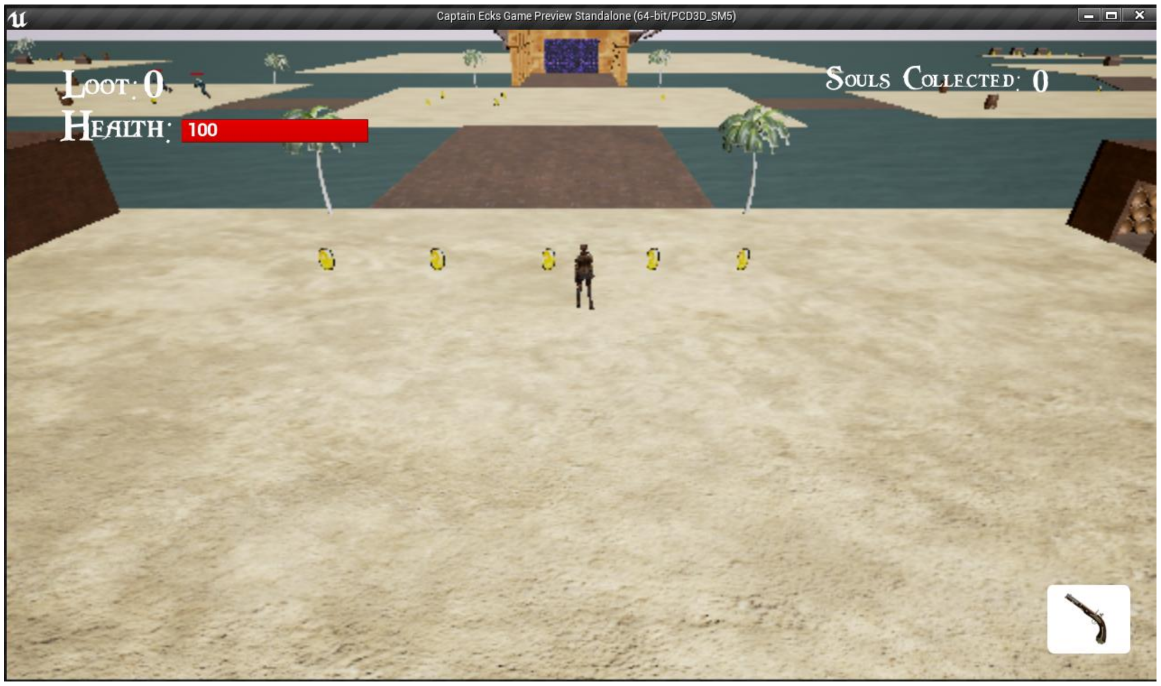
Seeing as the Health of the player cannot go down during the tutorial level I will speak about that in the next section, Gameplay HUD.

For the loot value on the HUD I set a binding to a simple text box on the GUI, a binding allows you to apply functionality to an object on a GUI such as a text box or a progress bar. The binding for the loot text box constantly changes to reflect the current amount of coins that the player has and to do this I called upon the player character in order to access it's variables and then requested the "Number Of Coins" variable and then I appended that integer variable into a text variable so it can be used as the value for the loot text box.

5.6 Gameplay HUD

During gameplay the HUD is very important, it must efficiently display all the neccessary information needed for the player to make educated decisions throughout the game. The gameplay HUD and the tutorial HUD share many characteristics as they stem from the same widget, however there are some slight changes that make them distinct from each other.

Once the game has been started it is now possible for the player to lose health and therefore this must be presented constanly throughout the game. To do this I called for the health value of the player character and set that as the percentage to fill the progress bar being used as the health bar.



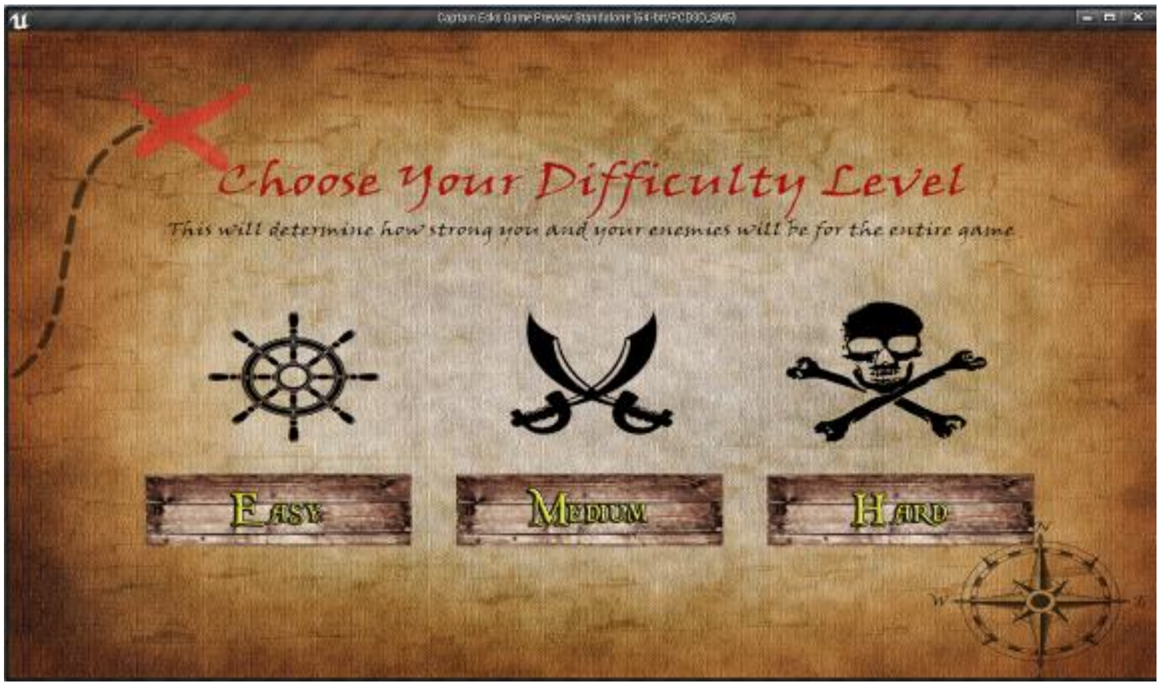
In the bottom right hand corner you can see the player's ability, this ability is discussed in detail in the implementation section of this document.

The tutorial message box present in the tutorial HUD has been removed once the main game begins and has been replaced with the "Souls Collected" statistic found in the upper right hand corner of the screen. This stat is important as the player is required to collect 10 souls before they can face the boss enemy. I began creating this function by first seeing if the player was in the tutorial level or the main game level, and if they were not in the tutorial level then the stat would be shown on the HUD. The stat used to populate the text box is the "Enemies Killed" variable from the player character, this variable is incremented by 1 each time the player kills an enemy and the text box updates itself accordingly.

The final thing to mention about the gameplay HUD is what happens when you approach the soulgate. When the player approaches the soul gate one of two messages will appear, if the player has acquired 10 or more souls by defeating enough enemies then the following message will appear; "Souls Accepted" and the door will open. If however the player has less than 10 souls when they approach the soul gate the the following message will appear "You need 10 souls to progress through the soul gate" and the soul gate will remain closed. To do this I used the same variable as before in the "Souls Collected" section. The "Enemies Killed" variable is put through a branch and if it's value is more than 9 than the door will be opend and if not the door will stay closed.

5.7 Difficulty Level

Once the player has begun the main game they will be presented with an important choice. That choice will be what difficulty level they wish to play the game on.


Choose Your Difficulty Level
This will determine how strong you and your enemies will be for the entire game
Easy
Medium
Hard

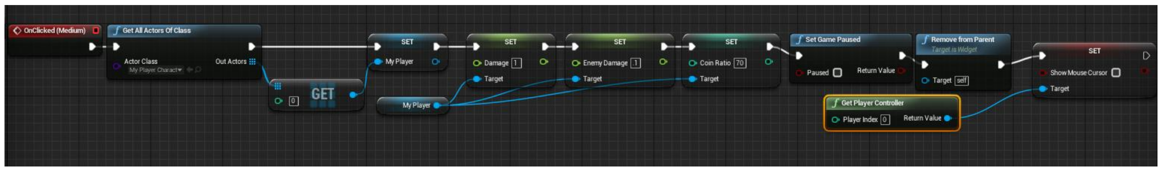
The difficulty levels are as follows;

Easy: Enemies have very little health and low damage, the score multiplier will be set to 1 and the ratio at which coins will spawn from loot sources is set to high with a very small chance of spawning an enemy.

Medium: Enemies will take a little more effort to kill than they would on easy mode. There will be a 30 percent chance of enemies spawning from loot sources and the player will be dealt more damage from enemy attacks.

Hard: Enemies will be very difficult to defeat. There will be a 50/50 chance of spawning an enemy through loot sources and the player will need to be very careful when fighting enemies as they will be able to defeat the player with only a few attacks.

Below is an example of the code that was used to set all of these variables once the player has selected a difficulty level;

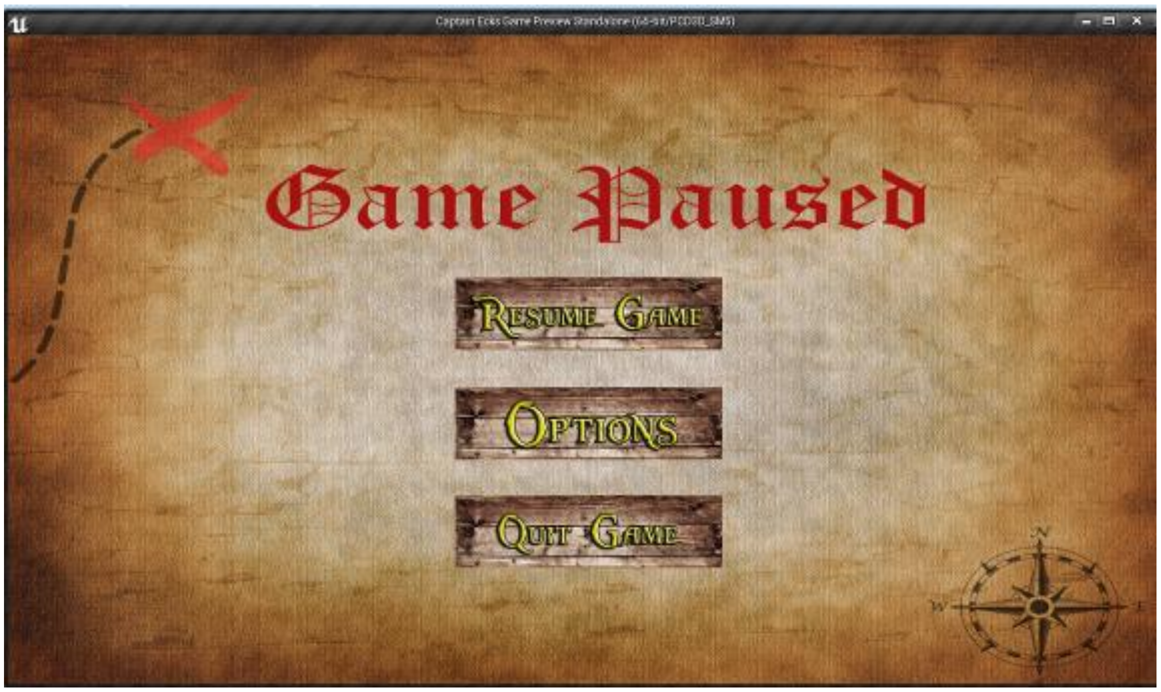


Depending on which button the player clicks the player's damage, the enemy's damage, the score multiplier, and the coin ratio will be altered accordingly and the difficulty level widget will be removed from the screen so gameplay can commence.

The player will be given the option to change the difficulty level of the game whenever they die and respawn.

5.8 Pause Menu

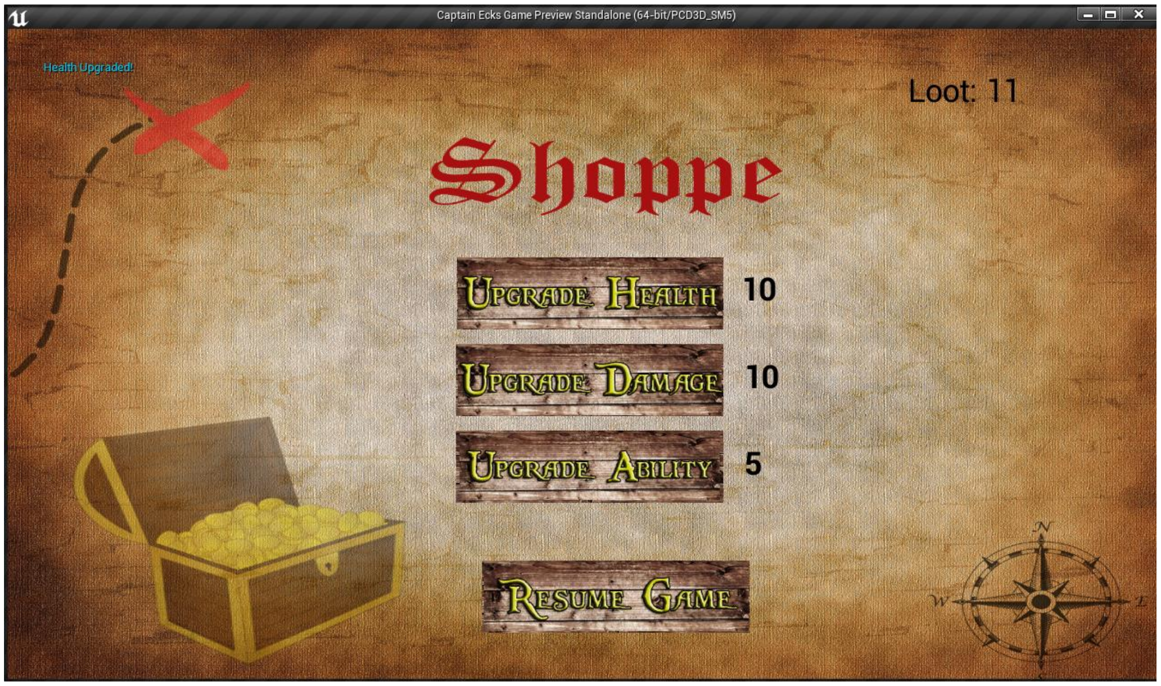
The pause menu can be accessed at any point from within the game when the player clicks the left shift button



From the pause menu the player will be able to resume the game from where they paused, adjust options through the previous "options" menu, and they will also be able to quit the game whenever they wish. A puase menu is vital in any game as it allows the player to take a break from playing and fulfill other tasks.

5.9 Shop

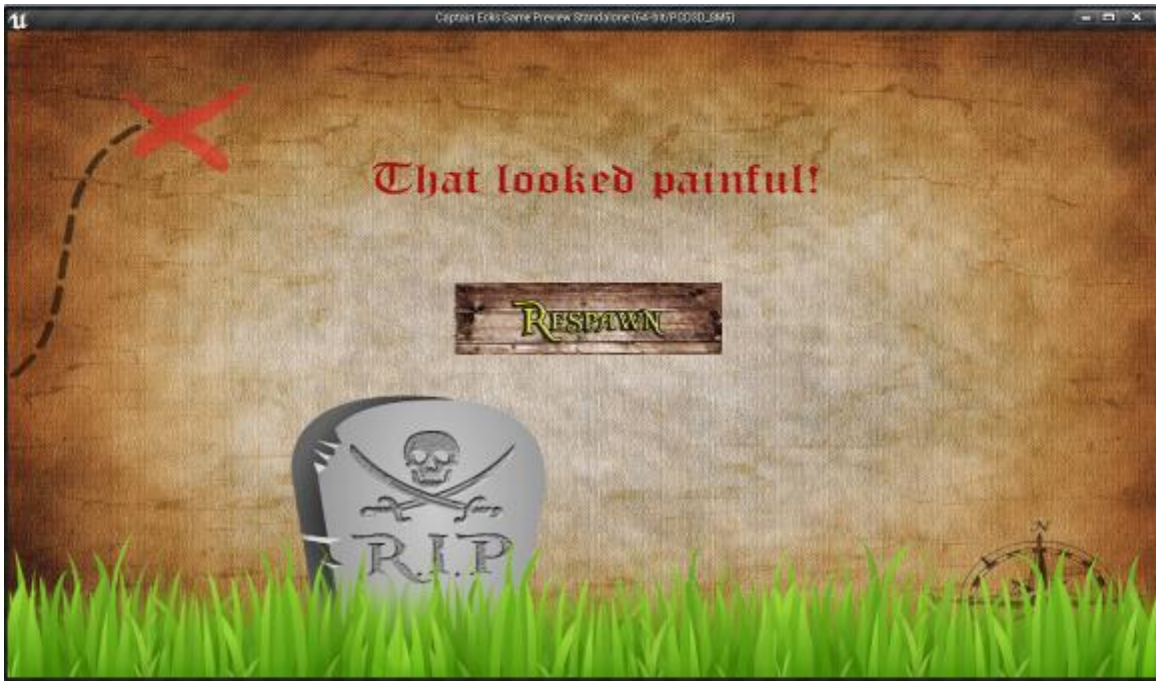
The shop is where the player will be able to purchase upgrades for their character. The shop page can be accessed at any time by pressing "I" during gameplay and the player will be greeted with the following widget;



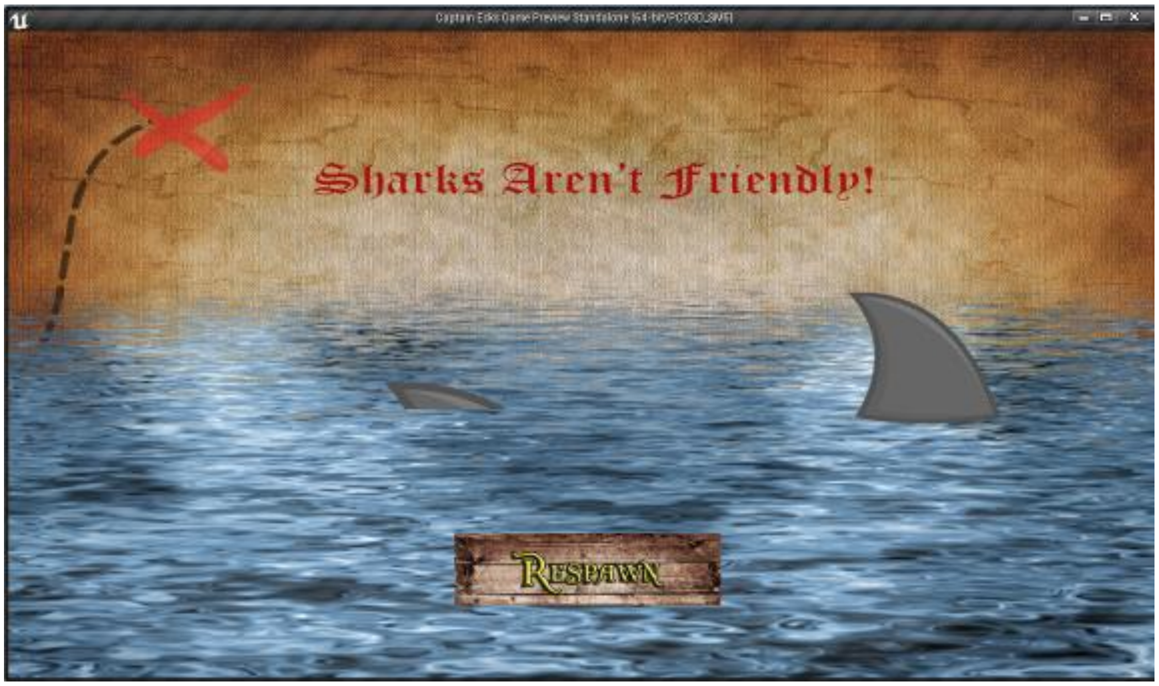
The shop shows the three possible upgrades that the player can choose from, Health, Damage, and ability. As you can see by the top left corner of this screenshot the player has just purchased a health upgrade and the cost of the health upgrade has increased from 5 coins to 10 coins. The increase in cost of upgrades with each purchase ensure a balanced progression for the player's character throughout the game, requiring the player to collect an increasing amount of loot should they wish to continue investing in the same upgrade. The functionality behind the upgrades on this page are explained in the implementation section of this document above.

5.10 Death Screen (Shark/Enemy)

There are two different ways in which the player can die in the game. The first and most common way is for the player to take too much damage from the enemy characters and for their health to fall below zero. When this happens the player will be greeted with the following death screen;



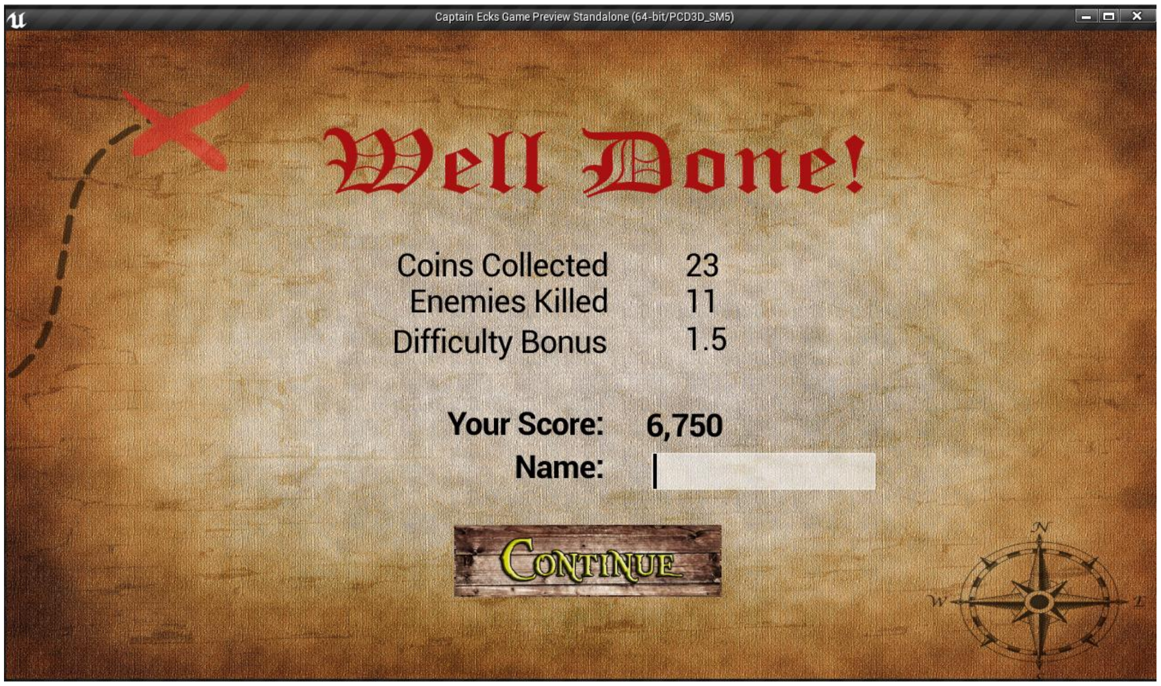
The second way in which the player can die while playing the game is by jumping into the shark infested water surrounding the islands upon which the game is played. Should the player fall into the water a short "Splash" sound will play and the following widget will appear on the screen;



As you can see both screens share one thing in common, they both have a single "Respawn" button present. Once the player clicks the respawn button on either one of these death screens they will be redirected to the "Difficulty level" page and they will be forced to begin the game again from the beginning. This added consequence of having to start the game again upon death is a design choice as it forces the player to think a little bit more about what they are doing and also add some tension to fights knowing that they will have an bigger overall impact rather than just dying and respawning without losing anything at all.

5.11 Score

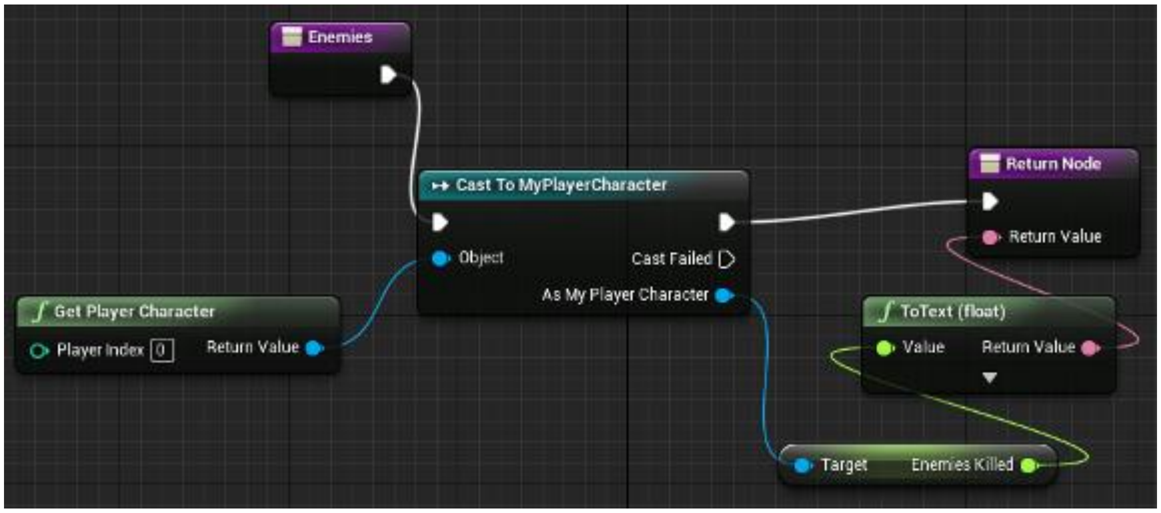
Once the player has acquired enough souls to face and kill the boss they will have completed the game, and it is upon completing the game that the player will be asked to enter their name. It is also on this screen that the player's score will be calculated. The "Score" page looks as follows;



As you can see the page displays all of the various statistics involved with calculating the player's score and also includes a text box that allows the player to enter their name.

The functionality behind the score calculation has been mention in the implementation section of this document, however, the way in which the stats are displayed and the way in which the player enters their name are not.

To display each of the stats I used a node called "Cast To MyPlayerCharacter" which allows me to access the variables of the player character. I then added a number of text boxes to the widget and binded them to hold a variable value instead of a set value like below;

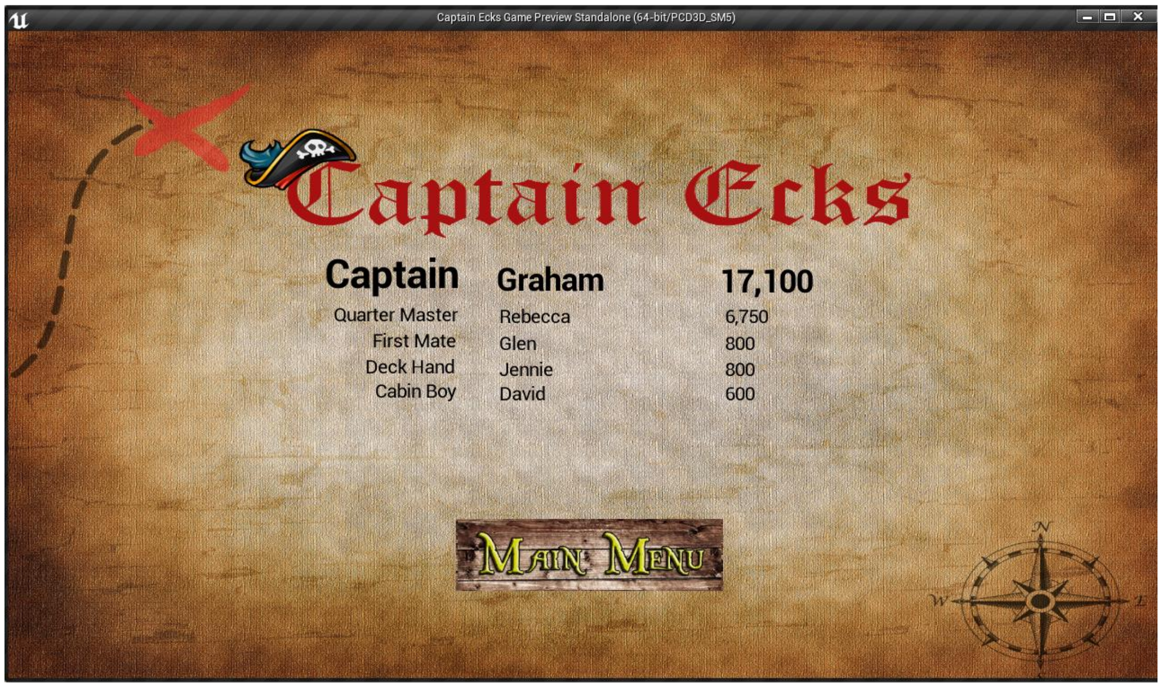


Here you can see that I am calling the "Enemies Killed" variable from the player character blueprint and appended it to a string in order to be used for the return node. This process was repeated with different variables for the other statistics present on the page.

The final element of this page was the text box in which the player will enter their name, I chose to limit the amount of character that the player can enter to 10 characters. This choice was in order to prevent the player's entry to distort the other elements on the page. Once the player presses "Continue" both the name of the player and the score that they set will be saved to the game files as mentioned in the implementation section above.

5.12 Leaderboard

Once the player has entered their name on the "Score" page and pressed the "Continue" button they will be brought to the leaderboard page below;



Captain	Graham	17,100
Quarter Master	Rebecca	6,750
First Mate	Glen	800
Deck Hand	Jennie	800
Cabin Boy	David	600

It is in this page that I am loading the stored scores and names from within the game files. The names and scores are sorted in descending order in terms of the score set by the players. The page shows 5 ranks;

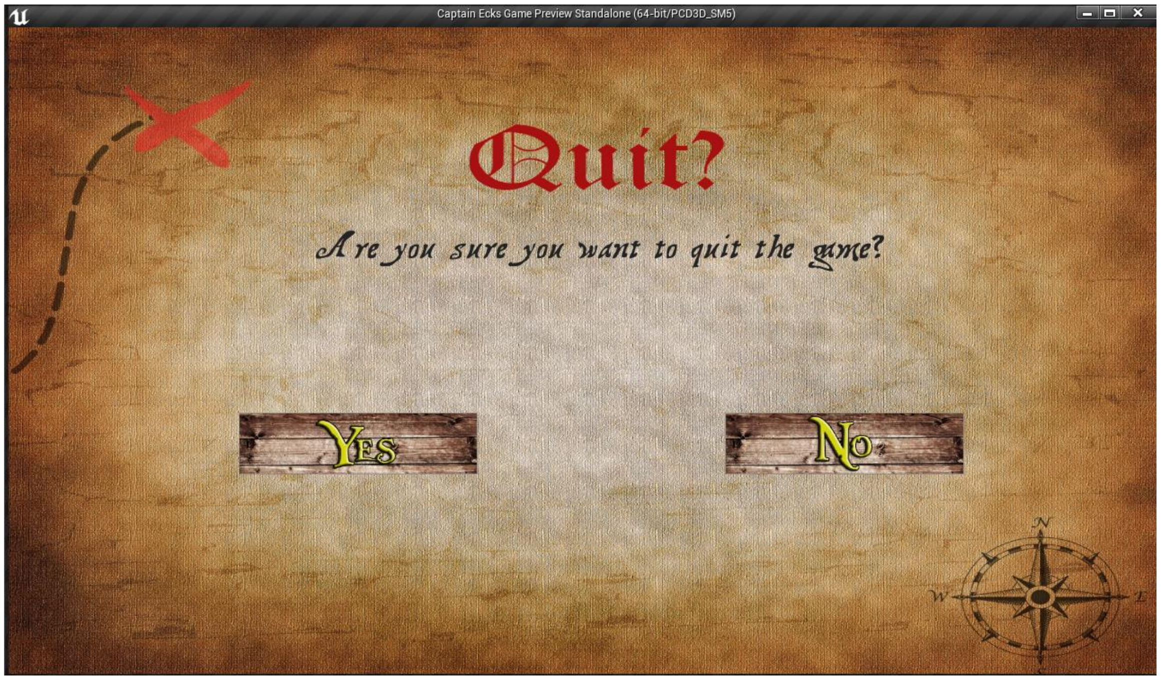
1. Captain
2. Quarter Master
3. First Mate
4. Deck Hand
5. Cabin Boy

It is the incentive to become captain that drives the players to replay the game once they have already completed it.

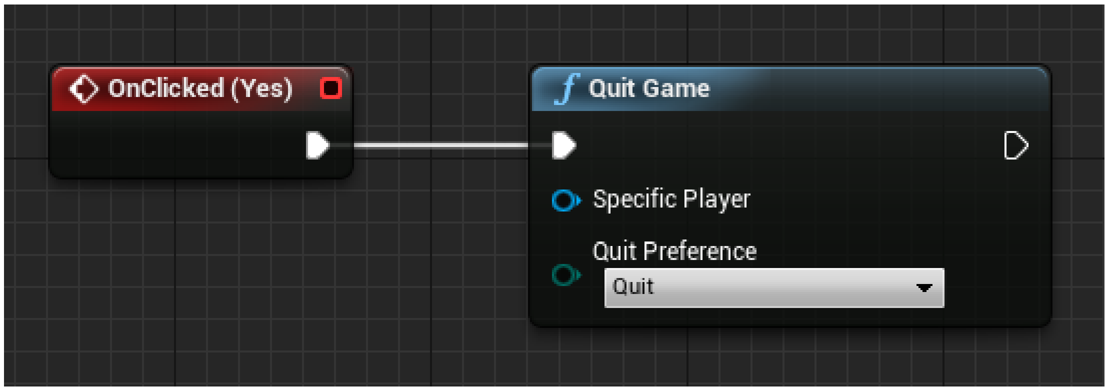
Once the player has taken the time to review the scores and where they are in the ranks they can then return to the main menu of the game where they will be met with the "Start Menu" page.

5.13 Quit Menu

When the player chooses to quit the game they will be met with the following screen asking them "Are you sure you want to quit the game?" and will be provided with two options, yes, or no.


Captain Ecks Game Preview Standalone (64-bit/PCD3D_SM5)
Quit?
Are you sure you want to quit the game?
Yes
No

If the player selects "No" then they will be returned to the previous menu from which they selected the quit option. If the player selects "Yes" then the game will close via the following simple function;



Quit Preference: Quit

Once the button is clicked the client will quit the game through the "Quit Game" node and the game window will close returning the player to their desktop.

6 Evaluation

After completing my project I compared it to the initial expectations I had going into my final year. I feel as though I have accomplished all of the tasks that I set out to do from the beginning and in some areas I have surpassed them. By playing the finished game I can see that it is well balanced in terms of the amount of enemies on the map and also in terms of the amount of loot available to the player, this was a very important area of the project as the player must feel that they are capable of finishing the game regardless of how challenging they may find it. The system was evaluated by both me and outside testers as explained in the previous "Testing" section in this document. When I tested the game myself I looked to see if all of the functional requirements set out for the project were met to an acceptable standard. To do this I repeatedly carried out certain events from within the game in an effort to trigger these functions once I saw that they were functioning properly I would move onto the next one and continue until all of the functional requirements were tested and working. The game has accomplished everything that was expected and also accomplished things that were only added throughout its development, such as the enemies individual health bars, the soulgate, and many more aspects that were only thought of after the initial proposal had been submitted. There were 10 functional requirements that were listed at the beginning of this document, they were as follows;

- Start game
- Save Score
- Load Score
- Choose Difficulty
- Tutorial
- Pause/Options
- Exit Game
- Acquire Loot
- Fight Enemies

- Shop

Through my own testing of the project I have found that every one of these requirements have been successfully accomplished, combining to create a fully playable and immersive game that was met with positive feedback from the individual testers.

The game performs at a reasonable frame rate and graphic quality on my home computer, the details of which are below;

RAM: 6GB

Operating System: 64-bit

Graphics Card: NVIDIA GeForce GT 640

It is to be expected that the game will not run as smoothly on computers with lower specs than the ones provided above. I tested the game on my laptop during testing and found it to run at a less optimal pace due to the limited power and graphics capability of the device. The game itself reacts to this by lowering its frame rate; the user can also alter the resolution of the game in order for the game to be run smoother.

I evaluated my project in terms of player feedback and also in terms of goals accomplished. In these two areas I feel my project excels. I also have ideas on how my project could be improved upon should I choose to continue working on it past the submission deadline; these ideas will be discussed in greater detail later on in the "Further Development or Research" section of this report.

7 Conclusions

There was only one major disadvantage to the project, which was the time constraint; there were some things that I would have liked to add such as a second level or a new type of enemy but due to the limited amount of time I had to work on the project I elected to work on new functions instead. The game was built using unreal engine which is a very capable development tool that is used my many developers. Using unreal engine was a huge advantage for my project as it supports many file types making the importing of objects and textures much easier than it would be if I had chosen to develop my game using another development engine such as unity.

There were also areas of the project that evolved throughout development such as the games enemies which went from being simple shapes that could damage the player to becoming fully animated characters each with their own individual health bar and variables. Initially I intended to allow the player to save the amount of coins they had collected but after consideration and meeting with my supervisor I elected to transform the save feature into a score and leader board dynamic. Instead of the player choosing to save the amount of coins they collected, the game will now automatically save the score set by the player at the end of the game and then load that score when the player calls for the leader board to be displayed. This feature added a competitive aspect to the game which promotes replays while removing the ability to save coins forces the players to be more careful with the decisions they make while playing the game in order to prevent death and keep their coins safe. The playable character also evolved throughout development as I removed the idea of adding a stamina bar to the HUD which would limit the amount of time the player could run and instead opted to include an ability to the character which would also be presented on the HUD. This choice was made because I wanted to reserve the elements on the HUD for the vitally important statistics to the character to prevent HUD from overpowering the game and distracting the player from the game play.

The game has the potential to be successful and with all of the distribution methods available to amateur game developers today there are a few ways I could go about bringing my game to an actual audience.

8 Further development or research

This game can be expanded on over time, I would hope to eventually add many more types of enemies, for instance, a brute enemy class that has high health and high damage output but is very slow compared to other enemies and an assassin class which could have high damage and speed but very low health. New levels could also be added to the game which would evolve the game play further and provide a new attraction for existing players to revisit the game. There could also be many more abilities added to the game to give the player different ways to beat their opponents with new abilities and animations. I would also like to give the player a lot more customization options such as the clothes the captain wears or the type of sword she carries, there are many ways this system can be evolved and expanded on should circumstances demand.

Earlier I touched on the topic of creating different ways to accomplish the same task. In my own personal experience with video games this can often be the distinguishing factor between a good game and a great game. Players like to explore a game world and find new angles on a task that they feel may have never been seen before, by rewarding an inquisitive player with things such as hidden loot or a secret area then you will automatically promote the replaying of your game as old players will repeatedly play through the game to see what other secrets you may have hidden for them. Should I continue to develop my game further this will definitely be an area that I will look into and expand upon.

Now that the game is finished I could look toward actually publishing my game to a wider audience, this could be done by manufacturing physical copies of my game to be distributed in stores around the area, however this method requires a high initial cost and would therefore be outside of my reach unless I was to receive some outside investment. The only ways I could receive any investment to my game would be to present the game to an already established game company who could then front the price of manufacture for a percentage of the overall profit that the game would make or the simpler option would be to open a kickstarter page for the game. Kickstarter is a funding platform that is used worldwide, it allows both big corporations and individual developers to approach the public for funding and in return they may receive rewards such as access to the beta of the game or even a limited edition of the game itself depending on the amount that the investor has donated. Kickstarter is a very popular website and according to their own website; since their launch, on April 28, 2009, 11 million people have backed projects and 2.4 billion dollars has been pledged, culminating in 105,027 projects being successfully funded. As an individual developer this would most likely prove to be the best route to the investment needed to manufacture hard copies of my game.

Another method of distributing my game and the most popular method would be to distribute it digitally through one of the many online platforms that have been created for games around the world. We have entered a digital age in gaming with a vast majority of video game sales occurring through digital storefronts such as PlayStation Store, Xbox Live, and Steam. The benefit of digital games is that there are no manufacturing costs involved, the buyer simply selects the game from the virtual store, purchases it and the game will then automatically begin to download on their own device. Both PlayStation and Xbox endorse amateur developer by helping their games receive a wider audience but Steam has always been the easiest and preferred way to ensure your game meets a bigger market.

Steam has initiated a program that they call Steam Greenlight. Greenlight allows developers to create a store page for their game to be viewed on Steam itself by potential buyers, the buyers then vote on whether or not they want to see your game get made, if your game receives enough positive votes steam will then contact you and work together with you on a timeline for the completion and release of your game. Steam really encourages new developers to submit their game to greenlight which is why they have established a very positive reputation among the gaming community. Seeing as the game does not need to be fully completed for steam to accept it and for it to be put up on their greenlight page, this is the best opportunity for my game to be distributed and therefore would be the preferred method I would choose should I ever consider distributing my game.

9 References

Bibliography:

2016 (2004) *What is unreal engine 4*. Available at: https://www.unrealengine.com/ (Accessed: 9 May 2016). In-line Citation: (2016, 2004)

Kickstarter (2016) in *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Kickstarter (Accessed: 1 May 2016).

Video game industry (2016) in *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Video_game_industry (Accessed: 15 January 2016).

10 Appendix

10.1 Project Proposal

Project Proposal

Captain Ecks

Graham Robinson, x12486282, x12486282@student.ncirl.ie

BSc (Hons) in Computing

Gaming and Multimedia

28/09/2015

Objectives

The objective of my project is to create and develop a fully functional game based on combat and loot. The player will take control of a single character and will fight their way through hordes of enemies while simultaneously collecting loot to sell and acquire upgrades for their character.

Throughout gameplay the player can upgrade their weapons and armor to do more damage to enemies and to reduce damage taken respectively.

The main features of the game will be, full character control (walk, run, jump, attack, block etc.), enemies of varying difficulty (common enemies, bosses etc.), immersive pirate themed levels, and sound effects.

I will also be designing a poster and a short cut scene to promote the game.

The main objective of the player is to fight their way through the maze-like levels of increasing difficulty to get back to their ship, once back on their ship they will be able to select which level to travel to much like how the level system in super Mario or ray man works, once one level is completed the next level will be available to travel to.

The player will need to keep an eye on their health bar (how much damage they can take before dying), their stamina bar (how long they can run) and their cool-down times (some unlocked abilities can only be used every 30 seconds or so.

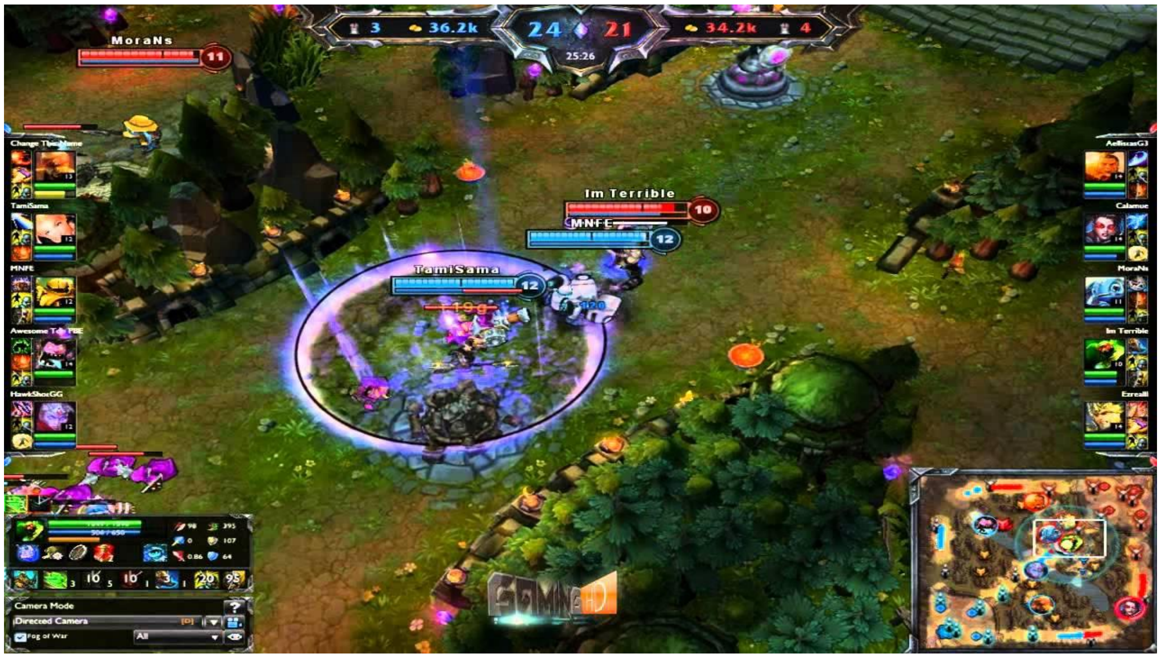
Background

The game industry has grown tremendously over the past 20 years, many computers owe their biggest breakthroughs to the gaming industry which provided graphics cards, faster CPUs, and 3D graphic accelerators.

The game industry now rivals the film industry in terms of both sales and profit and continues to smash records. The bestselling console of all time is the PlayStation 2 which sold over 155 million units since its release in the year 2000 while the bestselling video game of all time is Mine craft which has sold upwards of 70 million copies across many various platforms including playstation3, PlayStation 4, Xbox 360, Xbox one, android, IOS, PC, etc. these are only the top two example with other consoles and games selling just as impressively.

Video Games have grown in popularity so much that some games are now considered sports or “e-sports”. These games are usually competitive online multiplayer games such as league of legends or Dota 2, but single player games have also been included in the e-sports line up. These games are played by professionals in worldwide tournaments for incredible cash prizes with the fourth international Dota 2 championship in 2015 having a record breaking prize pool of $10.9 million. It is estimated that in 2013, 71,500,000 people watched competitive gaming world wide, even BBC televised the most recent League of legends world championship.

League of legends is a massively multiplayer online (MMO) game where two teams of 5 or 3 fight against each other in order to progress to the enemy nexus and destroy it. The game uses an isometric view as shown below;



The game I am creating is a third-person, isometric action game. Third- person video games have been very popular since the late 90’s with many playstation one games adopting the camera angle. Some of those games include, crash bandicoot and spyro. The isometric view that my game is undertaking is a sub-genre of the third person view that’s is rapidly growing in popularity with many indie games. Isometric games are very popular in both mainstream blockbuster games and games bought through app stores such as IOS and android allowing much more position awareness then the standard third person point of view on account of its zoomed out feel and diagonal point of view.

I have always had a keen interest in gaming ever since I played super mario on the N64. Since then games have evolved to a much more polished and realistic level, with some cinematics looking as lifelike as actors on a set. I have been playing video games for the most part of my life, about 15 years, and throughout that time I have played many games across a lot of consoles and I have played games of all types of genres including, first-person shooters, third-person, top-down, 2D side-scrollers, etc. I chose to do my project on this topic because not only is it something I have a huge interest in but I feel it could also evolve into a really good, interesting game.

Gaming has covered nearly every topic imaginable but I feel like there is relatively little content based around pirates in the medium, even less using the isometric camera method, therefore I feel there is a genuine gap in the market for a game like mine.

Technical Approach

Research:

I began my research by looking for games that use the isometric view I am using in my game and found the following two examples;

1. Dead Nation.

Dead Nation is a top-down *shoot 'em up* video game for the *PlayStation 3* developed by Finnish video game developer *Housemarque*. Dead Nation takes place in a fictional world afflicted by a *zombie apocalypse*. The player can play as a male or female character and fight different types of zombies. Players are awarded score multipliers and money when zombies are killed. Money is used to purchase and upgrade weapons at checkpoints. Each time players are hit, they lose health. Players fight their way through ten levels, using weapon shops that allow weapon upgrading and armor swapping. At times the players are trapped in areas where they must survive until they have accomplished a certain goal (e.g. wait for an elevator while fighting zombies or kill all zombies in the area (https://en.wikipedia.org/wiki/Dead_Nation)

2. Diablo III.

Much like in Diablo and Diablo II, equipment is randomized. In addition to base stats, higher-quality items have additional properties, such as extra damage, attribute bonuses, bonuses to critical hit chance, etc.

The proprietary engine incorporates Blizzard's custom in-house physics, and features destructible environments with an in-game damage effect. The developers sought to make the game run on a wide range of systems without requiring DirectX 10. Diablo III uses a custom 3D game engine in order to present an overhead view to the player, in a somewhat similar way to the isometric view used in previous games in the series. (https://en.wikipedia.org/wiki/Diablo_III)

My Research showed me that there were a very limited number of games following the same outline as mine, while I found many examples of isometric game, none had the same pirate them of mine and therefore I realised that this idea was unique and would be very interesting to develop further.

Requirements:

Operating System:	Windows 7/8 64-bit
Processor:	Quad-core Intel or AMD, 2.5 GHz or faster
Memory:	8 GB RAM
Video Card/DirectX Version:	DirectX 11 compatible graphics card

Implementation:

To begin my project I will create my character model and animations using blender by first creating the mold (shape) of my character, adding colours to the mesh, incorporate rigging to allow realistic movement and finally I will create some animations using my character model for actions such as walking and running.

After I have completed the creation of my character model I will export it to Unreal Engine which will be the engine my game will be built with. Once my character is working satisfactorily and running around a test map I will begin creating the three levels that my game will be played on. I will also add in enemy

AI to attack the player and incorporate fully functional health and stamina bars to the UI.

I will be creating cut scenes using Unreal Engine that will play throughout the games campaign for the arrival of new locations or the introduction of a boss character. These enemies will be found using Unreal assets which is a library of content available for download to be used in games.

Once my character is walking around the levels and fighting enemy AI without technical issues or bugs I will then work on polishing the gameplay experience and also work on some promoting for my game such as a poster or advert using unreal engine.

Special resources required

Required Software:

Unreal Engine 4

Blender

Project Plan

Gantt chart using Microsoft Project with details on implementation steps and timelines

Technical Details

Implementation language and principal libraries

To develop my project I will be using a combination of both blender and unreal engine. There are many libraries in unreal engine that I can avail of to improve the aesthetics of my project and I will be looking to incorporate these throughout development to both improve my game and free up some time to work on the more technical and difficult areas of my project. I will be using c++ while working in unreal engine for a number of details including a health bar, stamina bar, enemy health and AI.

Blender will be used to create my main character, While I can easily find a main character in the unreal engine library I thought it would not only add to the complexity of my project but it would also give it a much more unique and genuine feel. I will be creating the rigging and colouring of my character from scratch within blender before importing the character to unreal engine where it will become the player character.

Evaluation

Describe how you will evaluate the system with real technical data using system tests, integration tests etc. In addition, where possible describe how you will evaluate the system with an end user.

The main way I will test my game is by playing it, I will play the game a number of times to see what areas could be improved and I will also be asking my friends and family to play it so I can get some unbiased opinions on the non-technical areas such as how the game looks or which controls should be different or even if they have any input as to what the story should be like.

I feel through many play-troughs from different people I should get a well rounded interpretation of how the game plays and how the different functionalities do their jobs or don’t do their jobs.

The main method of testing I will use is black-box testing, to examine the functionality of my game and ensure that it is running to its highest potential. White-box testing will also prove useful when looking at how my game works and its internal structures but I feel functionality in a game is an extremely important aspect and should be tested vigorously.

10.2 Monthly Journals

10.2.1 October Journal

Reflective Journal

Student name: Graham Robinson

Programme (e.g., BSc in Computing): BSHc in Computing

Month: October

My Achievements

This month, I began working on my project. I started by downloading unreal engine, the engine through which I will be developing my game. I then created a test scene within the engine so I can test out any functionality I added to my game. Before I started working on any functionality in my game I created a simple test character in blender that will be used until I have created a real character for my game. I added in a Loot system that allows the player to pick up coins spawning on the map while at the same time keeping count of the amount of coins the player has picked up. This loot system will be improved upon in the later stages of my project as I hope to add the option for players to upgrade their characters health, damage, and other stats. I have also added many different sounds to my project to give it a more immersive feel.

My Reflection

At the moment I have established a strong starting point for my project and will keep adding to the foundations I have created in the coming months to complete my game.

I attempted to add enemies into my game but without success. I will try to add enemies into my game once I have a better understanding of the engine and the various elements involved.

Intended Changes

Next month, I will work on the user interface of my project by adding in some menus that will display when the game is started. I will also look into how enemy AI can be added to my game.

Supervisor Meetings

This month was the first time I had a meeting with my supervisor. During our meeting I explained the ideas I had for my final project. We agreed that I needed to add some complexity to my project idea which I will look to do in the coming months as the project develops.

10.2.2 November Journal

Reflective Journal

Student name: Graham Robinson

Programme (e.g., BSc in Computing): BSHc in Computing

Month: November

My Achievements

This month, I began working on the graphical user interfaces of the game. I created multiple menus that will be viewed and traversed by the player before they reach the main game. It is through the starting menu that the player will choose whether to begin the game or edit the games options. I began working on a number of menus including, start menu, options menu, quit screen. The start menu provides the user with three options, to start the game, to open the options menu and to quit the game. These are the first few menus that will be accessible to the player and I will look to add to the roster of GUIs as my project develops.

I began working on an enemy ai and am still early in the development of an enemy. I have begun working on the mesh of the enemy and have found many libraries within unreal engine that can assist me in this. I also spent time working on my analysis design report which was also due this month.

My Reflection

I feel that getting the first menus up and running was a big step forward in the development of my game as I now have a reliable base to build from when I continue to develop the menu system within my game. Although the menus still have very basic backgrounds and buttons I will be altering this at a later date. The enemy is proving far more difficult than first envisioned and I am still not making the progress I had hoped. I will continue to work on the enemy characters throughout the next month.

Intended Changes

Next month, I will update the existing GUIs in my game by creating backgrounds and buttons that fit in with the pirate them of the game. I will create these images using Photoshop. I will also focus on creating enemies for my game, I will use imported assets for my enemies that can be found in the libraries supplied within unreal engine. I will create enemies that spawn in certain areas of the map and that can attack the player once they have entered their field of vision.

Supervisor Meetings

In this month's supervisor meeting we discussed the status of our project and what we were planning on improving before our next meeting in December. In our next meeting I will bring a laptop in order to demonstrate my game and the progress I have made so far.

10.2.3 December Journal

Reflective Journal

Student name: Graham Robinson

Programme (e.g., BSc in Computing): BSHc in Computing

Month: December

My Achievements

This month, I completed the menu system of my game, with an options menu in which the user can change the resolution of the game along with a mute button with which users can mute any volume from the game. At the moment I can't seem to get the mute button to function properly but I am currently looking into how I can achieve this in the future. I incorporated a pause function and a pause menu into my game that can be accessed at any time by pushing the tab button on the keyboard. This pause menu shows the same information as the options menu giving the user a chance to change the resolution of the game and mute the audio. I have also made progress in creating enemies and now have a number of enemies that will disappear on contact with the player character; this will provide a strong starting point when I beginadding AI, damage, and health to both the games enemies and the player character.

My Reflection

I felt, I achieved a lot this month and now feel like I have an actual game to test. The custom backgrounds and buttons that I created using Photoshop add to the theme of my game and together with the sound effects I have added, give a much stronger sense of immersion. I now have a starting position for the development of enemies which is a huge step forward as I have been having trouble with this in previous months.

Intended Changes

Next month, I will try and improve on my enemies by adding things such as AI, Health, and damage. I will also try to add a health bar for my main character to the HUD. Another thing I will look into adding is a difficulty setting from which the user can dictate how difficult they want the game to be.

Supervisor Meetings

During my meeting with my supervisor this month we discussed how far we have come in our projects and our ambitions in the future in terms of what we would like to ultimately include in our projects.

10.2.4 January Journal

Reflective Journal

Student name: Graham Robinson

Programme (e.g., BSc in Computing): BSHc in Computing

Month: January

My Achievements

This month, I managed to get a working enemy into my game, my enemies now actively search for the player character and once they have seen them they will begin to chase the player. Once the enemy has caught the player they will begin to damage the player every second that they are within touching distance. I also added a health bar to the HUD of the game which accurately depicts the amount of health the player currently has left. I added in a health regeneration function to add to the mechanic of the game, every two second the player will regain a portion of their lost health back. I also managed to include a health bar for my enemies above their heads and can be viewed in game. Another thing I managed to add into my game this month was a difficulty setting, at the beginning of the game the player will be asked to choose between three difficulty setting, easy, medium, and hard. Depending on the players choice this will determine factors such as the player’s health, the player’s damage, and the frequency at which enemies will spawn from loot sources. I also found a mesh to use for my main character and have introduced many animations to my character to make her feel more life-like.

My Reflection

I felt, this has been a very productive month and I have added many things into my game, the introduction of a difficulty setting adds some more complexity to my game and also ensures there are more outcomes and possible ways to play through my game. The addition of enemies to my game now make the game more challenging from a player’s perspective and make it look more like an actual game, I am very happy with the health system I currently have in my game, in which the enemies can damage the player with that damage being represented in the HUD via a health bar, and also the opposite is true the player is now able to damage the enemies and that damage is also represented but this time through a small individual health bar above each enemies head. Now that my character has an actual skeletal mesh I am capable of giving it much more realistic animations again adding the realism of my game.

Intended Changes

Next month, I will continue to work on my main character, I have been having difficulties adding sockets to my characters skeleton which will allow my character to hold items in their hand, I will continue to look for a solution to the issues I am currently having with sockets and hopefully allow my character to pick up and use items such as swords and guns. I will also look into adding a save system into my game through which the player can save the loot they have collected in a bank like system to prevent the loss of loot should they die and need to restart the level.

Supervisor Meetings

During this month I met with my supervisor to go over what is expected from me during my mid-point presentation. I also asked my supervisor to look over my technical report and highlight any areas that need improvement before the deadline is met.

10.2.5 February Journal

Reflective Journal

Student name: Graham Robinson

Programme (e.g., BSc in Computing): BSHc in Computing

Month: February

My Achievements

This month, I had my mid-point presentation. In it I demonstrated the current state of my exam to two examiners and also explained the reasoning and theory behind my projects functionality. In terms of advancements in gameplay and mechanics, I have successfully added sockets to my character and now am capable of picking up and using items such as swords and guns. I now have a very realistic gun shooting animation, to achieve this I added two different sockets to my character, one in her hand and one on her hip. I opened the animation itself in the animation editor and found the precise second that the character should be picking up her gun and then using blueprints changes the socket being used from her hip socket to the hand socket, I then found the precise moment the gun was shot and added an explosion effect along with a sound effect to create a realistic idea of shooting a gun, finally I found the moment the character returns the gun to its holster and reversed the sockets. After I was successful in adding a shooting animation I focused on creating the bullet, I want the gun to cause damage to all items within a certain range from the player character and to have a cone of effect. To do this I created a cone shaped bullet that spawns once the gun is shot, once shot, barrels and enemies will be destroyed, chests however will not be affected by the bullet. I added in a cooldown for the use of the gun ability, when a player shoots their gun they will have to wait a certain amount of time before that ability will be available again, this cooldown is shown via a small icon in the bottom right of the screen that slowly becomes visible as the cooldown timer reduces. One small change I have made this month is that I have added a counter that keeps track of how many enemies the player has defeated, this is just a simple counter at the moment but I will be using it as a starting position for something more complicated later on. Finally, I began working on another level in which the player will carry out an optional tutorial.

My Reflection

This month I have added many things to my game and have drastically changed the gameplay, now that my game actually includes abilities and cooldowns it is starting to form into an actual RPG adventure game. I believe I now have very strong starting positions for some of the more advanced additions to my game and will look to further implement to ability system within Captain Ecks.

Intended Changes

Next month, I will look towards expanding the number of abilities that can be used in the game. I will also be completing a second map that will only become available to travel to once the player has defeated a certain amount of enemies. I will also look to getting a save feature working in my game that will allow the player to pick up from where they left off.

Supervisor Meetings

This month I had my mid-point presentation and so I met up with my supervisor prior to my presentation in order to get my report in order. My supervisor was also present during my presentation and gave me feedback on the status of my game and gave me some ideas of where I should direct my attention in the coming months in order to improve my project. My Supervisor gave me new ideas to help make my game more complex and interesting from the player's point of view, I will be looking into adding these suggestions next month which include a leader board system and also counting the amount of enemies the player has killed to ensure they have killed enough of them before allowing them to face the boss character.

10.2.6 March Journal

Reflective Journal

Student name: Graham Robinson

Programme (e.g., BSc in Computing): BSHc in Computing

Month: March

My Achievements

This Month I chose to focus on the suggestions given to me by my supervisor. The suggestions given to me were to incorporate a leader board function in the game and also to limit the player's access to the boss character until they have killed a certain amount of enemies.

I began with the leader board function, to do this I first had to allow the game to save variables. I first needed to create a Save Game Object in which I would include all of the necessary variables involved with creating the function. The game will create a save file called Score.sav within the game files when the player has completed the game, if there is already a Score.sav file in the game files then it will be overwritten with the new variable values. The score that is saved will be generated by a formula in the game depending on how well the player does in the game. The formula used is as follows; ((Total amount of coins collected) + (Number of enemies killed * 2) * Difficulty Multiplier) * 100. So if the player has collected 30 coins throughout the play through, killed 5 enemies and was playing on medium difficulty then the formula would look like this; (30 + (5 * 2) * 1.5) * 100 = 6000. Once the formula was working I then moved onto creating the GUI to display the scores and also a GUI where the player could enter their name. I ran into issues when trying to save the name and the player's score at the same time, to combat this I created more variables within the save game object and kept each name and score separate from the others. Once the player could enter their name and the game then displayed all of the saved scores to the player, the last thing I needed to do was to order the scores in descending order which was done by using branch blueprints in unreal engine which work as

IF loops. If the new score was bigger than the first score then it would go top of the list, if it was lower than the top score but higher than the second score then it would go second on the list and so on.

The other suggestion that my supervisor gave me was to restrict access to the boss fight until enough enemies were killed. I felt this would be a good addition to the game play as it provides an added challenge that must be overcome to complete the game and also provides a sense of accomplishment as the player finally gains access to the boss area after defeating enough enemies. To do this I came up with something I called the "SoulGate". The soul gate blocks off the boss area for the whole game and if the player approaches it the words "You need 10 souls to pass through the soul gate" will be displayed on the screen, this will be the message that will appear until the player has killed enough enemies, in that case the door will disappear and the words "Souls Accepted" will appear on the screen.

My Reflection

This month I have added all of the suggestions given to me by my supervisor and am very happy with the result. The addition of a leader board adds a competitive element to my game and promotes replaying the game after the player has already completed it. Although I ran into some problems with the very complicated saving system I was able to work through them and have developed a complex and fully functioning system to save and load both the player's score and the name that they enter once they complete the game. The addition of the soul gate also adds a secondary challenge or "mission" to the game that really helps expand the game play elements and forces the player to use all of their tools to eliminate enough enemies.

Intended Changes

I am at a very late stage in the development of my project and as of this moment all of the functionality I intended to develop has been included in my project in some way. I am happy with the functionality of my project and will now look to improving the appearance of the game by adding in more details such as tables, bottles, chairs, etc.

Supervisor Meetings

This month I met with my supervisor to demonstrate the changes made to my project since my mid-point presentation. Seeing that the functionality of my project is complete it was suggested that I move onto the testing of my game and to shift my focus from the practical side of my project to the technical side with my report still not being finished. My supervisor expressed what was expected of me in terms of each section in my report and also what I should focus on when it comes to my final presentation.

10.3 Other Material Used

10.3.1 Testing Survey

1. How old are you?
   - □ Under 20
   - □ 20 - 29
   - □ 30 - 35
   - □ Over 35

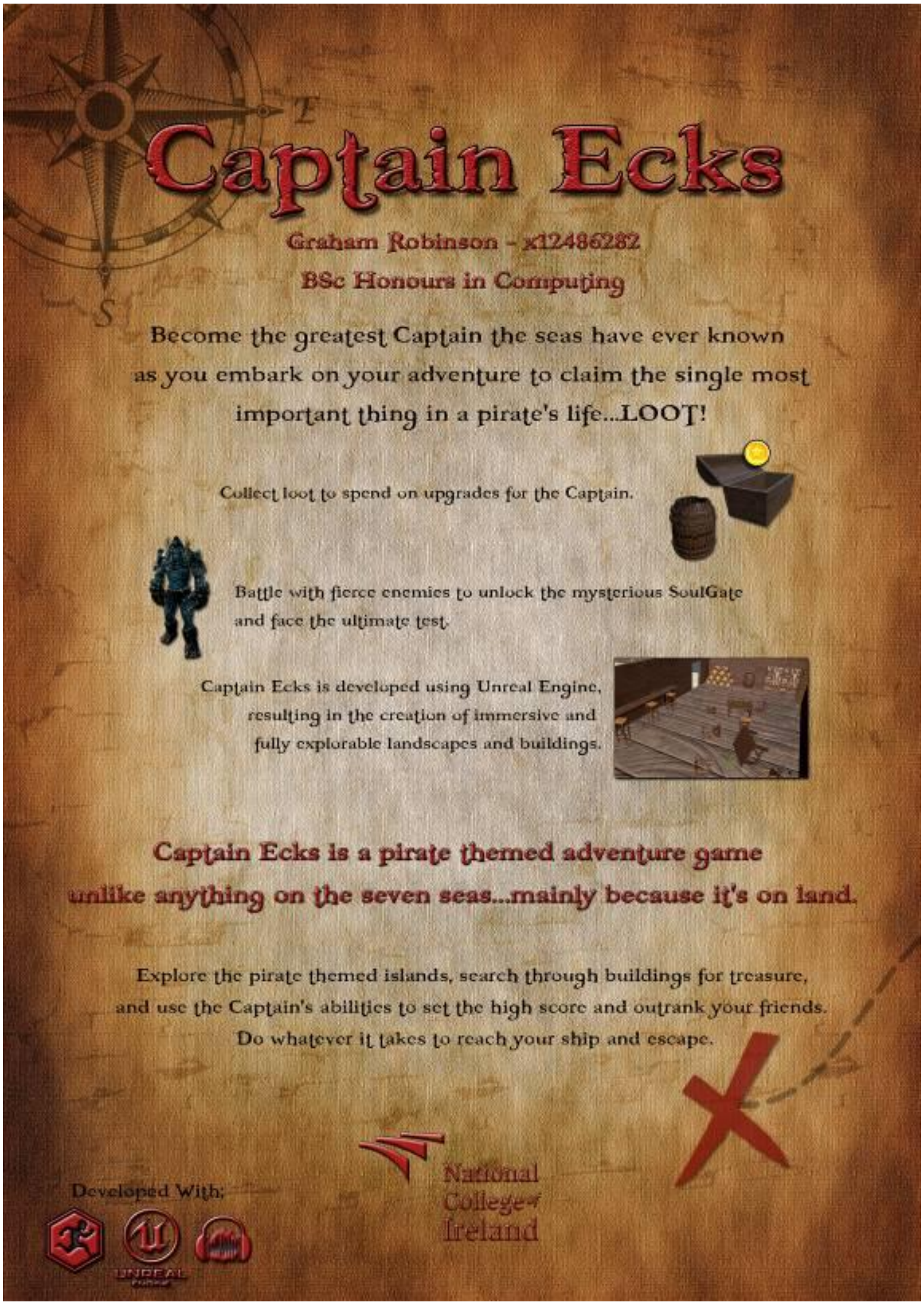
2. How many hours a week do you spend playing video games?
   - □ 0 - 2 hours
   - □ 3 - 7 hours
   - □ 8 + hours

3. While playing the game, was it clear what you needed to do?
   - □ Yes
   - □ No (if so please explain why below)

4. Did you find any bugs or errors during your play through of the game? If so what were they?
   - □ Yes
   - □ No

5. Do you have any suggestions that could improve this game?

10.3.2 Promotional Poster



Captain Ecks

Graham Robinson - x12486282

BSc Honours in Computing

Become the greatest Captain the seas have ever known as you embark on your adventure to claim the single most important thing in a pirate's life...LOOT!

Collect loot to spend on upgrades for the Captain.

Battle with fierce enemies to unlock the mysterious SoulGate and face the ultimate test.

Captain Ecks is developed using Unreal Engine, resulting in the creation of immersive and fully explorable landscapes and buildings.

Captain Ecks is a pirate themed adventure game unlike anything on the seven seas...mainly because it's on land.

Explore the pirate themed islands, search through buildings for treasure, and use the Captain's abilities to set the high score and outrank your friends. Do whatever it takes to reach your ship and escape.

Developed With:

UNREAL

National College of Ireland