University of Hertfordshire

School of Computer Science

BSc (Hons) Computer Science (Software Engineering)

6COM1056 - Software Engineering Project

Final Report

April 2021

Solve three major problems existent with current roguelike games today

Benjamin James Pearce

Supervised by: Hui Cheng

**Abstract:**

The goal of this project was to solve three major problems that persist throughout most roguelike games that currently exist, these three problems are:

* Missing a Save and Load Feature – meaning that the player has no way to save their progress and is forces to run the game all the way through in one play through.
* No Clear End Goals – meaning the player will have no clear goals to work towards which means it is not clear how the player should progress to complete the game, while also having little to no features to help the player track their progress.
* Poor Difficulty Balancing – meaning the game can either be too difficult or too easy for the player.

The methodology used for this project was an agile type methodology called test driven development. To act as plan and checklist for this project a Gantt Chart was created in the project planning phase that was split into seven different phases and was followed through the development of the project. The functional requirements for this project where created from mechanics and features that would be added to the game in order to offer solutions to one or multiple of the three major problems that acted as the main objectives within this project.

This artefact was designed with the main objects and the functional requirements of this project in mind so that the solutions designed would not only fulfil the functional requirements but would also achieve the main objectives of this project to an effective standard. The project research that was undertaken helped with this through researching pre-existing roguelike games along with in depth research on specific topics within game design and development, such as game mechanics and artificial intelligence, to make sure this project achieved its goals.

The implementation phase implements the solutions designed in the design phase, and as this project used the test driven development methodology there was testing after every new feature and after every planned phase on the Gantt Chart as a final review of what had been implemented up until that point.

The learning achieved throughout the development will be valuable when approaching projects in the future, especially the time management and self-management lessons learnt. The more detailed technically specific lessons will be valuable for the projects more similar to this one.

Contents

[1.0 Introduction 5](#_Toc70864536)

[1.1 Project Hypothesis and the Main Objectives 5](#_Toc70864537)

[1.2 Project Objectives and Functional Requirements 5](#_Toc70864538)

[1.3 Overview 7](#_Toc70864539)

[2.0 Project Research 7](#_Toc70864540)

[2.1 Research on other Roguelike Games 8](#_Toc70864541)

[2.2 Game Design 11](#_Toc70864542)

[2.3 Game Mechanics 12](#_Toc70864543)

[2.4 Player Motivation 14](#_Toc70864544)

[2.5 Artificial Intelligence (AI) 14](#_Toc70864545)

[2.6 Algorithms 16](#_Toc70864546)

[2.7 Methodology 17](#_Toc70864547)

[2.8 Technical Research 18](#_Toc70864548)

[2.9 Conclusion 18](#_Toc70864549)

[3.0 Legal Issues, Ethical Issues, and Project Risks 18](#_Toc70864550)

[3.1 Legal Issues 18](#_Toc70864551)

[3.2 Ethical Issues 19](#_Toc70864552)

[3.3 Project Risks 19](#_Toc70864553)

[3.4 Summary 20](#_Toc70864554)

[4.0 Planning the Project 20](#_Toc70864555)

[4.1 Player Design 21](#_Toc70864556)

[4.2 Design for the Random Dungeon Generation 21](#_Toc70864557)

[4.3 Designing the Enemies and the Boss 22](#_Toc70864558)

[4.4 Designing the Reward Drops 25](#_Toc70864559)

[4.5 Designing the Minimap 26](#_Toc70864560)

[4.6 Designing the HUB World 26](#_Toc70864561)

[4.7 Designing the HUB World Upgrades 27](#_Toc70864562)

[4.8 Designing the Save and Load Feature 28](#_Toc70864563)

[4.9 Designing the Player Upgrades 28](#_Toc70864564)

[4.10 Designing the Equipable Weapons 28](#_Toc70864565)

[4.11 Designing the Boss Trophies 29](#_Toc70864566)

[4.12 Designing the Tutorial 29](#_Toc70864567)

[4.13 Designing the Dungeon and Sector Level Menus 29](#_Toc70864568)

[4.14 Designing the Different Room Types 29](#_Toc70864569)

[4.15 Summary 30](#_Toc70864570)

[5.0 Implementing the Project Objectives 30](#_Toc70864571)

[5.1 Random Dungeon Generation 30](#_Toc70864572)

[5.2 The Player Objects 36](#_Toc70864573)

[5.3 Adding the Enemies and Boss to the Dungeons 37](#_Toc70864574)

[5.4 Adding the Reward Drops 43](#_Toc70864575)

[5.4 Implementing the Minimap for the Dungeons 48](#_Toc70864576)

[5.5 Implementing the HUB World 49](#_Toc70864577)

[5.6 Adding the HUB World Upgrades 50](#_Toc70864578)

[5.7 The Save and Load feature with Three Available Save Slots 52](#_Toc70864579)

[5.8 Add Player Upgrades 53](#_Toc70864580)

[5.9 Add Different Equipable Weapons for the Player 55](#_Toc70864581)

[5.10 Add Boss Trophies 56](#_Toc70864582)

[5.11 Add a Tutorial Menu 57](#_Toc70864583)

[5.12 Dungeon and Sector Level Selector Menus 58](#_Toc70864584)

[5.13 Implementing the Different Types of Dungeon Rooms 59](#_Toc70864585)

[5.14 Summary 61](#_Toc70864586)

[6.0 Project Testing 61](#_Toc70864587)

[7.0 Version Control 62](#_Toc70864588)

[8.0 Project Evaluation 63](#_Toc70864589)

[9.0 Bibliography 65](#_Toc70864590)

[10.0 Appendices: 66](#_Toc70864591)

[10.1 Appendix A – The Game Design Document 66](#_Toc70864592)

[10.2 Appendix B – GitHub Repository for the Project Code 68](#_Toc70864593)

[10.3 Appendix C – Gantt Chart 69](#_Toc70864594)

[10.4 Appendix D – Project Tests 71](#_Toc70864595)

# 1.0 Introduction

For this project a roguelike game artifact was made to solve three major problems that most roguelike games show, these problems being: Missing a Save Feature, having No Clear End Goals, and Poor Difficulty Balancing. So this project had three main objectives, one for each problem. These three problems need to be solved as they could risk player motivation decreasing and the game becoming less and less playable, and the goal of a game is to keep the player entertained and motivated to play.

## 1.1 Project Hypothesis and the Main Objectives

The hypothesis for this project, based on the brief above, is as follows:

“Based on the three main problems with roguelike games: Missing Save Features, No Clear End Goals, and Poor Difficulty Balancing. Make an effective solution that offers solutions to all three of these problems.”

To prove this hypothesis the main objectives must be completed, details for each of these main objectives are shown below:

**Missing save features** will demotivate the player to play the game as it forces a permadeath run on the player, a permadeath run being when a game must be played in a small amount of sittings with little to no save checkpoints. This can be done well with adding a game mode or difficulty for the game which deletes your save when you reach game over, however no save feature at all means that the game must be played in one sitting and once you reach game over or close the game you have to start at the very beginning. Not many games today have no save feature at all, however there are a lot of roguelike games that have so little done towards adding save features that they might as well not be there at all.

**No clear end goals** will demotivate the player to play the game as these goals will give the player a goal or checkpoint to work towards while simultaneously giving the player a way of determining how far they have progressed in the game. Having too few goals and leave the player with too little to do, while having too many long term goals and leave the player unsatisfied with their progress with no smaller benchmarks to show how much they are progressing. Having a good mix of short and long term goals that the player can complete will keep them motivated to play the game and satisfied with their progress with these smaller goals leading up to helping the player complete the bigger goals that the game has.

**Poor difficulty balancing** will demotivate the player to play the game as making the game to difficult will the game unplayable with the player not being able to complete their goals and stopping them from making progress, however if the game is too easy then the player will quickly complete all the goals and the feeling of progression will not be as satisfying to the player as if the goals where challenging yet completable. Loads of features can contribute to difficulty balancing, including the end goals the game gives the player and having a save feature also contributes. Especially if you allow the player to save a lot then there would be no challenge to the levels, yet with no save points can make the game almost impossible to complete.

## 1.2 Project Objectives and Functional Requirements

To complete the main objectives features were added that would work towards completing one or multiple of each main objective. These features where not only the projects objectives but where also functional requirements, along with some other features. The functional requirements for this project are listed below:

* Implement the player object
* Implement the random dungeon generation
* Implement enemies and a boss to the dungeon levels
* Implement reward drops on finished encounters, these include: health pickups, gold pickups, upgrade chip pickups, hyper core pickups, the boss key, and powerups
* Implement Minimap to the dungeon levels
* Implement the HUB World
* Implement HUB World Upgrades
* Implement a Save and Load feature with three available save slots
* Implement Player Upgrades
* Implement Different buyable and equipable weapons for the Player
* Implement boss trophies for when the boss is defeated
* Implement a tutorial menu
* Implement menus to choose different dungeons and sector levels within the game
* Implement Different Room Types within the Dungeons

A lot of the points in the functional requirements for this project where project objectives that needed to be completed and worked towards the main objectives of the project. To show this the table below goes through each objective for the project, a description of the objective, and which main objectives they work towards.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **Description** | **Save Feature** | **Clear End Goals** | **Difficulty Balancing** |
| Minimap | This feature show the layout of the dungeon as the player progressively explores the dungeon |  |  |  |
| Different Room Types | There are four different room types in the dungeon: the start room, the bot haven room, the workshop room, and the boss room. These rooms work a little differently to the normal enemy rooms which can give the player new challenges or give the player tools that they can use later on. |  |  |  |
| Reward Drops | Rewards that can be dropped after an enemy encounter and can also be brought in the Workshop room and given in the Bot Haven room |  |  |  |
| HUB World | A world where the player can be upgrades and choose the next dungeon the player will enter |  |  |  |
| Hub World Upgrades | Upgrades that can be brought in the HUB World to customise the HUB world |  |  |  |
| Player Upgrades | Upgrades that can be brought in the HUB World to give the player buffs and tools when in the dungeon |  |  |  |
| Save and Load Feature | Allowing the player to save and load the game in the HUB World and the game autosaves in many places but only loads a game into the HUB World |  |  |  |
| Different Weapon Types | Being able to buy and equip different weapons that allow for different playstyles and features that allow the player to choose which weapon is the most efficient for them |  |  |  |
| Boss Trophies | These are collectables which are collected once a boss has been defeated and can be seen in the HUB World |  |  |  |
| Sector and Dungeon Menus | The dungeon menu allows the player to choose the dungeon they will attempt next and shows which dungeons they have completed  The sector menu allows the player to choose different sectors which act as levels containing multiple dungeons, the player can only choose the next sector when all the dungeons in the current sector have been completed |  |  |  |

## 1.3 Overview

This section is an overview of the methodology used and an overview of this report.

1.3.1 Methodology Overview

For this project the test driven development methodology was used to focus on the implementation and testing for specific features and sections one at a time, refactoring code when needed to make the features work. This methodology was used to give more focus to each feature during development to make the most efficient and working solution for every feature added to this game.

1.3.2 Report Overview

In this report there are multiple sections that will cover the different phases of this project in detail, these sections being: Project Research, Legal Issues and Project Risks, Planning the Project, Developing the Project, Project Testing, Version Control, and Project Evaluation.

# 2.0 Project Research

In this section the project research undertaken is shown and explained, for this project the research undertaken needed to be not just about other roguelike games but also about game design, game mechanics, player motivation, artificial intelligence, algorithms, and the development methodology. Some experimentation was also needed to experiment how to implement features so the most effective solution could be found.

## 2.1 Research on other Roguelike Games

Research was conducted on other roguelike games to look for examples of how the projects main objectives, as the three main problems, affected these pre-existing games whether these problems existed in the game or not. This research was split into three categories for each of the main objectives.

2.1.1 Save Features

Within most roguelike games save features are limited to keep the game challenging with the player usually not being able to save in the dungeon and in some safe area of the dungeon or a separate HUB world. When the save feature is absent however it forces the player to play the game in one run and starting the game from the beginning on every game over. McMillen and Himsl developed the Binding of Isaac (McMillen & Himsl, 2011) which is a roguelike game where you go through multiple layers of a randomly generated dungeon, however with no safe area or place to save if you get a game over on any level then you have to start at the beginning as the save feature is absent in this game. This makes the game too hard as each dungeon level ramps up in difficulty so having to start from the beginning every time can frustrate the player and demotivate them from playing your game.

Figure 1.1 - The Binding of Isaac Gameplay:



FTL: Faster Than Light (SubsetGames, 2012) developed and published by Subset Games, this game has a save feature unlike the Binding of Isaac (McMillen & Himsl, 2011) as the game only saves when you click the Save and Quit button and quits the game entirely once it’s saved, however you can save in any dungeon room or area provided any encounters have been finished. Usually roguelike games do not allow you to do this however FTL: Faster Than Light (SubsetGames, 2012) has big dungeons with lots of levels to them which only get harder and harder so this does not make the game to easy.

Figure 1.2 - FTL: Faster Than Light Gameplay:



For this project inspiration was taken from Hades (SupergiantGames, 2018), a game developed and published by Supergiant Games which allows the player to save in a HUB world and gives the player three save slots so they can make new game saves without losing the old one unlike FTL: Faster Than Light (SubsetGames, 2012) which only gives the player one save slot.

Figure 1.3 – HADES Gameplay:



2.1.2 Clear End Goals

Clear end goals give the player completable goals that they can work towards which can in turn also give them a good grasp on their progression within the game. Two types of goals can be used in games, long term and short term goals, short term being goals being goals that can be completed quicker than the long term goals. Enter the Gungeon (DodgeRoll, 2016) developed by Dodge Roll and published by Devolver Digital is a game that gives the player a limited amount of goals with most of them being long term goals like completing the current dungeon and finishing the game, this leaves the player with a bad grasp on their progression as well as less to do as short term goals act as smaller rewarding goals that lead the player up to completing these bigger goals within the game. Without a good sense of progression and the same long term goals this can leave the player bored and unsatisfied with repetitive goals and little rewards.

Figure 1.4 – Enter the Gungeon Gameplay:



However Hades (SupergiantGames, 2018) offers a solution to this problem by adding lots of short term goals for the player to complete, whether that’s player and dungeon upgrades to help them progress further through the dungeon levels or if it’s cosmetic upgrades for the HUB world and giving characters gifts to learn more about their lore. All of these short term goals give the player more things to do between dungeon runs, which some even help the player by giving them more tools and abilities when completing dungeon levels and such. These upgrades and the different weapons you can unlock in the HUB world give the player more tools and stronger abilities that can make the earlier levels of the dungeon easier when going through them again so the player will slowly but surely make their way to newer levels further in, therefore giving them a good sense of their overall progression in the game and making the player feel like they’re actually progressing within the game.

For this project inspiration was taken from Hades (SupergiantGames, 2018) and added a HUB world where the player can buy upgrades and weapons that will help them through the dungeons, but also implemented cosmetic upgrades for the HUB world and boss trophies to give the player more short term goals to fulfil with their own rewards.

2.1.3 Difficulty Balancing

The game must be challenging enough to give the player satisfaction when completing goals and gaining rewards, however if the game is too difficult then the game can become incompletable. There are many features that add towards difficulty balancing, including where and how the player can save and the difficulty level of each goal that is in the game. The Binding of Isaac (McMillen & Himsl, 2011) gives the player challenging enemies and bosses that only get harder with each dungeon, however with no save option and not as many powerups that help with difficulty balancing so the game becomes almost impossible to complete.

FTL: Faster Than Light (SubsetGames, 2012) gives the player more challenging opponents and situations as the game progresses while also giving the player shops in every dungeon that they can always return to, these shops allow them to upgrade their ship, buy new crew members to maintain more areas of the players ship and to boost more areas at once, and also allows the player to buy new weapons and gadgets to equip which get better as the game progresses. With each ship upgrade the player gets the more the player can equip and the more powerups/boosts the ship gets for each area. This not only continues to give the player a good sense of progression, but most importantly it gives the player a lot of tools which allow the player to progress through the game with a good level of balanced difficulty.

Hades (SupergiantGames, 2018) is also a good example of good difficulty balancing with the different upgrades and tools the player can buy and equip along with having a HUB world the player can save in to keep progress. Hades (SupergiantGames, 2018) makes each new level of the dungeon challenging while the beginning gets easier as the player upgrades and learns more about the game, and when the player gets to a certain mark in progression new enemies will spawn in the earlier levels of the game to keep the game challenging but also add some variety to the game as well. Hades (SupergiantGames, 2018) also has special room types that can give the player unique powerups as well as more resources to buy the permanent powerups in the HUB world, along with also giving the player boons which act as temporary powerups for each specific dungeon run to keep the game interesting with variety while also giving the player useful tools that they should make use of to get further into the game.

For this project inspiration was taken from Hades (SupergiantGames, 2018) for the upgrades and weapons in the HUB world, the powerups, and special room types. However I took inspiration from FTL: Faster Than Light (SubsetGames, 2012) to give the player less types of special rooms with all those special rooms being able to grant the player useful tools. Adding a shop room called a workshop was direct inspiration from FTL: Faster Than Light (SubsetGames, 2012) as a shop in the dungeon allowed the player to choose put of a range of randomised options for tools and such they could use adding more strategy into the game while also keep the difficulty well balanced. Also like from FTL: Faster Than Light (SubsetGames, 2012) the rooms can be revisited so the player can come back to buy more useful tools if they get more gold and progress further into the dungeon.

2.1.4 Summary

This section showed research undertaken from different types of roguelike games which showed practical examples of how and why these main problems are problems and how and why they should be solved. This research also allowed inspiration to be taken from these games for the solutions created in this project.

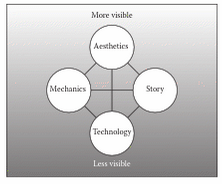
## 2.2 Game Design

Game design is an important process for games and the first step to creating any game as this helps plan out the features that the game will have and make sure that these features come together to make a final product which is well balanced and enjoyable to play. For this project topic research was conducted on: the important elements that form games, MDA (Mechanics, Dynamics, and Aesthetics), and Game Design Documents.

2.2.1 Important Elements that form Games

Schell (Schell, 2019) made the elemental tetrad which made up of the four most important elements for a game, these elements are: Aesthetics, Mechanics, Story, and Technology. Each of these elements linking together to form the games we play. The elemental tetrad diagram is shown in Figure 1.5 below.

Figure 1.5 – The Elemental Tetrad:



Schell (Schell, 2019) also defined each of these elements as such: he defined Mechanics and procedures and rules that describe the goal of your game, how players achieve or fail these goals, and what happens when goals are achieved (Pg. 53). He defined Story as a sequence of events that unfold during gameplay (Pg. 54). He defined Aesthetics as the looks, sounds, and feels that are experienced throughout the game (pg. 54). And he defined Technology as materials and interactions that make the game possible which for this project would the a PC as the game was made and can be run on a PC, but it could be different for different types of games like mobile games and console games (Pg. 54).

For this project the elemental tetrad was used to remember the important elements that make up games so when designing this game each of these elements in the elemental tetrad can be used effectively.

2.2.3 MDA – Mechanics, Dynamics, and Aesthetics

Hunicke, LeBlanc, and Zubek (Hunicke, LeBlanc, & Robert, 2004) define MDA as a framework which is made up of Mechanics, Dynamics, and Aesthetics (Pg. 1). MDA is a formal framework which is used to analyse a game by breaking down the individual mechanics that make up this framework. Mechanics and Dynamics were defined in the section above by Schell (Schell, 2019) and this framework uses the same definition for both; however Hunicke, LeBlanc, and Zubek (Hunicke, LeBlanc, & Robert, 2004) added Dynamics to this framework which they define as the run time behaviour of the mechanics which activate in a certain way depending on the inputs received from the player and other variables (Pg. 2).

For this project MDA was used when designing the game along with the elemental tetrad to make sure each element was being used as effectively as possible within the game and that the designs used this framework to design mechanics and the relationship between mechanics and the variables that activate said mechanics.

2.2.4 GDD – Game Design Document

Rogers (Rogers, 2010) defined a GDD as a document that was made up of different types of documents that come together in this GDD to work as a proposed design for the game. The types of documents that make a GDD are as follows: the one sheet, the ten-pager, and the beat chart (Pg. 59). Rogers (Rogers, 2010) defines the one sheet as a simple design overview of the game being created (Pg. 60), he defined the ten-pager as a design document which lays out the so called spine of the game, referring to the elements of the game which server as the main features or mechanics that make this game playable (Pg. 62), and he defined the beat chart as a document that would be expanded upon throughout the development lifecycle of the game being created and allows the developers and designers to develop more designs and elements as a continued plan for the games structure (Pg. 74).

For this project a GDD was created to be part of the designing phase of this project and was continuously updated through the beat chart, the GDD document created can be seen in Appendix A – The Game Design Document.

2.2.5 Summary

This section showed research undertaken about game design and was important for the design phase of this project so a game could be designed so that the objectives of this project could be fulfilled effectively.

## 2.3 Game Mechanics

Research was conducted on game mechanics as game mechanics were created and designed in this project, so research was needed to understand game mechanics more and how good mechanics are created to allow the player to play the game while implementing the objectives for this project.

2.3.1 Gameplay vs Game Mechanics

There is a difference between gameplay and game mechanics, as gameplay is defined as the events players can interact with whereas the game mechanics are the systems in place that make those events intractable and work for the player. Suter, Kocher, and Bauer (Suter, Kocher, & Bauer, 2018) state that game designers have to distinguish the difference between these two different levels of design as shown below:

* Gameplay design is regarded as motivation design for the players playing the game while taking into considerations the perspective of the players as they play.
* Game mechanics design contains the programmable visceral construction of the architecture of the game which will manifest itself on the level of the program code or an abstraction of it, meaning that the game mechanics are the elements in the game the player uses to interact with the game and allows the player to experience the gameplay. (Pg. 73)

For this project both gameplay and game mechanics had implications on the design and development process as they do for most games, so designing these two elements well was crucial for this project to design and develop gameplay and game mechanics which offer solutions towards the main three problems.

2.3.2 The Layers of Game Mechanics

Suter, Kocher, and Bauer (Suter, Kocher, & Bauer, 2018) make reference to the Adams and Dormans categorization of the five different layers of game mechanics, which are defined as shown below:

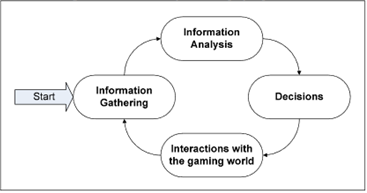
* The first layer represents the physics of the game which refers to the physical laws of the virtual world created and how they work.
* The second layer represents the internal economy within the game which refers to the law of production and distribution of in game resources. For this project this would be represented in the currency used to buy upgrades and weapons in the HUB world.
* The third layer represents the progression of the games which refers to the design of each individual level and progress conditions that are shown throughout the game.
* The forth layer represents the tactical manoeuvring within the game which refers to the strategic distribution of the game units such as the enemies and how the player can move around the level, this is especially used within respect to combat within the game.
* The fifth and final layer represents social interaction within the game which refers to the techniques and processes that allow and define the interaction between the player and the AI and other intractable elements, is also used for interaction between multiple players but this project does not used this as it is a single player focused experience. (Pg. 83)

This project implements these different layers that are important as they all come together to make a playable game that can have mechanics added to offer solutions to the three main problems being solved in this projects objectives. Not only that the mechanics made within the projects objectives also fall under some these layers.

2.3.3 Interactive Cycle in the Gameplay Experience

Fabricatore (Fabricatore, 2007) defines an interactive cycle in the gameplay experience of a game, which regardless of specific game contents while the playing the game the player will interact with this virtual universe as this virtual universe will receive the inputs from the player. Information regarding the outcome of the interaction is then conveyed to the player as output which then eventually gets gathered and used by the player themselves to decide what to do next. (Pg. 3) The explanation above defines the interactive cycle of play as defined by Fabricatore (Fabricatore, 2007) which is used by most games to map how the player interacts with the game while they play. The Interactive cycle in gameplay diagram is shown in Figure 1.6 below.

Figure 1.6 – The Interactive Cycle in Gameplay:



Taking this cycle of play into consideration was important for this project as to solve the main three problems and successfully complete this projects objectives, the solutions offered needed to be able to follow this cycle of play to keep the player engaged and keep them interacting with the game.

2.3.4 Satellite Game Mechanics

Fabricatore (Fabricatore, 2007) defines Satellite Game Mechanics as game mechanics which enrich the core gameplay if a game without increasing the games complexity so the learning curve is not as steep. He also stated that satellite game mechanics are named as such since their design is functional to already existing core mechanics, meaning they are mechanics that work more in the background. There are three kinds of satellite game mechanics: enhancement mechanics to enhance pre-existing game mechanics, alternate mechanics which is the same activity or event that has the option to be completed in a different way, and opposition mechanics which hinder the player’s progress to give them more of a challenge. Opposition mechanics need a lot of thought put into them so they are not overdone and make the game too difficult for the player. (Pg. 13)

This project makes use of these types of mechanics to help boost other mechanics that offer similar solution as part of the projects objectives.

2.3.5 Summary

This section showed research undertaken about game mechanics so game mechanics that were designed and developed for this project were effective in completing the project objectives.

## 2.4 Player Motivation

Player motivation is a very important research topic as this details how to keep the player playing the game and in what ways this can be done.

2.4.1 Intrinsic Motivation

Przybylski, Ryan, and Rigby (Przybylski, Ryan, & Rigby, 2010) defined intrinsic motivation as behaviour pursued for their own sake rather than pursuing the behaviour for any inherent satisfactions or rewards. (Pg.155) For this project having goals that can be achieved for the sake of achieving them have been added in the form of HUB world cosmetic upgrades and the collectable boss trophies when defeating a boss as these do not add anything to the game apart from cosmetic looks and some background text on the boss however these goals can still be achieved.

2.4.2 Extrinsic Motivation

Przybylski, Ryan, and Rigby (Przybylski, Ryan, & Rigby, 2010) defined extrinsic motivation as goals and behaviours that are pursued to access desired end states or avoid aversive ones depending on the end result of said behaviour or goal. This project implements a lot of extrinsic motivation, from completing the dungeons to unlock more of the game and progress further to the upgrades and weapons that can be brought and equipped to give the player abilities and tools to use as they play the game. All of these goals that work towards extrinsic motivation offer some form of reward that can be used in the game functionally as the gameplay progresses.

2.4.3 Summary

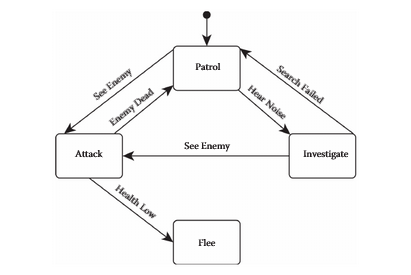
Both of these are very important when considering the player and how the player the game as they are motivated to the play game as if they are motivated to play the game then the game will be left unplayable and the projects objectives will have failed.

## 2.5 Artificial Intelligence (AI)

For this project as a game, Artificial Intelligence (AI) refers to any characters that the player does not control but can interact with, which can come in the form of any type of NPC (non-playable characters) such as enemies and bosses in this project.

2.5.1 Finite-State Machines

Rabin (Rabin, Game AI Pro: Collected Wisdom of Game AI Professionals. 1st Edition., 2013) defines a Finite-State Machines (FSM) as the most common behavioural modelling algorithm used in game AI programming today. A FSM breaks down and NPS’s overall AI into smaller and discrete pieces known as states, each of these states represent a specific behaviour or internal configuration that an AI performs and the conditions for said behaviour to occur (Pg. 48). An example of a Finite-State Machine is shown in Figure 1.7 below.

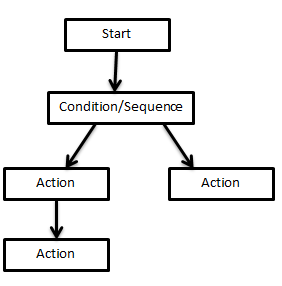
Figure 1.7 – Finite-State Machine Example:  


The project uses these FSMs to plan out the different types of enemies and the boss’s behaviours as they interact with the player and the game world.

2.5.2 Behaviour Trees

Rabin (Rabin, Game AI Pro: Collected Wisdom of Game AI Professionals. 1st Edition., 2013) defines behaviour trees as data structures starting from a root note which then branches off to many different nodes which represent behaviours or actions of an AI. Each parent node can have child behaviours in the form of child nodes which are connected to the parent behaviour or action nodes; this gives the algorithm a tree-like quality (Pg. 52). An example of a Behaviour Tree is shown in Figure 1.8 below.

Figure 1.8 – Behaviour Tree Example:



This is used in the project for further designs on how the AI in the game will behave based on actions and other activated behaviours.

2.5.3 Probability and Possibility Maps:

Rabin (Rabin, Game AI Pro 2: Collected Wisdom of Game AI Professionals. 2nd Edition., 2015) defines a probability map as a map that provides mapping from game states to probabilities that represent the relative likelihoods of the game being in different shown states, and Rabin (Rabin, Game AI Pro 2: Collected Wisdom of Game AI Professionals. 2nd Edition., 2015) defines possibility maps as simplified versions of probability maps that dispense with probability values and only map states to indicate as to whether the states are possible within the game (Chapter 7.2).

For the project these maps helped map out the different probabilities of certain states within the AI and how probable these states are, along with mapping out the rules of the AI along with their abilities and weapons.

2.5.4 Summary

The research undertaken in this section helped with designing effective AI that could be used when developing the AI enemies and the boss as one of the projects objectives

## 2.6 Algorithms

Algorithms are defined as sets of rules and procedures to be followed in calculations or other problem solving operations. For this project the specific algorithms used where randomisation algorithms to choose random rooms to generate while spawning rooms for the dungeon along with other randomisation algorithms for choosing room types and such. There is also an algorithm used for spawning the randomly chosen rooms into a dungeon which is not a randomisation algorithm.

2.6.1 Randomisation Algorithms

Aspnes (Aspnes, 2020) states that formally we think of randomised algorithms as a machine M that computes the formula M(x,r), where x is the problem input and r is the sequence of random bits (Pg.1). Using this definition of the algorithm the choosing of random indices for choosing which room will be spawned or other features like choosing which out of the spawned rooms will be the boss room. This is shown with M for the machine running the game, x for the array or set that includes all the different types of rooms or room types depending on which this algorithm is randomly choosing, and r for the randomly chosen index or indices depending on how many indices are being generated.

2.6.2 Spawning the Dungeon

For spawning the different rooms for each dungeon a different algorithm needed to be used to make sure the dungeon stayed within a range for the max amount of rooms spawned to make sure the dungeon was not too big and that each room spawned has openings that lead into rooms and not into walls. Using the randomisation algorithm to choose a room inspiration was taken from a tutorial series made by Blackthornprod (Blackthornprod, 2018) which had each spawn able room have spawners and destroyers to spawn new rooms and to make sure that rooms don’t spawn over already existing rooms. Also the system and algorithm in the spawners used in the tutorial by Blackthornprod (Blackthornprod, 2018) was for a 2D game and convert it for a 3D game as this project is made for 3D which meant using different code for the slightly different working system. Using this inspiration a similar system was made with the spawners and destroyers using the randomisation algorithm to choose which random room to spawn and then an entirely new algorithm was created to check the rooms that have been spawned and the rooms and the rooms around said spawned room to check that all openings lead to a room and no openings will lead to a wall blocking the player from going through.

2.6.3 Experimentation

For the randomisation algorithm to choose a random room to spawn the array was shuffled and a random index was generated through using the random object from the basic C# System library and the Fisher-Yates shuffle algorithm, defined in a GeeksforGeeks article (GeeksforGeeks, 2020), was used for shuffling the array itself. This algorithm works in O(n) time complexity, meaning the function operates really quickly and is an efficient algorithm to use. The Fisher-Yates shuffle algorithm works by swapping the last element with a randomly chosen element using the random object in the C# System library to generate a random index to find this randomly chosen element. The spawners set up outside the rooms as child objects are then with spawning the new room, unless this spawner collides with a destroyer as these are placed in the middle of rooms this will detect that a room has already been spawned here and delete the spawner that collided with said destroyer as it should not spawn a room.

2.6.4 Summary

The research undertaken in this section was used to figure out how the random dungeon generation should be implemented in this project, as this generation is used for every dungeon level.

## 2.7 Methodology

Methodology refers to the models and techniques that were used while designing and developing this project.

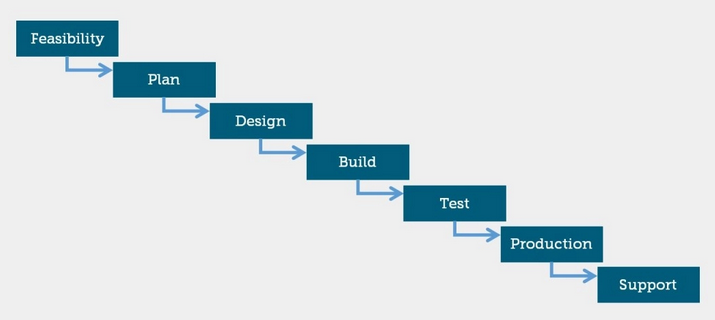
2.7.1 Agile Development Methodology

As defined by cprime (cprime, Undated) the Agile Development Methodology refers to a group of multiple software development methodologies based on iterative development, where requirements and solutions made evolve throughout the project lifecycle. Out of many of the software development methodologies that came under the agile development methodology umbrella, the one this project chose to follow was the Test Driven Development Methodology.

2.7.2 Waterfall Methodology

Waterfall was the other type of methodology that was considered, this methodology being defined as a model which is made up of different phases and these phases take place in a sequence, so progress flows steadily downwards through the phases like a waterfall (Bowes, 2014). Figure 1.9 below shows a diagram for the Waterfall methodology (Bowes, 2014).

Figure 1.9 – Waterfall Methodology Diagram

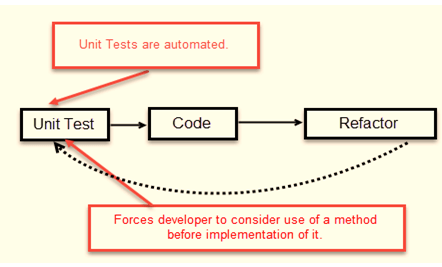


However as the Waterfall methodology does not work well with changes to requirements and going back to previous sections does not really happen means that this methodology is less flexible, which this project needed flexibility for testing and different implementations of the same functional requirement as the previous implementation may not work as intended. This is why the Test Driven Development Methodology was used as discussed in the section below.

2.7.3 Test Driven Methodology

As defined by Kanchan Kulkarni (Kulkarni, Undated) Test Driven Development (TDD) is a type of software development approach in which test cases are developed to specify and validate the code that has been written within a project. This means that test cases are made for each individual functionality that is created and if the test for this functionality fails then new code is written or old code is refactored to make the functionality pass the next test, meaning this methodology is more flexible which would be the best for this project. Figure 2.0 below shows a diagram for the Test Driven Development Methodology (Kulkarni, Undated).

Figure 2.0 – Test Driven Development Methodology Diagram



2.7.4 Summary

While using TDD within this project, tests where designed and developed for each individual mechanic that was created within this project, this is so when the mechanics have been created they can be tested effectively to find any bugs or errors that may still exist and then fix any of those bugs or errors that showed up during any created tests.

## 2.8 Technical Research

In order to create the project in Unity, research needed to be done on the engine and research needed to be done on the language used which in this case is C#. Unity was decided the better engine to use as it had the option for 3D which was what this project would use. For all the scripting and engine information that was needed during development the Unity Documentation by Unity (Unity, 2020) was used, which was used to look up functionality behind specific in built Unity libraries, methods, and classes which is the most accurate source to learn about these in built features as this was created and published by Unity themselves.

## 2.9 Conclusion

With this research done it could then be used when designing and developing the different mechanics within the objectives of this project. The games and the specific mechanics, and Ai where designed and the algorithms where tested in an experimentation using the research conducted in the section. The methodology research was important as this decided the structure of how this project was to be designed and implemented, which test driven development was chosen for this project.

This research was important for this project so the projects objectives could be met effectively while also finding three main problems that were solved as three main objectives in this project along with examples of why having these problems was bad with existing solutions to take inspiration from when designing the solutions within this project.

# 3.0 Legal Issues, Ethical Issues, and Project Risks

This section will cover the potential legal issues that could have been encountered, and the project risks that could have happened or did happen during the development of this project.

## 3.1 Legal Issues

For games there is only one big legal issue that could arise, that being copyright issues and other similar issues that could arise. As the assets were made as part of this project and was made by a third party, there would be no licensing or publishing errors as no third party helped in the making of these assets. However if characters and elements of the game are very similar to that of another game then that could a violation of copyright laws, as New Media Rights (NewMediaRights, 2020) states that artwork and other assets within a game only has copyright as long as no one has closely duplicated it, as an example if a character was created that looked very similar to a pre-existing character then there could an issue. To mitigate this research was conducted on other roguelike games not just for research on my project objectives but also to make sure that the assets and ideas designed and used were original enough that no copyright laws could be violated, and making the assets without the use of a third party mitigates the violation of any licensing issues that could have occurred.

## 3.2 Ethical Issues

The only ethical issues that could be involved in this project is taking other people’s work and taking it as an original part of the project, which falls under the legal issue of copyright laws. This will not be an issue in this project however as all assets used are made using Blender and not taken from an external source. The code has been inspired by external sources in specific parts as stated in the implementation section further into this report; however code was not copied directly from these external sources and was modified and changed to fit the projects objectives so this will not violate this issue. If testing of this game in the future were to occur with third party participants who are not involved in the development of this project, then ethics approval/permission would need to be approved to use the information gathered from testing this project using these third party participants.

## 3.3 Project Risks

Project risks refer to the risks that the project faced in regards to the implementation and development process, these risks could have a variety of impacts on the project so it was important to make sure these were listed out. The table below shows these risks, a description of said risk, the probability that this risk could have happened, the impact this risk could have had, and the mitigation strategy that was followed to avoid these risks.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk Source** | **Risk Description** | **Probability** | **Impact** | **Mitigation Strategy** |
| Game Engine | If the game engine used (Unity) crashed during development | Low | If the project hadn’t been saved then progress could have been lost  Project could have been corrupted if the crash was severe | Saved the project frequently during the development process  Version Control was used to have saved backups in case of corruption |
| Gameplay | Bugs and crashes that would have occurred during gameplay | High | The game wouldn’t work as intended  Some features could have stopped working completely or stopped the game from working at all | Version Control was used to have backups stored in case rollback was needed  Test driven development methodology was used to frequently test for bugs and errors and to fix them once they have been caught |
| Lack of Backups | Losing or deleting of the project file losing any progress that had been made | Low | This could have resulted in a significant loss of progress depending on the last backup that was stored | Version Control was used often to keep frequent backups of the project |
| Time spent on fixing issues | If bugs and errors take longer than expected to fix meaning that less time was available for the rest of the implementation | Moderate | This could have a severe impact on the time management of the project depending on how long it would take to fix the issue | This was considered within the plan which is why test driven development was used and the project split into different phases. This meant lots of testing that was already considered within this projects plan |
| Difficulty Balancing | The game could have been too easy or too hard to complete for the player | Low | Players would be demotivated to play the game meaning this project failed one of its main objectives | Using test driven development there has been constant consistent testing of the game to check for this  Testing was used for using different variables for enemies, bosses, and the upgrades to ensure that good difficulty balancing is achieved |
| Bad Controls | The controls could have been too hard to use and if too many controls are introduced it could overwhelm players | Moderate | This could demotivate players by making the game unplayable as the bad controls could keep the player from performing certain actions as they may be too hard to complete | This project uses the W, A, S, and D keys for movement like most games that are developed for the PC and the same controls are used for the researched games that have a PC version  Due to the constant consistent testing while using the test driven development methodology, tests were designed to make sure the player could perform every action in the game and that they weren’t impossible |

## 3.4 Summary

Detailing the legal issues and project risks meant that this project was less likely to encounter any of these issues or risks during development, and this also meant with fewer complications during the development of this project there was more time for the development of the project itself so the project had more time that could be focused on the project objectives.

# 4.0 Planning the Project

This project was split into multiple phases with each phase containing the mechanics and features to be implemented in that particular phase. This plan was shown in the Gantt Chart created for this project which not only shows the objectives for each phase but also the estimated hours for each objective and the total estimated hours for each phase. This was used during development as a guide to follow for the order in which each objective should be implemented. See Appendix C – Gantt Chart for the Gantt Chart that was created and used in this project.

As part of the planning and designing phase, a game design document was created and is shown in Appendix A - The Game Design Document. This section will detail the planning done for each of the functional requirements for this project.

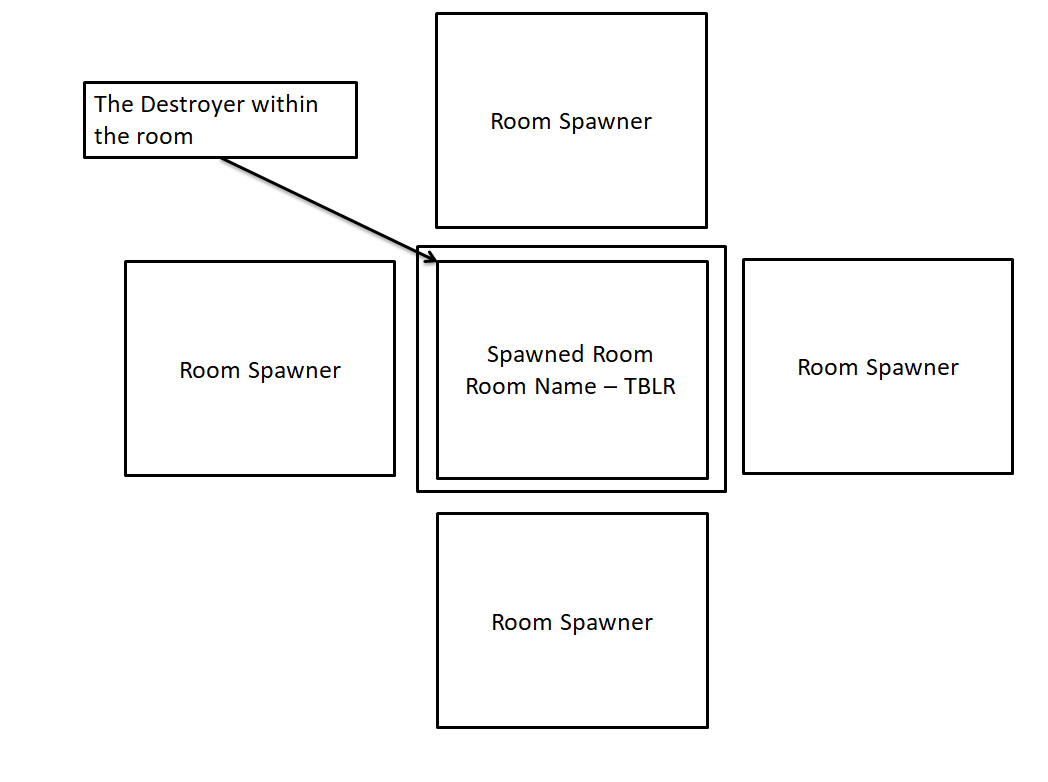
## 4.1 Player Design

The player would be able to move around in two separate scenes, the Hub world and the dungeons the player would try to complete. The player was created to move around with the W, S, A, and D keys as most games used these keys for player movement. The direction the player would shot in would depend on the position of the mouse on screen, the player object would look in the direction of the mouse pointer and when the left click is pressed the gun will shoot bullets in the direction the player is facing.

## 4.2 Design for the Random Dungeon Generation

Through conducting research on algorithms, this knowledge was used to develop a prototype algorithm using a 2D dungeon generation system that would eventually be modified into a 3D system for the final artefact. Using the Fisher-Yates shuffle algorithm and the dungeon spawning system by Blackthornprod (Blackthornprod, 2018), these two where combined with modifications added so that this would work for a 3D system for a prototype random dungeon generator. Through this experimentation, a bug was found where the rooms would spawn and have openings that would lead straight into doors which would not allow the player to go through. To fix this I made a custom script within the experimentation to check the room names, which showed all the openings the room had with a single letter for each room, and then if the checked rooms showed a room that had an opening that lead to a wall then the room would be deleted and a new room would spawn with the specific openings that were needed. After this experimentation was fully developed it showed a full design that could be used for the final artefact with commented scripts and placeholder assets, this code was then later refactored in the final version to get rid of lines that weren’t being used and to follow software development practices. The diagram below in Figure 2.1 shows the plan for how each room object was set up, with spawners to spawn the next room and destroyers to destroyer any spawners that collide with it so no rooms spawn on top of each other.

Figure 2.1 – The Room Plan Diagram



As shown in the above diagram for the TBLR room, all the exits have spawners outside of them. Each room has a room name like TBLR where there is on letter that represents each exit and room is made up of these letters to show the openings/exits they have. The letters are shown below:

* T – This means the room has a top opening
* B – This means the room has a bottom opening
* L – This means the room has a left opening
* R – This means the room has a right opening

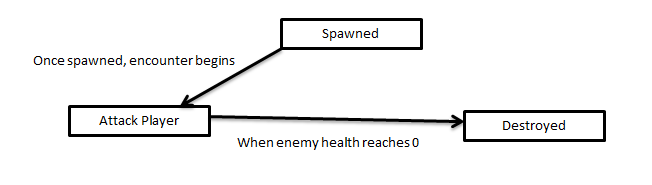
## 4.3 Designing the Enemies and the Boss

There were two types of enemies designed along with one boss as enemies the player defeats in encounters they may encounter while exploring the dungeon. To map the behaviour of each of these I used the diagrams and such found in the research section which includes: Finite-State Machines, and Behaviour Trees.Below is each of the designs of these three enemy types.

4.3.1 Drone Enemy Design

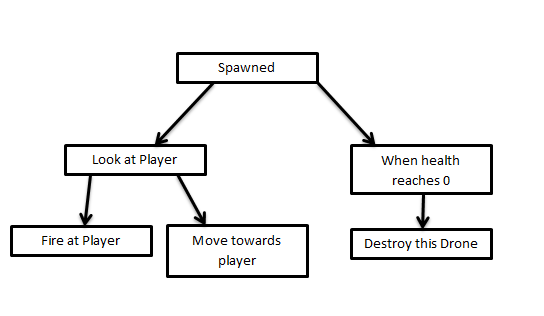
These are the basic ranged enemies that the player can encounter in the dungeons. First a Finite-State Machine was created as the first behavioural plan for this AI as shown in Figure 2.2 below.

Figure 2.2 – Finite-State Machine for the Drone Enemy Type:



This helped when planning out the states for a simple drone enemy the player would defeat multiple of in an encounter, however to plan out the behaviour in more detail a Behaviour Tree was created as shown in Figure 2.3 below.

Figure 2.3 – Behaviour Tree for the Drone Enemy Type:

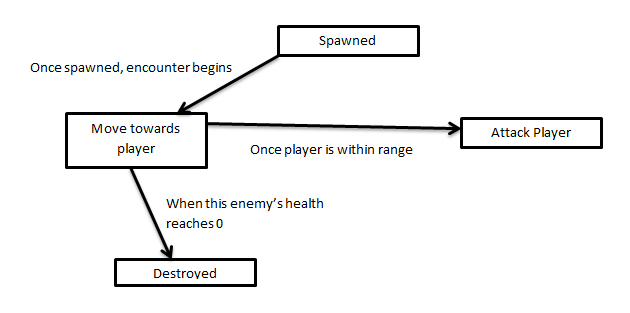


After finishing the behaviour tree, the behaviour for this simple enemy AI was mapped out and would be implemented with little complexity to the behavioural design so the player could easily understand how the AI works in combat.

4.3.2 Head Jar Enemy Design

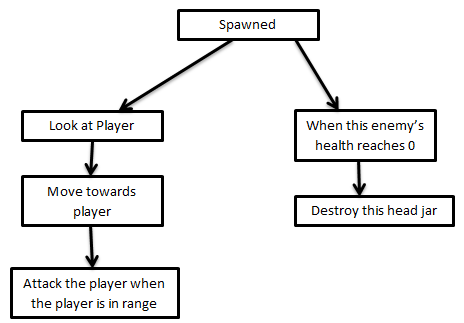
These are the basic melee enemies that the player can encounter in the dungeons. First a Finite-State Machine was created as the first behavioural plan for this AI as shown in Figure 2.4 below.

Figure 2.4 – Finite-State Machine for the Head Jar Enemy Type:



This helped when planning out the states for a simple head jar enemy the player would defeat multiple of in an encounter along with multiple drone enemies. Next was to plan out the behaviour in more detail using a Behaviour Tree as shown in Figure 2.5 below.

Figure 2.5 – Behaviour Tree for the Head Jar Enemy Type:



After finishing the behaviour tree, the behaviour for this simple enemy AI was mapped out and would be implemented with little complexity to the behavioural design so the player could easily understand how the AI works in combat.

4.3.3 Boss Design

This boss was more complicated than the last two AI enemies, so this enemy needed more complex planning. The boss would only spawn as an encounter in the boss room, and with more health and different phases instead of the simple behaviour of the previous two enemy AIs. The created Finite-State Machine and Behaviour Tree are shown in Figure 2.6, and Figure 2.7 below.

Figure 2.6 – Finite-State Machine for the Boss:

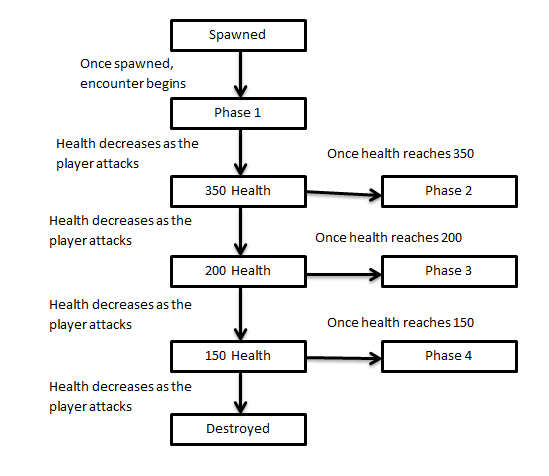
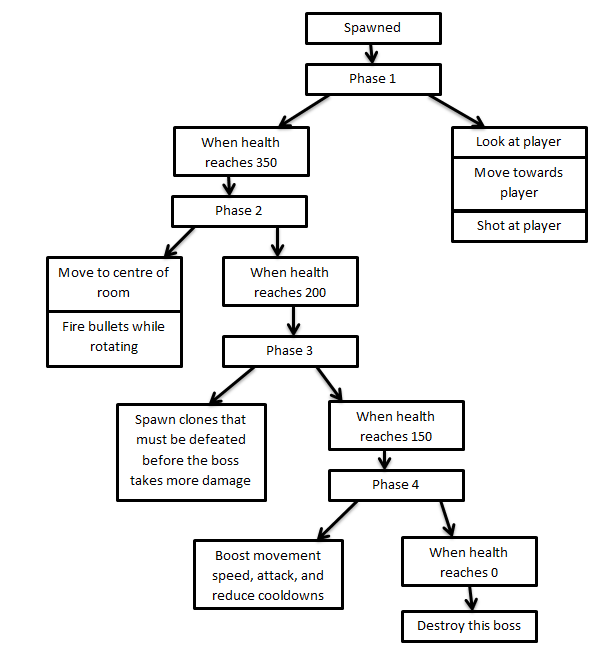


Figure 2.7 – Behaviour Tree for the Boss:



As shown in the above two figures the Finite-State Machine and Behaviour Tree are a lot more complex than the previous two enemy AI designs as the boss has multiple Phases with different effects and conditions. These diagrams helped when developing the boss as a guideline for the boss’s behaviour and to plan when and where each Phase would happen.

Overall these created diagrams helped during the implementation phase as these brought solid designs as to the behaviours of each type of enemy AI that was going to be added to the game. All of these enemies give the player obstacles to face with different levels of difficulty, the drones and the heard jars being spawned together multiple times to form an encounter. These obstacles work towards the Difficulty Balancing main objective to give the player some challenge so completing the dungeon is not too easy.

## 4.4 Designing the Reward Drops

The reward drops would spawn after the player completes an encounter in the dungeon, these reward drops include: health pickups, gold pickups, upgrade chip pickups, hyper core pickups, the boss key, and powerups. This section goes over every one of these reward drops and the design process for each of them.

4.4.1 Health Pickup

Health pickups purpose was to give the player health when picked up, these would spawn after an encounter to let the player gain health before exploring deeper into the dungeon. This works towards the Difficulty Balancing main objective as it gives the player more chances to regain health as they go through the dungeon, however as these do get randomly allocated to rooms there is some luck involved.

4.4.2 Gold Pickup

Gold Pickups purpose was to give the player gold when picked up, these would spawn after an encounter to let the player gain more currency to use in the workshop that spawns in the dungeon. These workshops are used to buy items that will give the player more resources, whether that is health for the dungeon or more hyper cores and upgrade chips for the upgrades available in the HUB World. This works towards the Difficulty Balancing main objective as it gives the player the ability to use the workshop and the benefits it brings as long as the player has the right amount of gold to buy the stock in the workshop. As this is the case the gold helps to balance the workshop so the player is not able to grab loads of resources from the workshop without cost.

4.4.3 Upgrade Chip Pickup

Upgrade Chip Pickups purpose was to allow the player to collect the currency Upgrade Chips that allow the player to buy certain upgrades from the HUB World. The upgrades brought using this currency are the player upgrades and the different weapons. This works towards the Difficulty Balancing main objective as it gives the player the ability to buy player upgrades and different equipable weapons while also making sure the upgrades are balanced and have to be brought instead of the player gaining upgrades for free.

4.4.4 Boss Key

The boss key is used to open the boss room door so the player can defeat the boss and complete the dungeon, which works towards the Clear End Goals main objective by giving the player a goal to complete before taking on the boss. This also ensures that the player has to explore the dungeon before facing the boss which works towards the Difficulty Balancing main objective as this gives the player an obstacle of sorts to complete before they can face off the boss and complete the dungeon adding some more challenge to each individual dungeon. This also means the player cannot go straight to the boss room to defeat the boss and is forced to explore to find this boss key.

4.4.5 Powerups

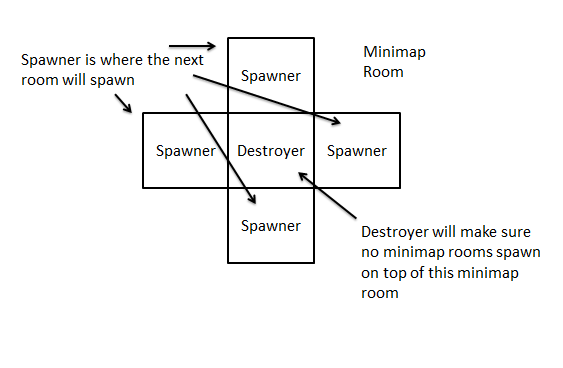
The powerups spawn in the form of pickups that the player can pickup and are designed to give the player abilities which can then be used to help complete the dungeon. The player is only allowed to have two powerups at a time, but can replace the powerups they have with powerups they may find while exploring the dungeon. This works towards the Difficulty Balancing main objective as it gives the players tools to complete the dungeon while also having limitations implemented so the player cannot have too many powerups equipped at once. The list of available powerups is as follows:

* EMP – This powerup allows the player to stun spawned enemies when the space key is pressed which will stop the enemies from moving for a small amount of time. There is a cooldown between each use of the EMP.
* Attack Drone – This powerup gives the player a drone that follows them by their side which shoots enemies when they have spawned in an encounter. The direction this attack drone shoots in is in the same direction as where the player is facing.
* Overdrive – This powerup increases damage and increases movement speed once health is low.
* Shields – This powerup gives the player another smaller health bar in the form of shields that regenerate over time when the player is not in an encounter. If the shields are at 0 then the player’s health will start taking damage.
* Extra Plating – This powerup lowers the movement speed of the player, however it reduces the damage the player takes.

## 4.5 Designing the Minimap

The minimap purpose was to show what portions of the dungeon the player has already explored, which works towards the Clear End Goals main objective by showing the players progress towards the goal of exploring the dungeon with room icons to also show where the entry room, the workshop room, the bot haven room, and the boss room. By showing the progress towards this goal this also gives the player extrinsic motivation to continue exploring the dungeon as it shows their progress so the player gets a sense of progression towards this goal so the player knows when they have completed the goal. Figure 2.8 below shows the plan for each minimap room.

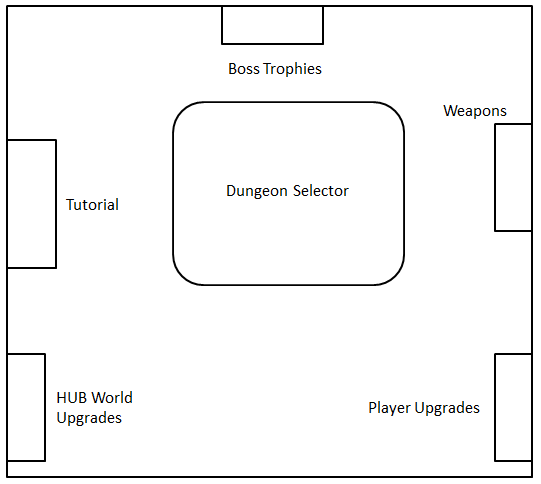
Figure 2.8 – Minimap Room Plan



## 4.6 Designing the HUB World

The HUB Worlds purpose was to be the place for the player to buy upgrades, save the game, and select the next dungeon or sector level to continue playing dungeons and complete the game. This works towards the Difficulty Balancing main objective as it gives the player a place to save and get better tools between dungeon runs, and works towards the Clear End Goals main objective with the dungeon and selector select menus to show how many dungeons and sectors the player has left to complete. Figure 2.9 below shows a plan of the HUB World for this project and what objects are interactable within the HUB World.

Figure 2.9 – HUB World Plan



## 4.7 Designing the HUB World Upgrades

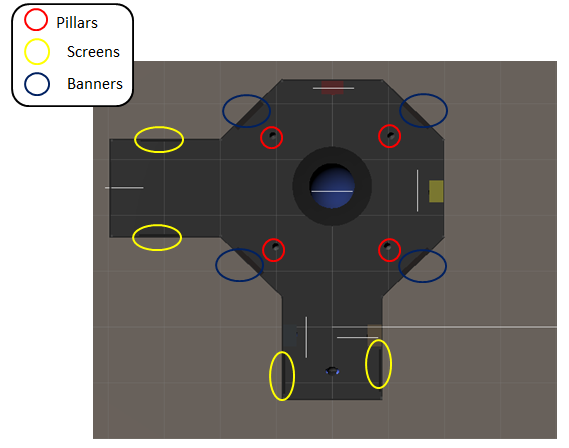
The HUB World upgrades purpose was to give the player cosmetic upgrades that can be brought for the HUB World as more short term goals for the player. This brings more intrinsic motivation as the rewards for completing the goal are cosmetic, meaning the goals are mainly completed for the satisfaction of completing said goal. However there is also extrinsic motivation, as if the player did want one of the cosmetic upgrades for the HUB World then their reward would be said cosmetic item. This works towards the Clear End Goals main objective as it gives the player more short term goals to complete while encompassing some element of the two different types of motivation for the player, as these are optional goals and don’t give the player any abilities or buffs these goals could be overlooked by some players who care more about the rewards that help boost the player and give more abilities so they can progress further in the game, which is why these are only optional cosmetic upgrades while the weapons and player upgrades act as the tools and boosts the player can unlock that help them progress.

The HUB World Upgrades are shown below:

* Pillars that show in the middle of the HUB World
* Screens that show on the walls of the HUB World
* Banners that show on the walls of the HUB World

Shown in Figure 3.0 below is the plan for where the HUB World cosmetic upgrades would show using the HUB World created in Blender for this project.

Figure 3.0 – HUB World Upgrade Cosmetic Locations:



## 4.8 Designing the Save and Load Feature

The save and load feature is the Save Feature main objective, implementing this feature is important as most roguelikes don’t implement this feature even though it allows the player to play the game in multiple sittings without having to worry about losing progress and if the save feature is there but it’s all auto save then that gives the player no control over how they save their game data. This is why I allowed the player to save in the HUB World so the player would have control over their save data along with auto saving at certain parts to make sure the player doesn’t lose the data on an unexpected crash or a similar another situation. Allowing the player to only spawn in the HUB World means that the player has to complete the dungeon run they may be one, regardless of whether that run is a success or not, to save the data themselves, but the data is also auto saved before every dungeon attempt to minimise data lose and the player having to start again. Allowing the player to only save in the HUB World was a Difficulty Balancing choice so the player couldn’t save and load wherever they want in the dungeon as that could have made the dungeons way too easy and it’s why a lot of other roguelikes do not tend to have this feature, however there are ways to give the player the ability to save without making the game too easy.

## 4.9 Designing the Player Upgrades

The player upgrades are obtainable in the HUB World and allow the player to buy upgrades that gives buffs to them so they can continue to complete the game. This gives the player a way to control their development as the game gets harder and harder, while also supplying the player with extrinsic motivation to get these upgrades so they can progress further within the game. This works towards the Difficulty Balancing main objective by giving the player more abilities that can be used during dungeon runs to help balance the player with the enemies and bosses that will continue to grow stronger as they progress through the game.

The different player upgrades are shown below:

* Health Upgrade – Increases max health to 200 instead of 100
* Starter Gold – The player gets 50 gold at the beginning of every dungeon run
* Speed Boost – Increases the players movement speed by 20
* Plus Armour – The player takes less damage from enemy attacks

## 4.10 Designing the Equipable Weapons

The equipable weapons were implemented to be brought in the HUB World so the player can buy different weapons to use in their dungeon runs. These give the player more tools when completing dungeons and works towards the Difficulty Balancing main objective because of this, this also offers up different playstyle opportunities for the player as each weapon works differently so the player must choose the weapon they like the most. These help the player progress through the game with better weapons that can be used as the player gets further into the game.

The different equipable weapons are shown below:

* Pistol – The default weapon for the player. This weapon has a long range with less of a spread of bullets
* Shotgun – This weapon has less range but has more of a spread of bullets, this gun shoots 3 bullets at a time instead of one bullet at a time

## 4.11 Designing the Boss Trophies

The boss trophies were implemented as trophies that were collected once the boss has been defeated by the player, which works towards the Clear End Goals main objective as it gives the player the goal of collecting all the boss trophies as extra motivation to clear the game. This also gives the player a view on how much they have progressed in the game, as the more boss trophies the player would have collected the further they would be into the game.

## 4.12 Designing the Tutorial

The tutorial menu is an important feature for the Clear End Goals main objective as it shows the controls and how the game works for new players. This will then allow the player to complete goals in the game as they can find out how the game works and how to complete these goals. The tutorial menu itself has loads of buttons that when hovered over will show info panels containing information about the relevant topic that has been hovered over.

## 4.13 Designing the Dungeon and Sector Level Menus

The dungeon and sector select menus both allow the player to select dungeons and sector levels respectively, with each sector containing multiple dungeons. This allows the player to complete the long term goal of completing the game while also giving the short term goals of completing dungeons and sectors as part of completing the game. These menus also give the player a view on how far they have progressed along with how far they have left to go, especially the sector menu as when all the sectors have been completed then the game is completed. Because of this these menu features are also very important for the Clear End Goals main objective.

## 4.14 Designing the Different Room Types

The dungeons have four different types of rooms as well as the encounter room that will spawn enemies, these rooms are: the entry room, the bot haven room, the workshop, and the boss room. The entry room represents the start room for the player and where the dungeon begins, this is assigned so that the entry room will not be any other type of room. The boss room will be a room that spawns the boss the player must defeat to complete the dungeon.

The bot haven room and workshop room are only assigned once in the dungeon like the other two rooms types, however unlike the other two types of rooms they can be assigned anywhere. The bot haven room allows the player to choose one out of several options; each option gives the player a certain amount of one resource for free however it can only be used once per dungeon. The workshop has a stock of options which restock when the player buys them all; the player buys each option using gold.

This is an important feature for the Difficulty Balancing main objective which makes sure that the player has a good amount of encounter rooms while also giving the player the bot haven and workshop room to gain more resources at any time. The entry room means that the player won’t have an encounter as soon as they start the dungeon, defeating the player before they have a chance to explore the dungeon. To keep the challenge more of the rooms will be encounter rooms and the boss room will be the room the player will have to defeat to complete the dungeon.

## 4.15 Summary

As this section has detailed how this project was planned along with the designs for all the major mechanics that act as sub objectives, while also detailing why these sub objectives have been implemented within the context of the three main objectives for this project. The next section explains how these sub objectives have been implemented.

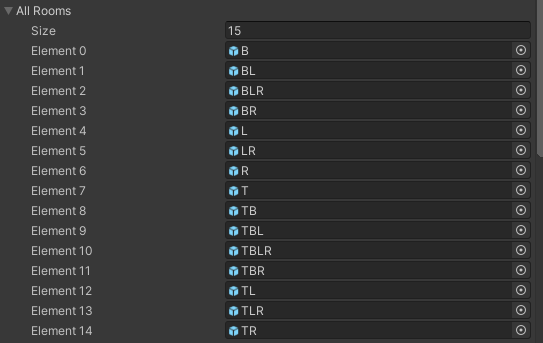
# 5.0 Implementing the Project Objectives

This section goes over the implementation for the sub objectives mentioned in the section above, while explaining how each sub objective was implemented. All the commented code for this project is shown in Appendix B – GitHub Repository for the Project Code.

## 5.1 Random Dungeon Generation

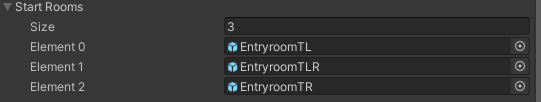
Before the room is spawned, the room that would be spawned must be chosen through randomly picking an element in an array. The array that is being used is in the RoomTemplates script with the room prefabs being added through the inspector as shown below in Figure 3.1.

Figure 3.1 – The Array of Rooms:



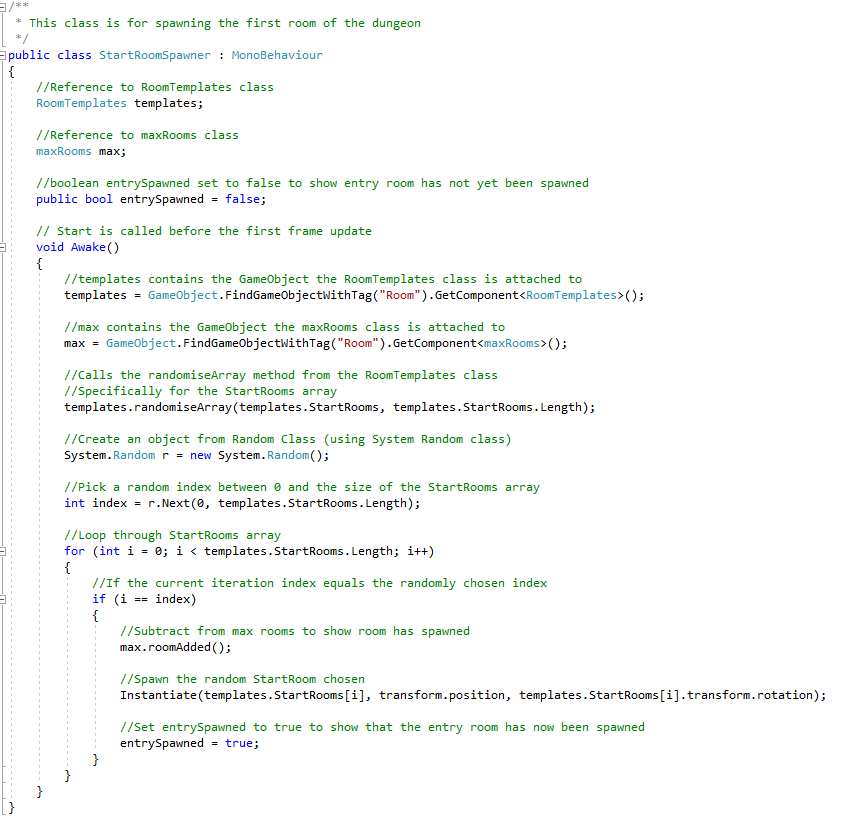
To spawn the start room as the entry room a different array was used before the above array got used by the rest of the room spawners. The start room array used is also in the RoomTemplates script within the RoomTemplates object that was set with the room prefabs in the inspector. This is shown in Figure 3.2 below.

Figure 3.2 – Array of Start Rooms:



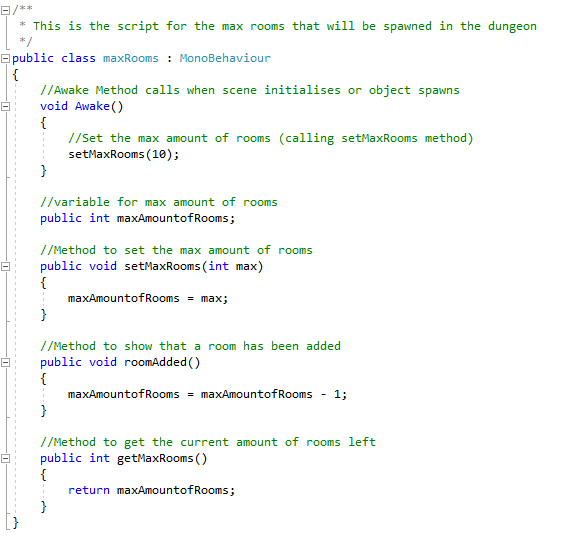
A Startroom\_Spawner empty object was created to spawn this entry room using the start rooms array by using the Random object in the System namespace of C# to generate a random index and then using that index the code loops through the start rooms array in RoomTemplates and spawns the entry room. This is done in Unity’s Awake method which is called before the Start method which both calls on the first frame the object has been created or set active. This is shown in Figure 3.3 below.

Figure 3.3 – Start Room Spawner Code:



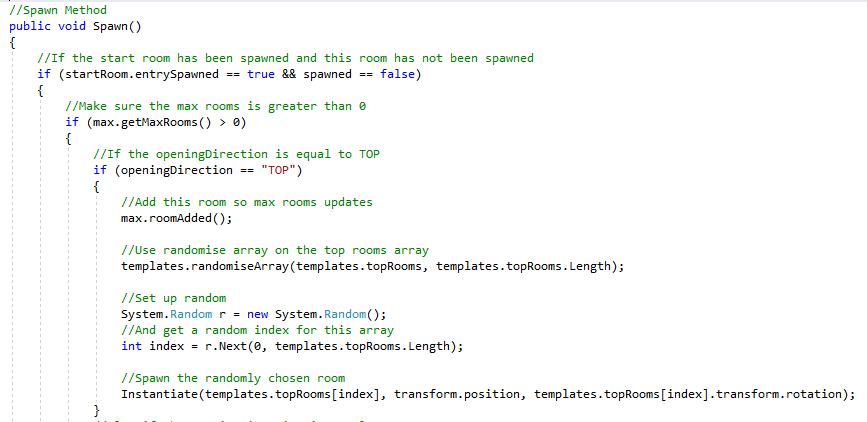
The Instantiate used is for spawning the rooms. The max rooms shown in the code screenshot above keeps count on the max amount of rooms spawned before the dungeon should spawn closed rooms to close off the dungeon. This script is added in the RoomTemplates object and uses an int variable to count how many rooms are left to be spawned before the dungeon spawns the closed rooms to close off the dungeon. The code for this max room script is shown in Figure 3.4 below.

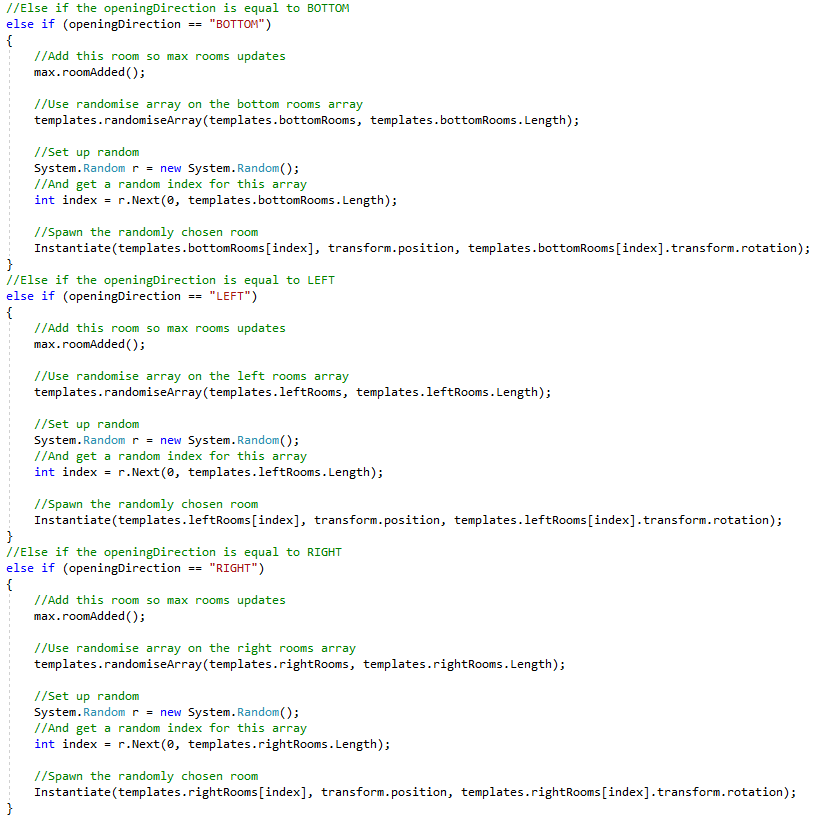
Figure 3.4 – The Max Rooms Script:



Once the start room is spawned the rest of the rooms spawn using room spawners that are part of the room prefab which get destroyed once a room has been spawned or when the spawner collides with the destroyer in the middle of the room so the rooms don’t spawn on top of each other. Also to this end when two spawners collide and neither of them have spawned a room then one of those spawners will have spawned set to true so the spawner does not spawn a room and will get destroyed when a room spawns due to the destroyer. The rooms will continue to spawn in this fashion until the max amount of rooms have been spawned, in this case the max rooms are 10, and will start working on closing off the dungeon by calling the close dungeon method instead of the spawn method. The close dungeon method will find the closed room that needs to be spawned based on the openings that do not have rooms spawned and will spawn the correct closed room. Figure 3.5 and 3.6 show these methods and their commented code below.

Figure 3.5 – The Spawn Method:





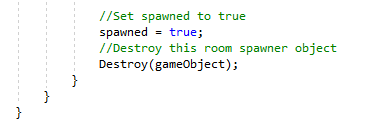
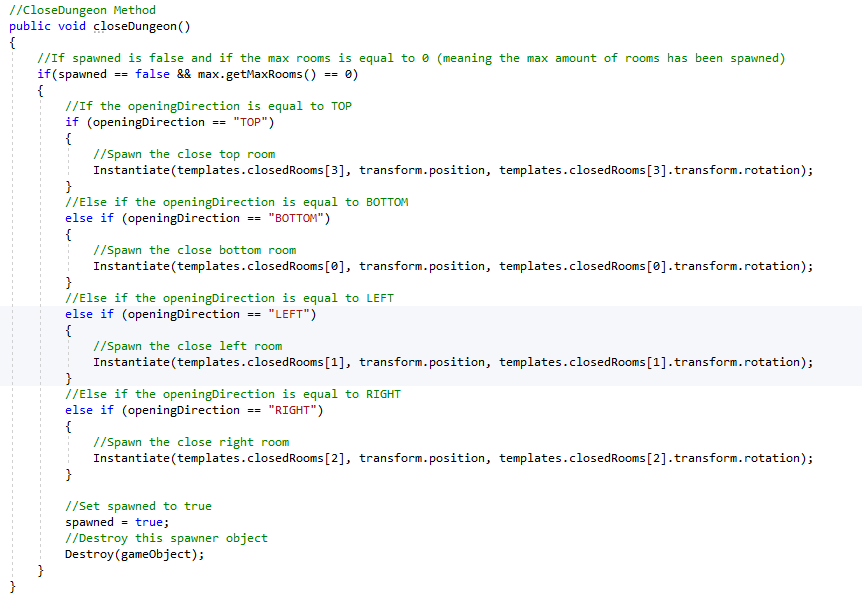
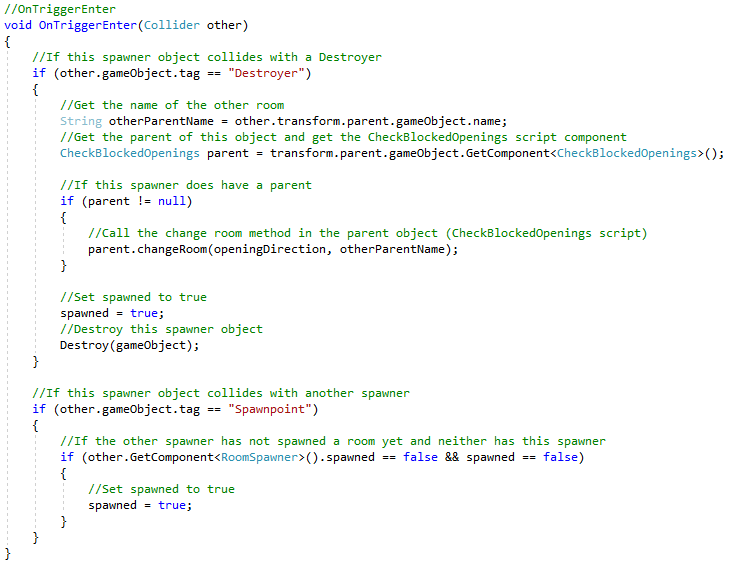


Figure 3.6 – The Close Dungeon Method:



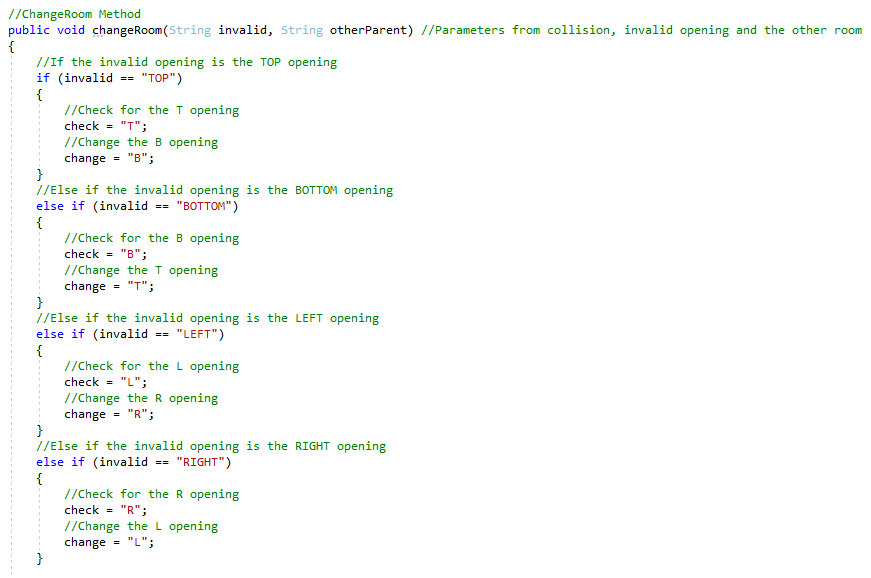
When a spawner collides with a destroyer a method needed to be called from a script that was also attached to the room to check whether the room’s openings are attached to other room openings and not to walls that do not allow the player through. The code for detecting this collision is shown below in Figure 3.7.

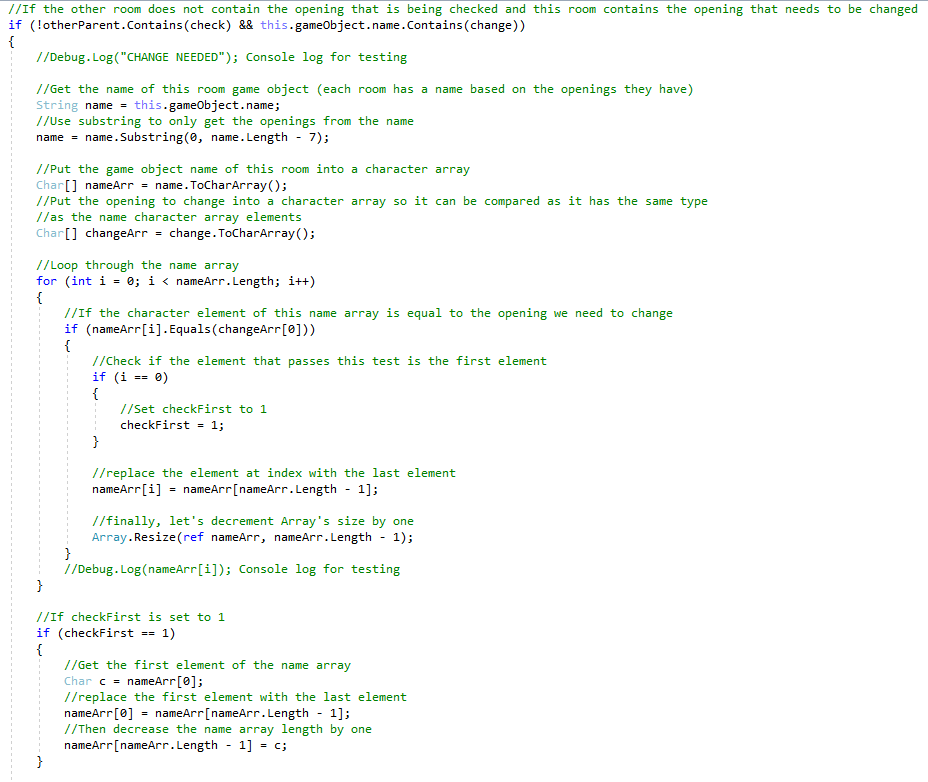
Figure 3.7 – Checking the Spawners Collisions:

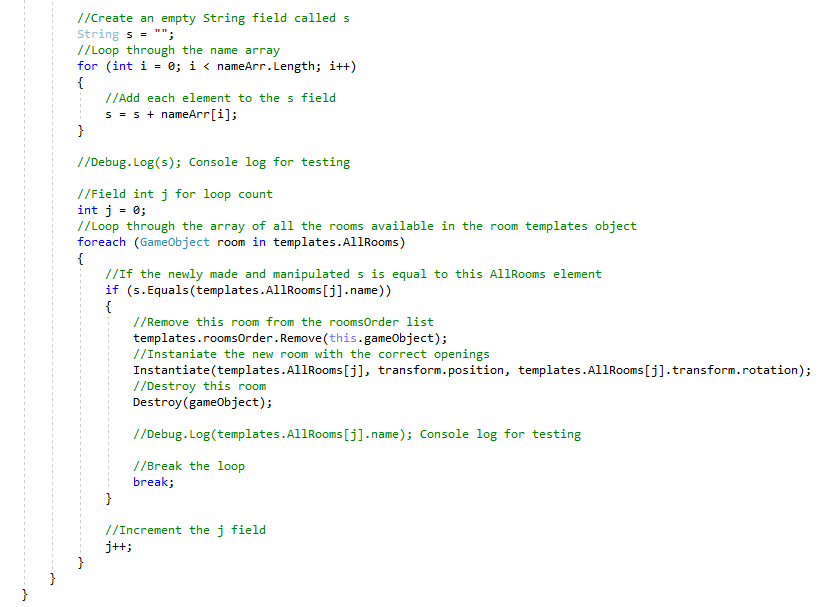


The spawn point collision shown above is to check that rooms don’t spawn on top of each other at the same time, so if two spawners collide only one spawner should spawn a room. However all this code lead to a bug where openings for rooms would lead to a wall the player could not get through so the CheckBlockedOpenings script was made and attached to the room objects. The CheckBlockedOpenings script has a changeRoom method that checks the direction of the spawner that collided with the Destroyer and the name of the room the Destroyer is a part of as a child object. Using this the changeRoom method checks which exit should change and what it should change to and will the proceed to create a String name of the room that should be spawned, spawn it, and destroy the old room whose child spawner object collided with the destroyer so there’s no room being spawned on top of another room. The code for this method is shown in Figure 3.8 below.

Figure 3.8 – Change Room Method





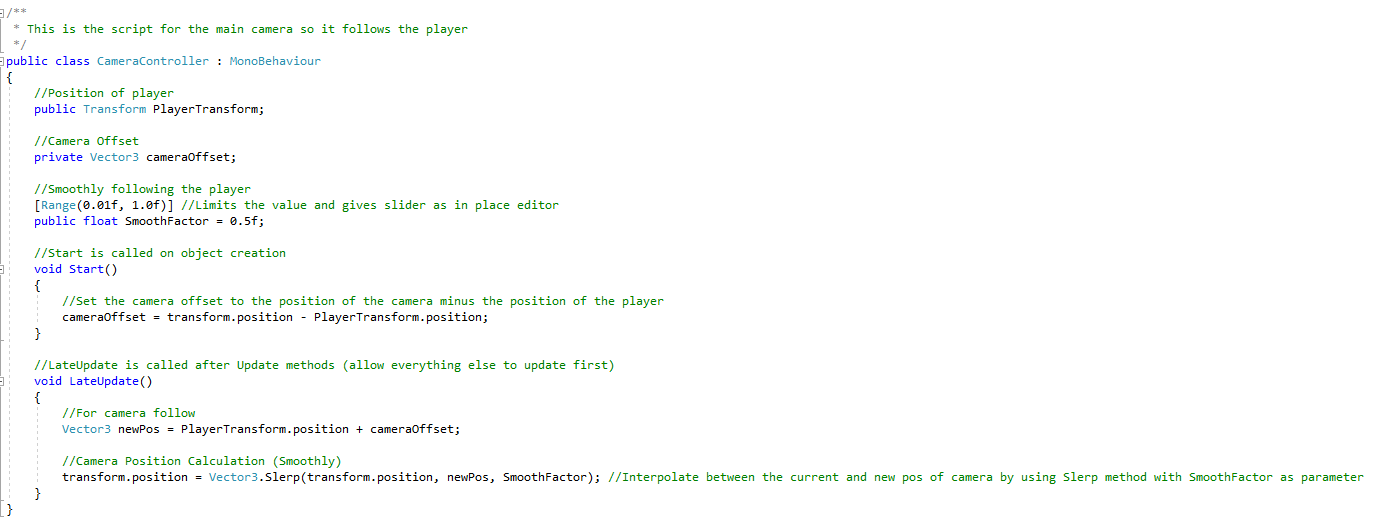


Having different scripts for these different methods means these scripts are more cohesive and stick to their specific tasks and this also makes the scripts more maintainable as the code for each is easier to find as their in their own scripts.

## 5.2 The Player Objects

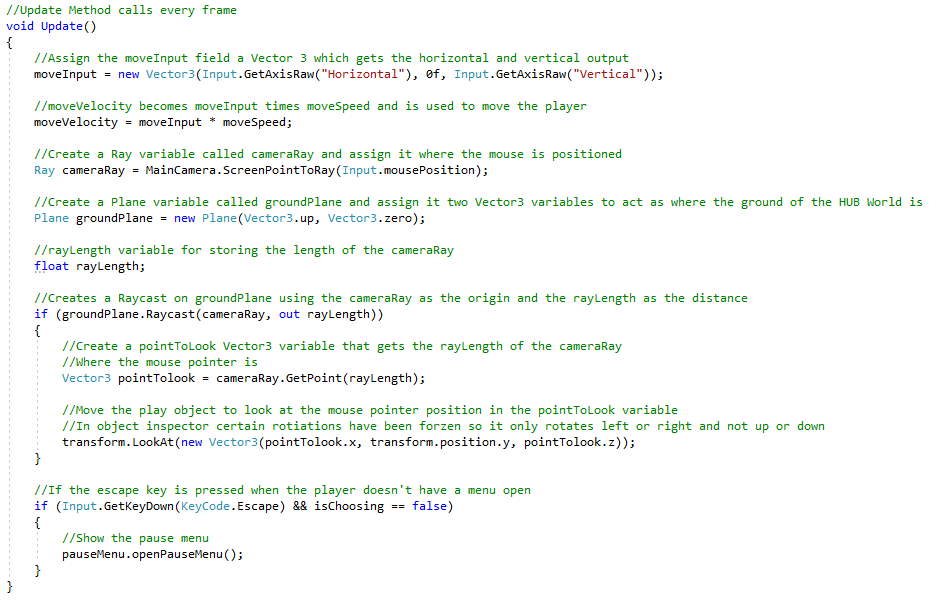
There are two players’ objects, one for the HUB World and one for the Dungeon player. They both have the same movement system and the same save and load data methods, but the dungeon player also has code for spawning the equipped weapon, code for adding the player upgrades, and code for the powerups with implementation on how they are activated. The save and load methods that call the relevant methods in the SaveSystem script are called in the player as the data that is always loaded into the player as the player contains all the variables being saved. There is a camera controller script which is used for both The HUB World and the Dungeon which makes sure the camera follows the player as they move, the code for this is shown in Figure 3.9 below.

Figure 3.9 – The CameraController Script:



The dungeon player also has a current room variable that shows which room the player is currently in so the minimap can spawn properly and also show the red dot on the minimap that visually shows in which room the player is in currently. The movement of the player in the dungeon and the HUB World have the similar code for moving the player; this is shown in Figure 4.0 below.

Figure 4.0 – Player Movement:



The Figure above shows the movement code in the HUB World, the difference between the movement code for the HUB World player and the dungeon player is that the dungeon player has loads of checks for checking for player upgrades brought and powerups which manipulate the movement speed of the player. This reason the movement itself was split into the two different scripts and not reused for both is because of the amount of checks the dungeon player controller has to do which can change variables like movement speed for the player movement, so for these manipulations to take place both player objects needed the movement code in separate scripts.

The camera control also had to be in a separate script so it could be attached to the camera object and adding camera control to the player would make the player script less cohesive and harder to maintain.

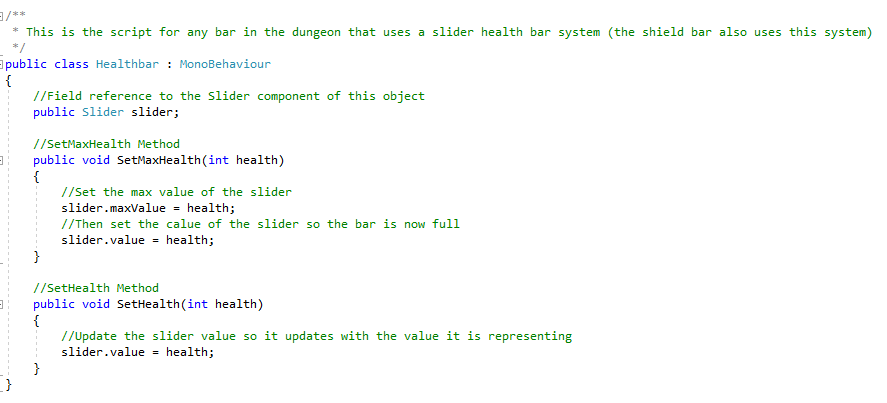
## 5.3 Adding the Enemies and Boss to the Dungeons

The enemies developed in this project have two different types called Drones and Head Jars, and the boss has only on type of boss implemented. The details on the process of implementation for each are listed below.

5.3.1 Drone and Head Jar Enemies

The drone is a slow enemy that shots bullets at the player, the bullets that would be shot are spawned from the Firepoint child object along with a Canvas child object which holds the enemy health bar that shows over the Drone and Head Jar enemies. The healthbars themselves work as a slider with methods that update the slider value that are called when the enemy health changes. The healthbar script used for both the Drone and Head Jar enemies are the same and is shown in Figure 4.1 below.

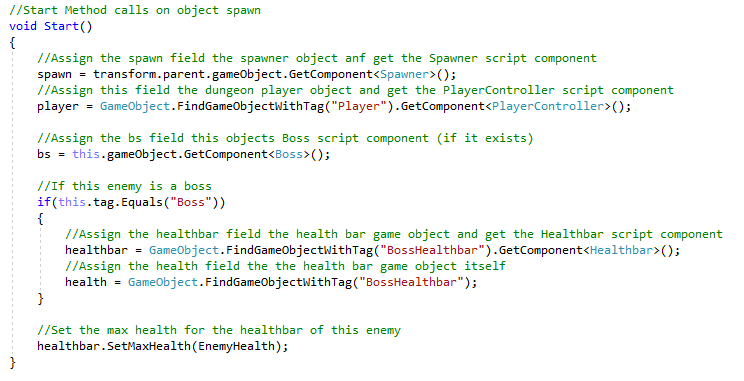
Figure 4.1 – Healthbar Script:

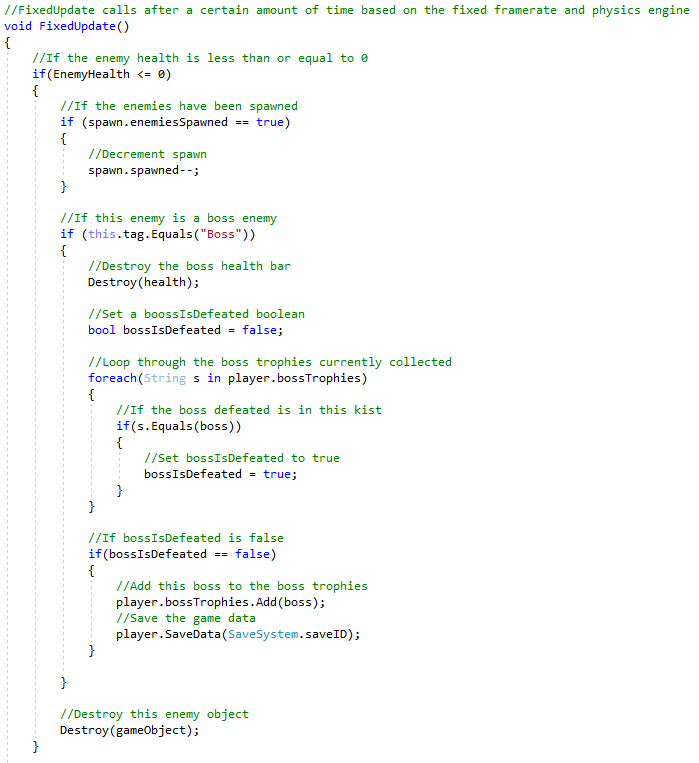


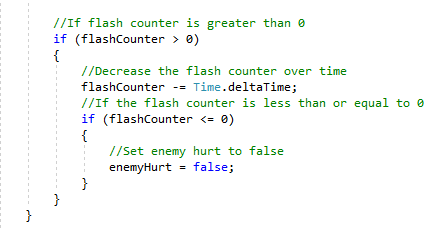
This healthbar script is also used for the Boss health, the player health, and the Shield bar for the Shields powerup the player can collect during their dungeon run. More details on these specific features will be shown in their own section.

The Drone and Head Jar enemies also share the same Enemy and EnemyController scripts with the different being the values that are edited in the inspector. This means that these scripts are scripts that can be used for multiple different enemies and making it easier for more enemy types to be added in the future along with better maintainability as the cohesion of each script focuses on different specific tasks. The Enemy script focuses on the enemy’s health and when that health value gets manipulated, along with managing the red flash that appears when the enemies with the flash script itself being in the GFX objects of the enemy as those are the individual shape objects that make up the enemy object. The EnemyController script focuses on the movement and speed of the enemy along with checking whether or not the enemy has been affected by the EMP as it stops them from moving, therefore affecting their movement. Both commented scripts are shown below in Figures 4.2 and 4.3.

Figure 4.2 – The Enemy Script:







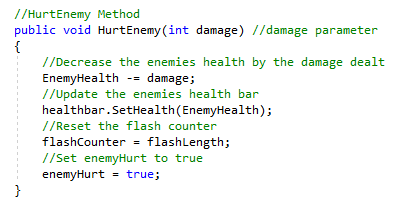
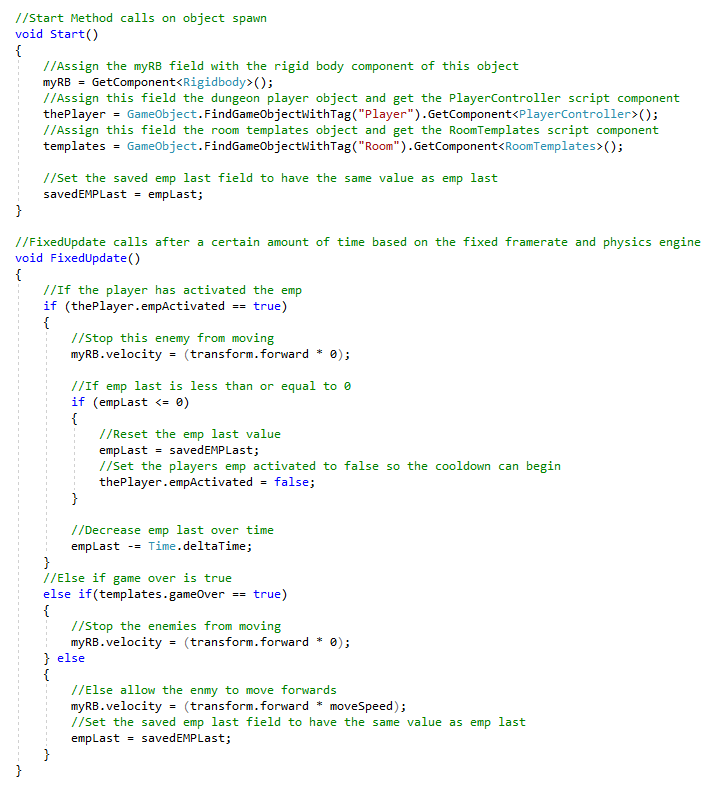
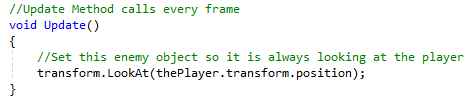


Figure 4.3 – The EnemyController Script:



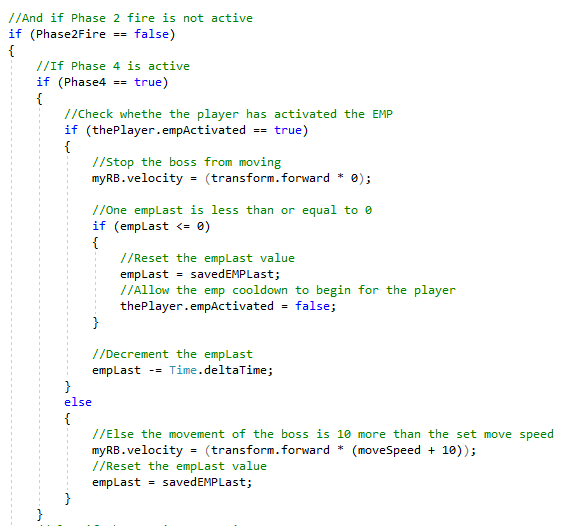


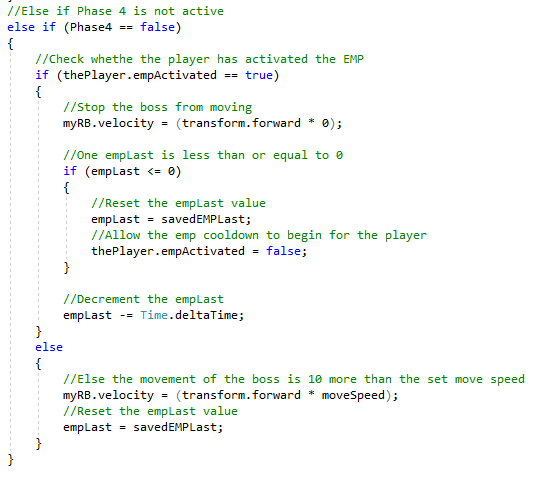
Having these scripts used by all the different types of enemies meant that these scripts were reusable for every type of enemy, so adding more types of enemies would be a lot easier. The scripts being split up into three instead of putting them all in one added to the cohesion of the scripts and also meant the Boss could use the Enemy script for controlling the boss’s health.

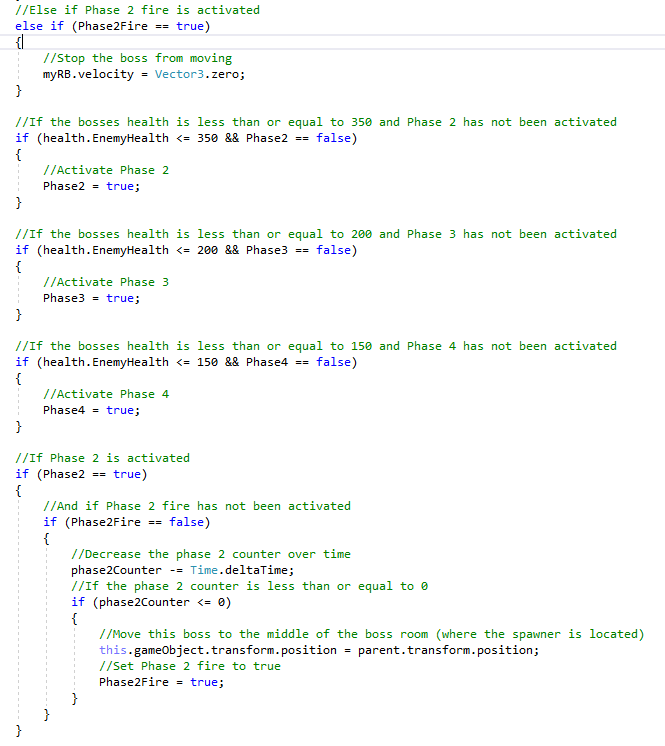
5.3.2 Implementing the Boss

The boss uses the Enemy script that was shown in the section above with the Drone and Head Jar enemies to control the bosses health, however the Boss object does not have the EnemyController script but the Boss script instead to not only determine the movement speed and how the boss moves but also how each of the individual phases of the boss work and how they are activated. The bullets fire out of Firepoints like the other enemies, however the boss has Firepoints surrounding the boss with the three at the front being the front facing Firepoints meaning that these Firepoints will always fire whereas the rest will only fire when the actions in Phase 2 are activated. There are also clone spawners which are child objects to the boss object which are spawners that spawn the clones when the actions in Phase 3 are activated. Figure 4.4 below shows the code for each phase of the Boss which is contained within a FixedUpdate Unity method so that these checks are made continuously.

Figure 4.4 – Boss Phase Code:







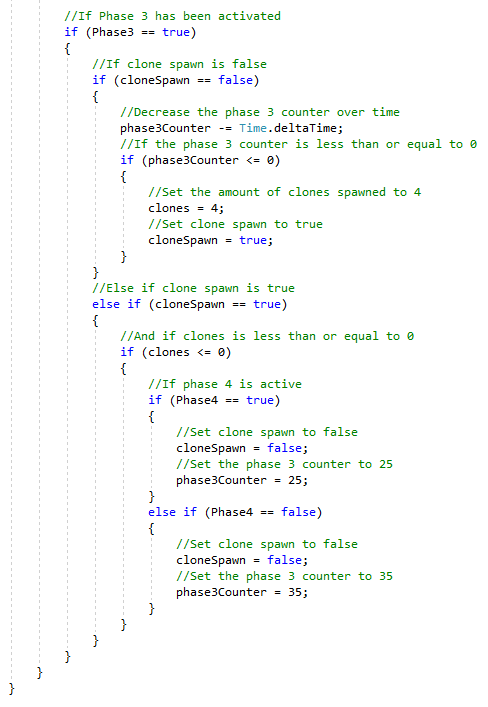
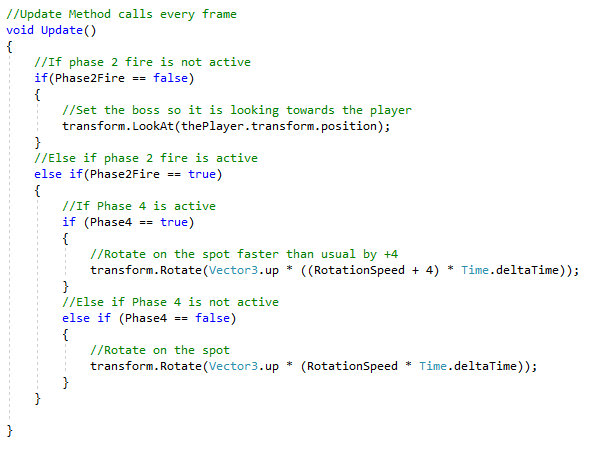


Figure 4.5 below shows the code written for controlling the boss’s rotation and they rotate, whether that be from following the player or in Phase 2 when the boss rotates in the middle of the room and fires bullets as they rotate.

Figure 4.5 – Boss Rotation Checks for Phase 2:



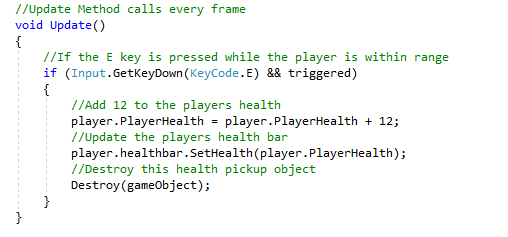
The Boss had to use a new Boss script instead of the EnemyController script as the behaviour of this boss was vastly different compared to the other types of enemies which meant this script had cohesion as it was only manipulating the behaviour of the boss and not the other types of enemies as well. This also makes both the Boss and EnemyController scripts more maintainable as they both are easier to understand.

## 5.4 Adding the Reward Drops

Each reward drop acts like a pickup in the sense that the player can interact with them to gain some effect then they are destroyed, and these reward drops would only spawn on a finished encounter meaning all the enemies within a room have been defeated. All these rewards also have a tooltip that shows when the player is within range to interact with this object. The different reward drops implemented are: health pickups, gold pickups, upgrade chip pickups, hyper core pickups, the boss key, and powerups.

5.4.1 Health Pickups

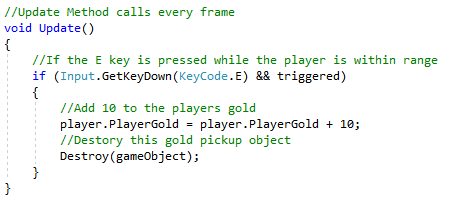
When interacted with this pickup gives the player plus 12 health. The code for this pickup is shown below in Figure 4.6.

Figure 4.6 – Health Pickup Script:  


5.4.2 Gold Pickups

When interacted with this pickup gives the player plus 10 gold. The code for this pickup is shown below in Figure 4.7.

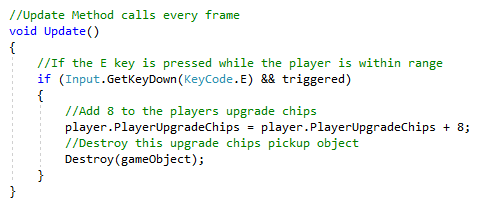
Figure 4.7 – Gold Pickup Script:



5.4.3 Upgrade Chip Pickups

When interacted with this pickup gives the player plus 8 upgrade chips. The code for this pickup is shown below in Figure 4.8.

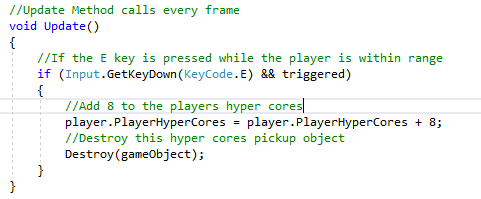
Figure 4.8 - Upgrade Chip Pickup Script:



5.4.4 Hyper Core Pickups

When interacted with this pickup gives the player plus 8 hyper cores. The code for this pickup is shown below in Figure 4.9.

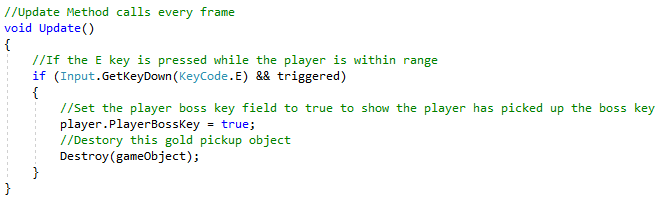
Figure 4.9 – Hyper Core Pickup Script:



5.4.5 The Boss Key

When interacted with this pickups gives the player the boss key so they can open the boss room door and challenge the boss. The code for this pickup is shown below in Figure 5.0.

Figure 5.0 – Boss Key Pickup Script:



5.4.6 Powerups

The powerup pickup when interacted with gives the player the choice of two different powerups randomly selected from a randomly generated stock based on the String array of powerups in the RoomTemplates script. The array of powerups is shown in Figure 5.1 below.

Figure 5.1 – Powerup Array:



Using this array the Powerup Stock script in the PowerupMenu object in the dungeon scene generates a new random stock and then spawns the menu whenever the player interacts with the powerup pickup, If the player already has 2 powerups then the replace menu will open when one of the options is clicked giving the player the ability to switch out on of their current powerups for this new one. The max amount of powerups the player can have equipped at a time is two.

If a powerup pickup is left and a powerup is not changed then the generated stock that existed when the player interacted with the powerup pickup would save onto the pickup and if the player got a powerup later on that was the same as one of the powerups in a spawned powerup pickups generated stock then the powerup pickup would wipe their generated stock so a new one can be generated when the player interacts with that pickup again. Figure 5.2, Figure 5.3, and Figure 5.4 below show three methods that are within the PowerupStock script, that work with the generation of the powerup stock.

Figure 5.2 – The Check Stock Method:

This method checks if the powerup pickup that is interacted with already has a stock, and if it does then use that when opening the menu.

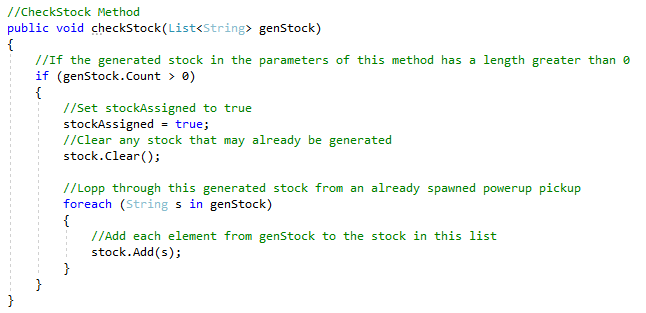


Figure 5.3 – The Add Stock Method:

This method generates a random index and calls the Choose Powerup method to use that randomly generated index to choose a valid powerup before adding it to the powerup stock list that shows the generated stock for the powerups.

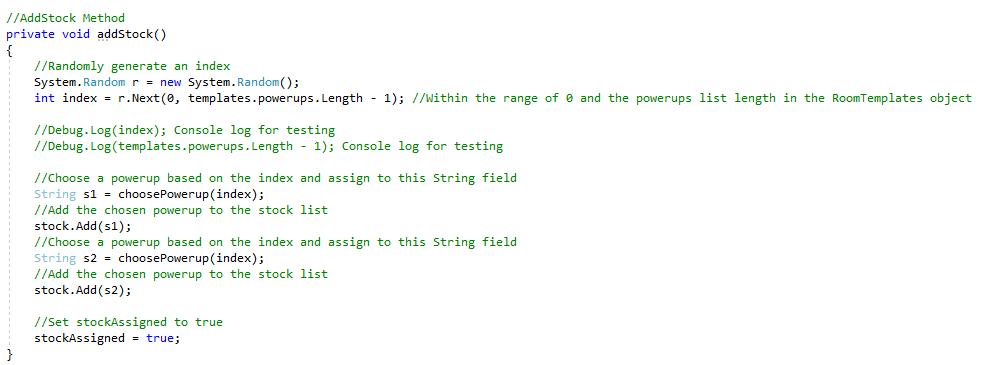
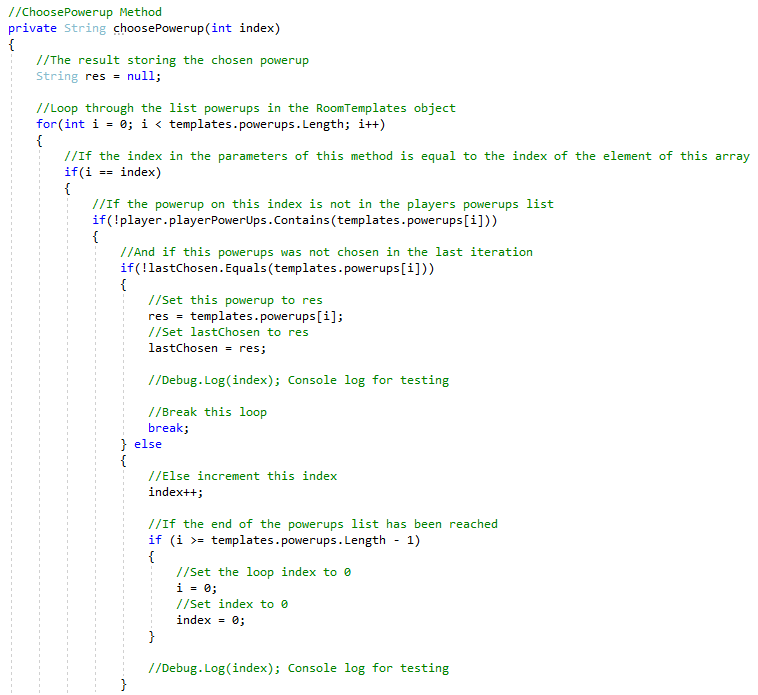
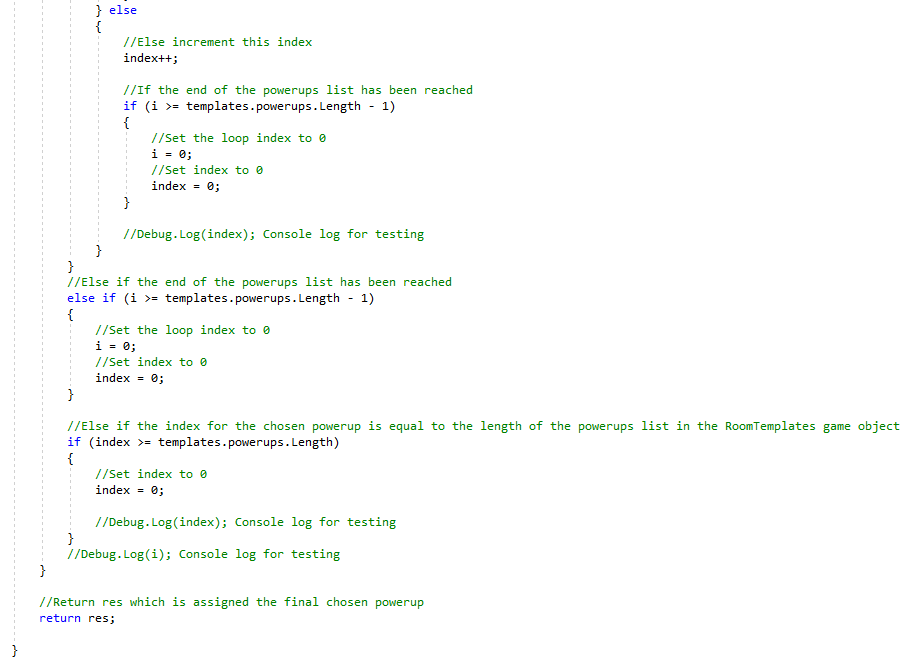


Figure 5.4 - The Choose Powerup Method:

This method chooses a valid powerup that can be added to the generated powerup stock.



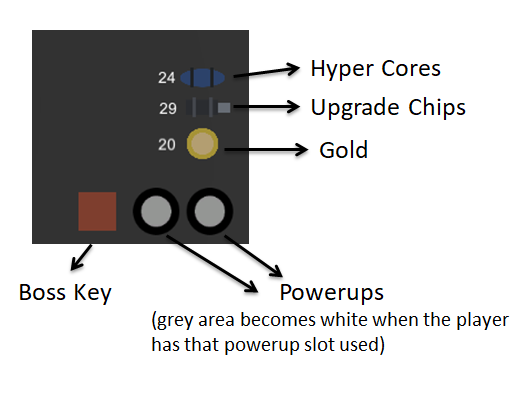


The player would gain the powerup they click in the powerup menu, and would replace it if the player had chosen a powerup they already had to replace it with if it was needed.

5.4.7 The Reward HUD

There is a HUD for the different collectable rewards that show how many of a reward the player has or appears/changes sprites when the player has that reward. The HUD is shown below in Figure 5.5 below.

Figure 5.5 – The Reward HUD



5.4.8 Summary

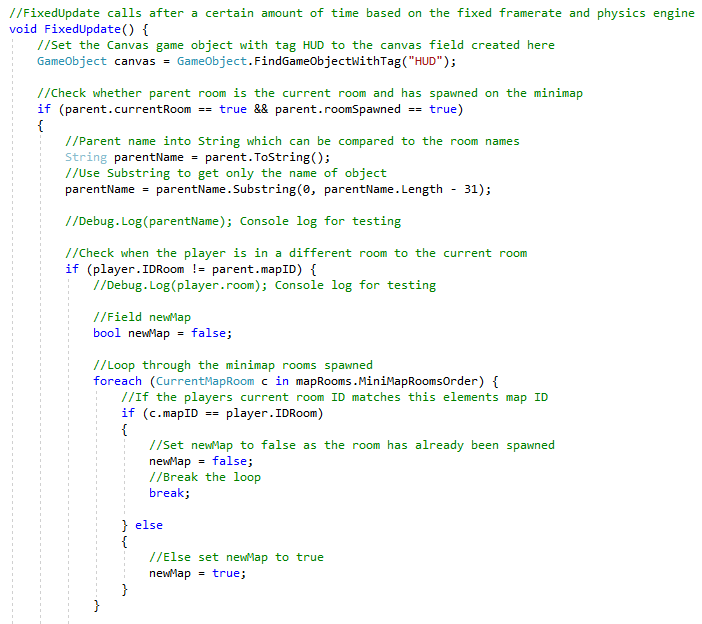
The scripts for all the pickups had to be separate as they all affected the player in different ways; however the health pickups, gold pickups, upgrade chip pickups, hyper core pickups, and the boss key scripts could have been merged together though the scripts wouldn’t be as cohesive as they would deal with multiple objects that affect the player differently. However as the code for each object wasn’t that big these scripts could have been merged in a way that shows cohesion, as I learnt later on during the development of this project.

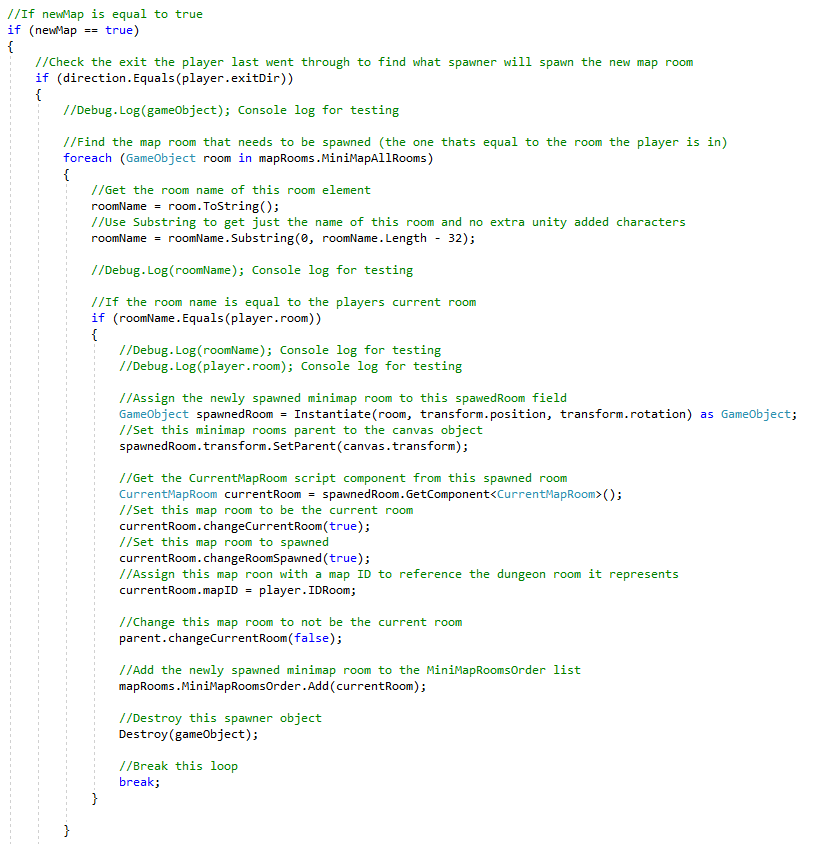
The powerups scripts had to be separate as their behaviour was vastly different from the other pickups, these scripts being split up to different features of the powerups mechanic to keep the powerups easier to maintain as they are easier to understand with these separate scripts.

## 5.4 Implementing the Minimap for the Dungeons

The minimap spawn in the dungeon by checking what current room the player is in and which direction is the exit the player went through last so the minimap rooms can be spawned in the right direction and by the right room. Each room is assigned an ID so the game knows which minimap room represents which room and that is how the current room is checked, along with checking the type of the current room so the right room icons can spawn when the room spawns. Checking the current room of the player allows the player allows the player icon to appear in the right room on the minimap too. The script for spawning the rooms is in the MapRoomSpawner script shown in Figure 5.6 below.

Figure 5.6 – MapRoomSpawner Script:





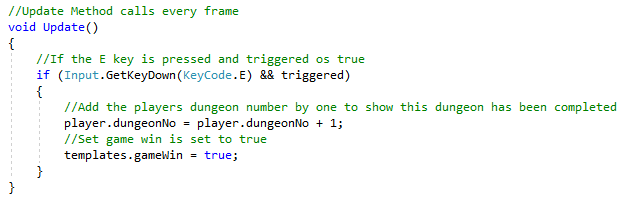
This code spawns rooms based on the current room of the player and if that current room changes then the next minimap room will spawn. The current map room is stored in the CurrentMapRoom script of every minimap room so the game knows where to spawn the player icon and where the next room should be spawned on the minimap. The current room stored in the dungeon player script is used to spawn the next room as when the player moves to a different room not only will the new minimap room spawn but the current room the player is in will update, so the game uses the room name of the current room stored in the player to spawn a minimap room representation of that room and then assign that minimap room the relevant roomID of the current room to the mapID of this current minimap room.

Keeping the minimap code separate from the room spawner code meant that both scripts would be easier to maintain and would keep cohesion for all the scripts for both mechanics, this was the easier way to implement the minimap as it depended on the player and not the rooms spawned and would make little sense to have both the room and minimap spawner features in the same scripts.

## 5.5 Implementing the HUB World

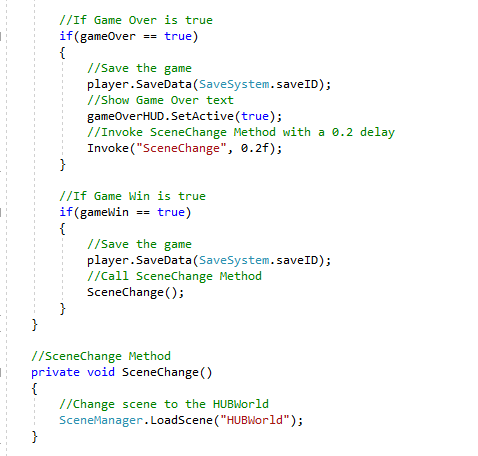
The HUB World contains the upgrades, buyable and equipable weapons, boss trophies, the tutorial, menus to choose the dungeon and sector, and a pause menu for the player to save. The HUB World is a separate scene to the Main Menu scene and the Dungeon scene, so the dungeon selector is used to move to the Dungeon scene as explained in a separate section below. The dungeon exit in the Dungeon that spawns when the player beats the boss will take the player to the HUB World when interacted with and the player will also be taken to the HUB World on game over. Figure 5.7 below shows the DungeonExit script.

Figure 5.7 – DungeonExit Script:



This part of the script shown the switch between the Dungeon and HUB World scenes as it changes the gameWin to true in the room templates, with the player changing gameOver to false when the players health reaches 0. The actual scene change is in the RoomTemplates script as shown below in Figure 5.8.

Figure 5.8 – RoomTemplates Script:

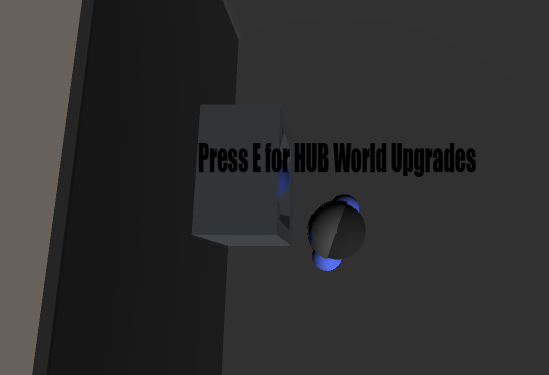


Having RoomTemplates check for the game win and game over conditions meant that this would be done outside any object and as this object was always active it meant that these checks would not be interrupted for any other features or mechanics. The dungeon exit being separate to the game win and game over made more sense than putting both features together as it allowed for both scripts to be cohesive and make them easier to understand because of this.

## 5.6 Adding the HUB World Upgrades

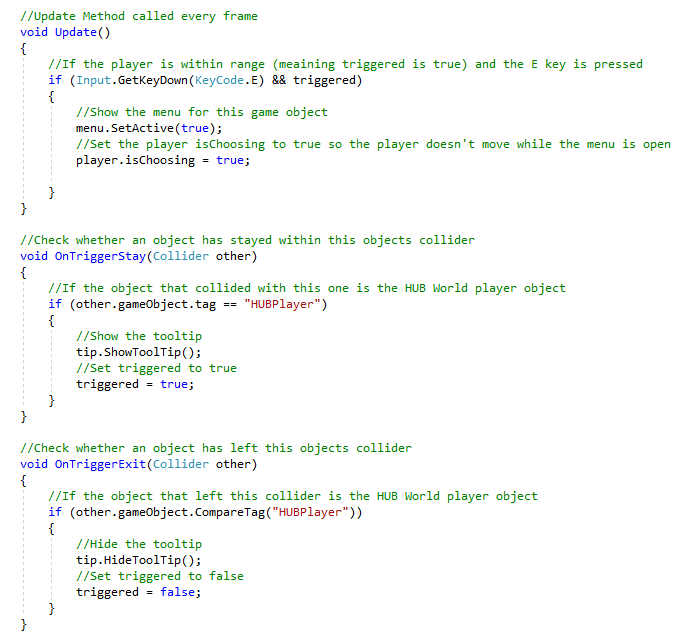
An object was added into the HUB World which the Player can interact with to open the HUB World Upgrade menu so the player can choose which upgrade they want to buy if they have enough hyper cores as shown in the reward HUD in the HUB World. The object the player must interact with is shown below in Figure 5.9.

Figure 5.9 – The Object for HUB World Upgrades:



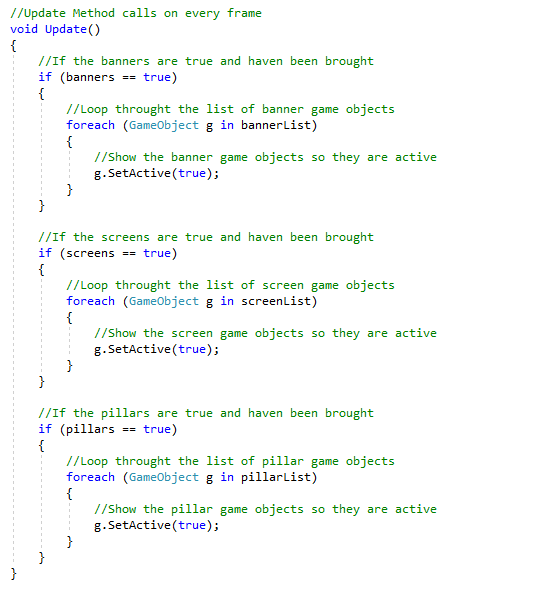
To interact with this object and every other intractable object in the HUB World the MachineTrigger script was created to not only show the tooltip when the player is close enough but also opens the menu when this object is interacted with. The code for this in the MachineTrigger script is shown below in Figure 6.0.

Figure 6.0 – Object Trigger Code:



The HUB Upgrade Menu manipulates three bool fields in the HUBUpgradeM Script that each represent a type of cosmetic object that would spawn in the HUB World if the HUB World upgrade was brought. These are checked in this script within an Update Unity method and if any of them do turn true then the cosmetic objects will be set to active using the fields that their stored in within this script. This script is attached to the same object as the MachineTrigger object, the code for checking these Boolean values in the HUBUpgradeM script is shown below in Figure 6.1.

Figure 6.1 - Activating the Upgrades:



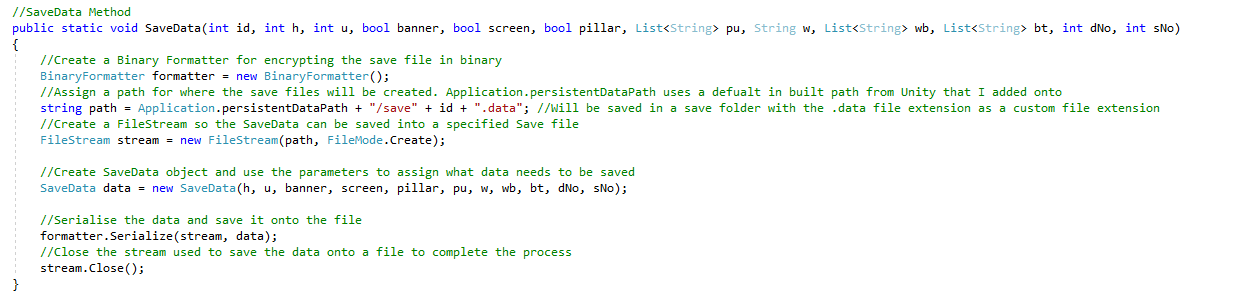
Having the script for spawning the upgrade menu and activate the upgrades separate kept both scripts cohesive and easily maintainable, and putting them both on the same object had to be done as these scripts needed to be on an object that would continuously be active to carry out the checks needed. It made more sense for these scripts to be attached to the HUB World Upgrades object as the scripts are for the HUB World Upgrades keeping the objects functionality cohesive as well.

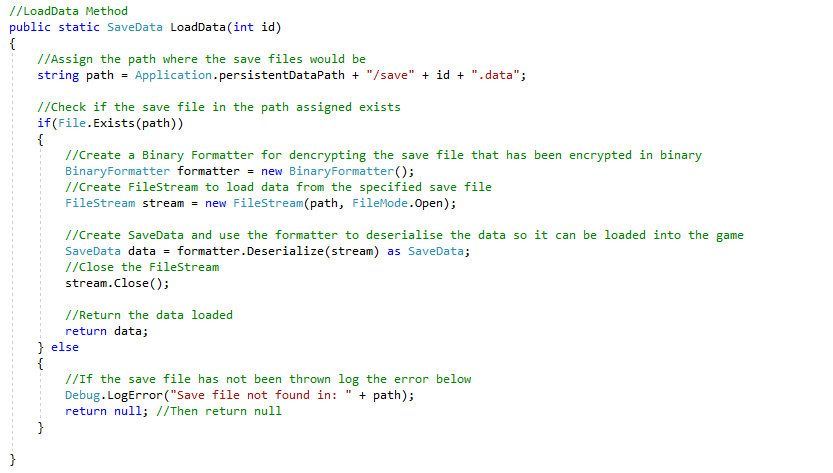
## 5.7 The Save and Load feature with Three Available Save Slots

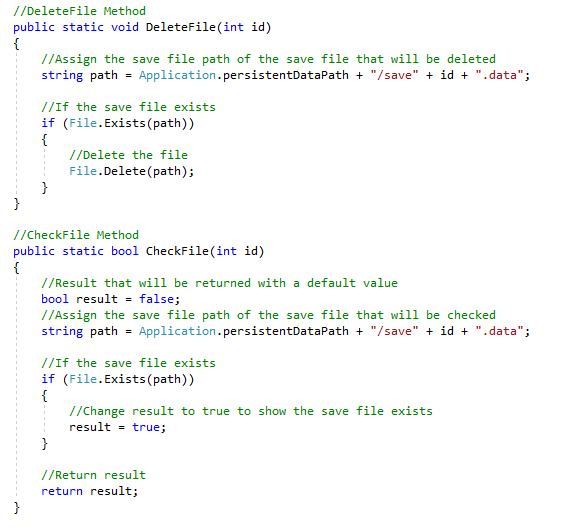
Inspiration was taken from Brackeys (Brackeys, 2018) save and load system as the save files where encrypted in binary to a custom file type, however Brackeys (Brackeys, 2018) save system only used one save slot and the system for this project needed three so I modified Brackeys (Brackeys, 2018) existing system with a save ID and different data that was being saved from the player objects which is why the save and load methods are in both the player object scripts in the dungeon and HUB world scenes as the data being saved and loaded would all be in the player objects.

The SaveData and SaveSystem classes do not implement MonoBehaviour so they can be referenced without having to be attached to an object, with the SaveData being a serializable class so the data of this class can be written to a file with the SaveSystem having methods to save, load, delete, and check save files. The SaveSystem class has a saveID field which is used when manipulating save files and making sure that the relevant save file is being manipulated depending on said saveID field. Figure 6.2 below shows the methods in the SaveSystem class and with comments on how they work.

Figure 6.2 - SaveSystem Class Methods:





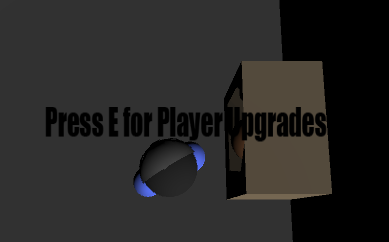


Separating both the SaveSystem and SaveData functionality had to be done as the SaveData would be saved to the save files as the class is serializable, so to keep cohesion and for error avoidance a separate class would create the SaveData objects and have methods for manipulating the save files themselves.

## 5.8 Add Player Upgrades

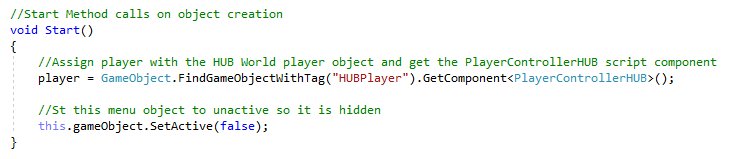
The Player Upgrades have an object like the HUB World object with the Machine Trigger script attached that opens the menu, this object is shown in Figure 6.3 below.

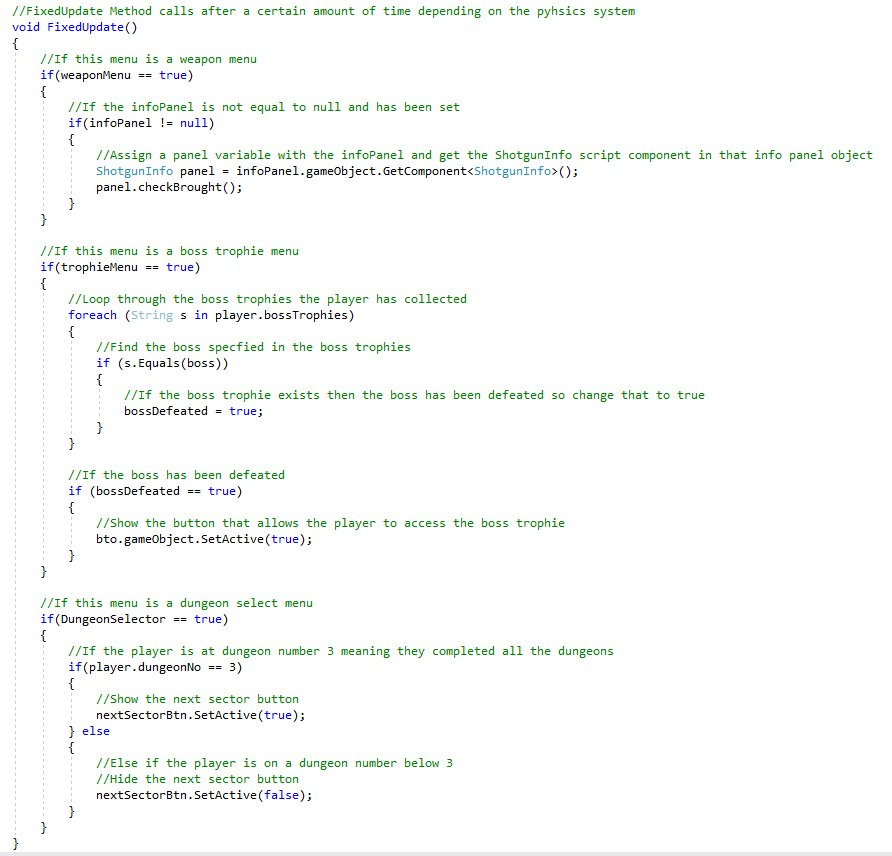
Figure 6.3 – Player Upgrades Interactable Object:



All the menus, including the HUB World Upgrade menu, use the Menu script which has checks and such for the specific types of menus which are set through a Boolean field in the inspector and referenced in the code. Figure 6.4 below shows the Menu script.

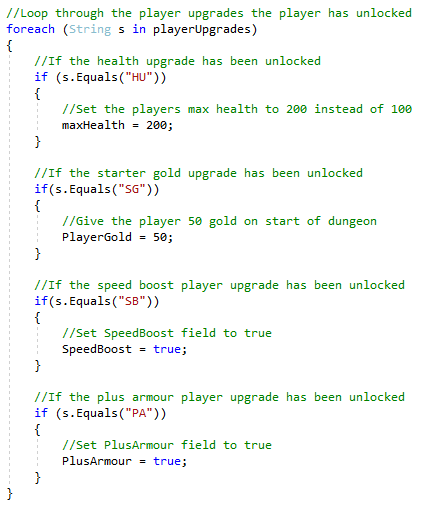
Figure 6.4 – Menu Script:





When the player buys a player upgrade it is then saved like every other upgrade and specifically for this and the weapons they are loaded properly into the dungeon player so the player can make use of these upgrades brought for their dungeon runs. The dungeon player and the gun spawner will check for the weapons and player upgrades brought respectively and if they are all brought and equipped, player upgrades equip automatically, then the abilities and such that they offer can now be used in the dungeon. The dungeon player check for the Player Upgrades is shown in Figure 6.5 below.

Figure 6.5 – Player Upgrade Checks in the Dungeon Player



Saving the upgrades meant that when the player buys these upgrades they will not have to buy them again as they save when brought or any other time the game saves. Creating a menu script for each menu in the HUB World meant that this script could be reused for every type of menu in the HUB World, especially new ones that could be created in the future, and makes the menus easy to maintain and modify with the script being cohesive as it only deals with the menus in the HUB World.

## 5.9 Add Different Equipable Weapons for the Player

This uses the same system as the Player Upgrade system; however this uses a different interactable object to show the menu and when a new weapon is equipped it changes which gun is then spawned on the player using the gun spawner object with each weapon having their own prefab so they can be spawned. Figure 6.6 below shows the interactable object for buying and equipping weapons.

Figure 6.6 – Weapon Selector Interactable Object:

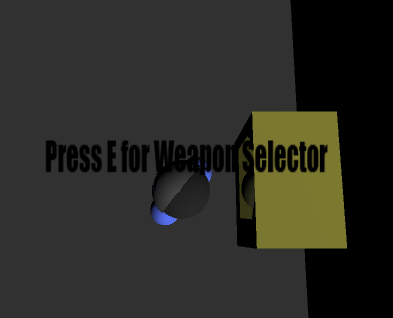
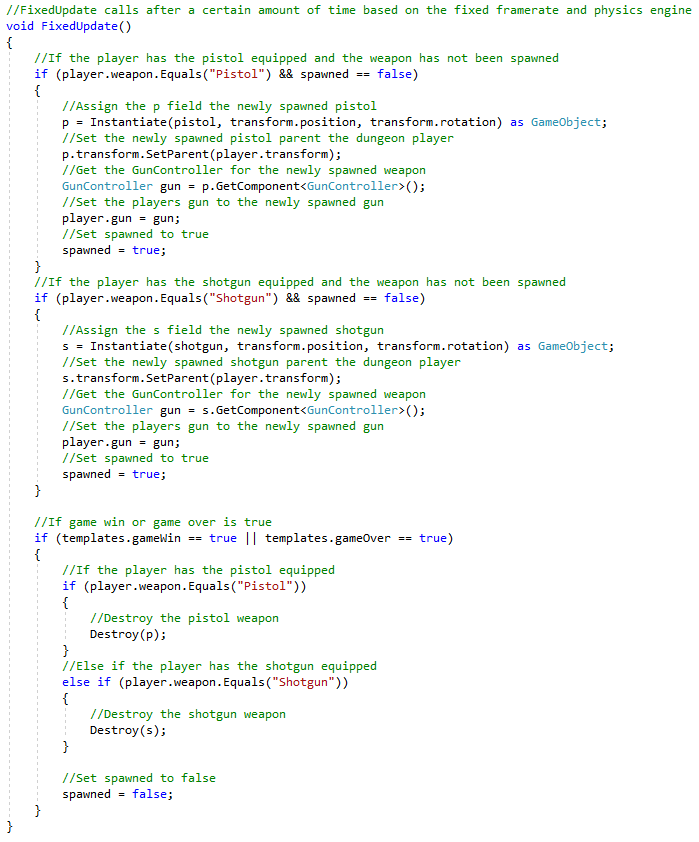


Figure 6.7 below shows the code in the GunSpawner class for the dungeon player object that spawns the relevant equipped gun for the player.

Figure 6.7 – Spawning the Equipped Weapon Code:



The Gun spawner code is in a separate script for cohesion sake to make sure that everything in the GunSpawner script is about spawning the equipped gun for the player to use. New weapons can be added easily as there would need to be a new button option for the menu and then a new check would be added to the GunSpawner code for this new option. Not much code would need to be added and no new scripts would need to be added which makes this script easy to modify and easily maintainable.

## 5.10 Add Boss Trophies

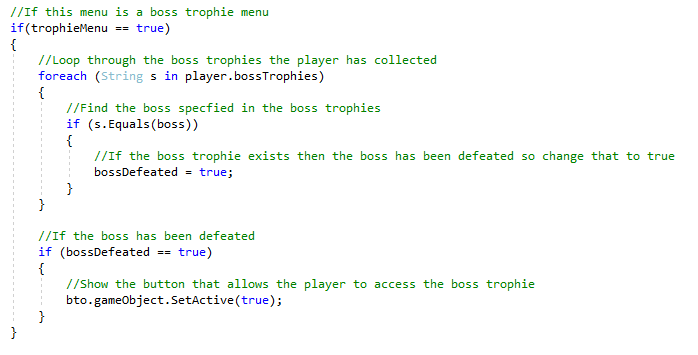
The boss trophies work with the same system as the upgrades and the weapon selector, with an interactable object with the MachineTrigger script attached to it that spawns the menu. The menu for Boss trophies will only show the boss icons if the boss has been defeated. This is because there is a script on the button for the relevant boss which keeps the button unactive even when the room spawns unless the boss has been defeated, which is value within a field that is saved and carried over between both the HUB World and the Dungeon scene so when the player does defeated the boss that field changes to the initials of the boss and then when the player gets back to the HUB World that data would be loaded in so interacting with that menu again would show the boss icon which would set the relevant info panel to active when it’s hovered over. Figure 6.8 below shows the interactable object for the boss trophy mechanic in the game world.

Figure 6.8 – Boss Trophy Interactable Object:



Within the FixedUpdate Unity method of the Menu script, there is code as shown below in Figure 6.9 that sets the relevant boss buttons to active when the relevant boss is defeated.

Figure 6.9 – Setting the Boss Trophies to Active Code:



The boss trophies are added when the boss has been defeated, with the Menu script checking the boss trophies if the menu is a boss trophy menu. This shows the reusability of the menu script and how this menu script is not just used for the weapon and upgrade menus, meaning adding new menus and features to the HUB World will be easier with scripts that can be reused and modified of needed.

## 5.11 Add a Tutorial Menu

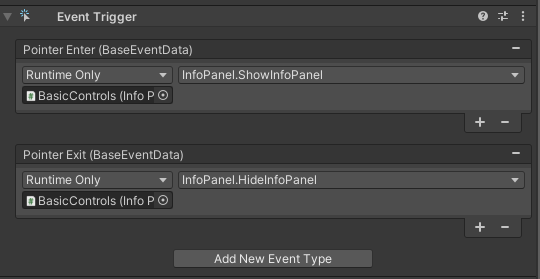
The door in the HUB World is an interactable object with the MachineTrigger script, which spawns the tutorial menu that has many different buttons which show an info panel when hovered over. This info panel would have information about the specific topic on the button, like the basic game controls and such. Shown below in Figure 7.0 is the interactable object for the tutorial menu.

Figure 7.0 – Interactable Object for the Tutorial Menu



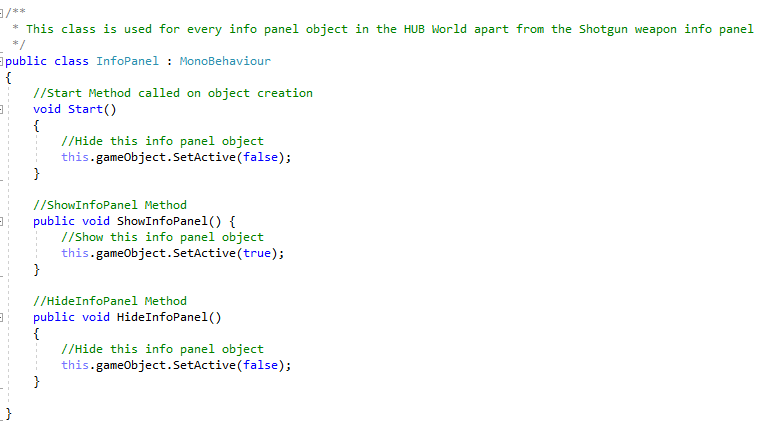
The buttons would use methods that were set within the info panel to show or hide the info panel when the button is being hovered over or not. Each button will show and hide the info panel that is relevant to their tutorial topic and not any other info panels. For example, Figure 7.1 shows the button setup for the Basic Controls button.

Figure 7.1 – Basic Controls Tutorial Button Inspector Setup:



The pointers in pointer enter and pointer exit refers to the mouse pointer. All the info panels share the same info panel script shown in Figure 7.2 below.

Figure 7.2 – InfoPanel Script:



The info panel was also used for the upgrade and weapon menus that show info panels with details on the upgrade or the weapon, the tutorial menu reuses this script for the info panels used on this menu. This means that any new menus made or any new panels that are added to any of the menus as new options could get added then this script can be reused making it easier to add more. As this script only focuses on the showing and hiding the info panel this script is cohesive, it doesn’t have any code for the buttons or any other part of the menu. It would not make much sense for the info panel to also have code for the buttons and such of the menu, so this makes the script easy to use and add to new info panels when needed in the future.

## 5.12 Dungeon and Sector Level Selector Menus

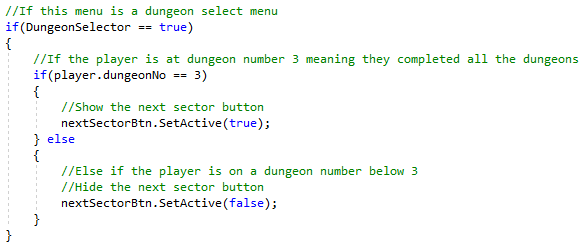
This uses the same MachineTrigger system for all the other interactable objects, however the menus work differently. The dungeon selector menu shows buttons for each dungeon in the current section, the hiver sprite showing when hovered or the button would be in that hover state permanently to show the dungeon has been completed. This system carries over to the sector select menu which only opens when the next sector button is clicked on the dungeon selector menu which only shows up when all the dungeons of the current sector have been completed. Figure 7.3 below shows the interactable object in the game world that allows the players to interact with these menus in the game world.

Figure 7.3 – Dungeon Selector Interactable Object:



In Figure 7.4 below the Dungeon Selector code in the Menu script is shown, this code checks for whether the next sector button should be set to active or not when the dungeon selector menu spawns and if it is set to active then the player can choose the next sector.

Figure 7.4 – Next Sector Button Check in the Menu Script:



The sector and dungeons work as int fields which increment whenever the player goes up a dungeon level or a sector level. To go up a dungeon level the player must complete the current dungeon and to go up a sector level all the dungeons must be completed in the current sector and the next sector must have been chosen.

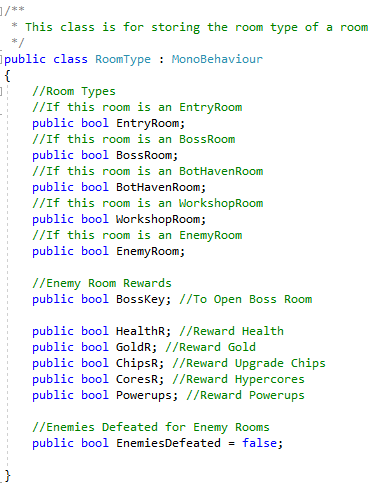
Having the next sector button check in the Menu script keeps the cohesion of the Menu script while adding a check for this specific menu type that spawns the next sector button when needed, and changing this code to work for more dungeons would not take much time and the hardcoded number would just need to be a variable that could be manipulated.

## 5.13 Implementing the Different Types of Dungeon Rooms

First the different room types had to be assigned, and then the objects for said rooms would spawn. Each of these rooms also has a specific room icon which is shown in the minimap. The bot haven room and workshop room has an object that spawns in the middle of the room that the player can interact with, and the boss room has an object that is intractable too however it spawns only when the player has defeated the boss.

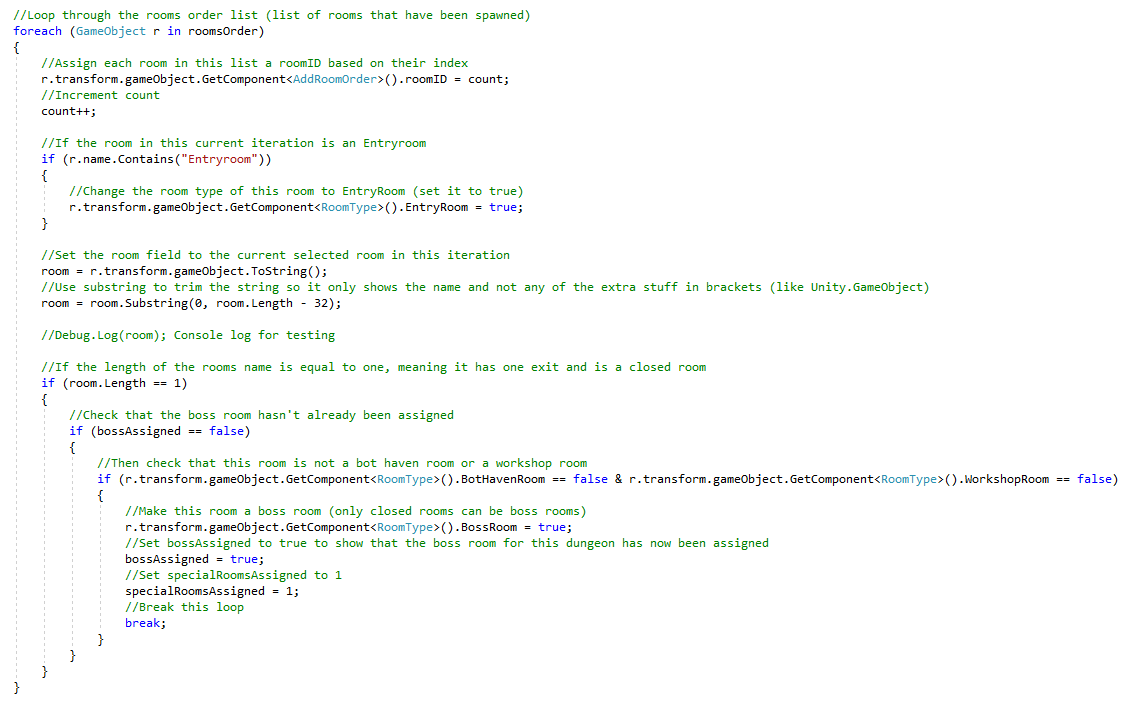
Figure 7.5 below shows the Room Type script that is attached to every single room and it stores which room type this spawned room is and what reward will drop if it is an enemy room which spawns an encounter.

Figure 7.5 – Room Type Script:



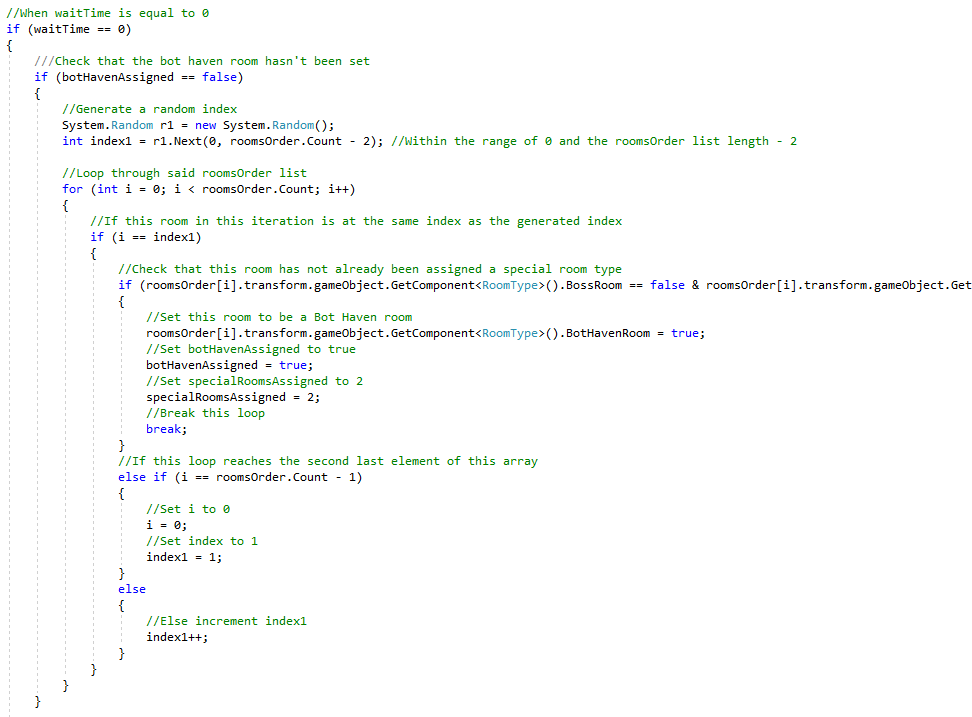
Within the FixedUpdate of the RoomTemplates script, the different room types are assigned. This FixedUpdate loops through a list in the RoomTemplates script which contains which rooms have been spawned to make up the dungeon. When the entry room is spawned the room has Entryroom as part of the rooms name, this is because the entry room has only three possibilities of room prefabs to spawn which have this naming convention. Within this FixedUpdate there is a check to check the name of the room and if it follows this naming convention it will assign the room the entry room type. The boss room is assigned to a closed room, meaning that there is only one exit to this room. These checks are shown below in Figure 7.6.

Figure 7.6 – Assigning the Entry and Boss Room:



The bot haven and workshop room type is assigned by generating a random index and then looping through the rooms spawned list in RoomTemplates, then checks the room chosen through this process has not been assigned another room type before assigning the room the relevant room type, which is either bot haven or workshop. After all these rooms have been spawned then the script will assign the rest of the room who don’t have a room type the enemy room type. Figure 7.7 below shows the bot haven assign script which is very similar to the workshop assign script.

Figure 7.7 – Assigning the Bot Haven Room:



The room type script assigned to every room is a very easy to understand and maintainable script which would just need an extra field for any new room types or rewards that would be assigned. The assigning of these room types is in the RoomTemplates script as it is attached to a script that is always active meaning that nothing would interrupt the assigning of these rooms by de activating the object this script is attached to. However the room assignment code itself could have been copied over to a separate script and could have been attached to the RoomTemplates object, this way there would be more cohesion with these scripts which was learnt later on in the development process.

## 5.14 Summary

This section went over how each project objective was implemented and why it was implemented the way it was. The next section goes through the testing phase of this project, and more specifically the testing done to each of the sub objectives covered in this section and the project plan section.

# 6.0 Project Testing

Testing was done after every mechanic was implemented and a final test was done after every phase as this project used test driven development. Below is the table that was used when testing was used.

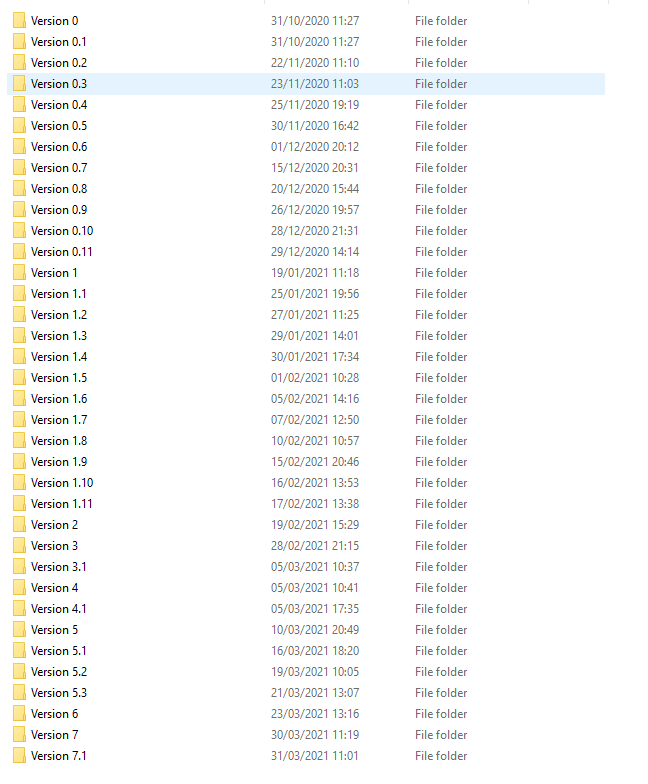
|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| The description of the test | Green for pass  Red for fail | Fixes needed if test failed |

The tests committed on every project objective with notes are shown in Appendix D – Project Tests.

# 7.0 Version Control

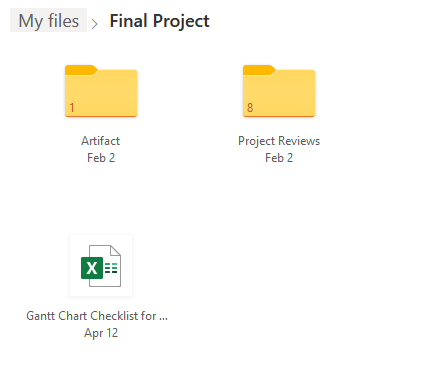
GitHub was used for version control as shown in Appendix B – GitHub Repository for the Project Code; this was used to have backups in case of any file corruptions and such. However there was also local version control being used as well in case a version needed to be rolled back as the current version was completely bugged out because of a new feature added or just for quick and easy access to previous versions. The local version control is shown in Figure 7.8 below.

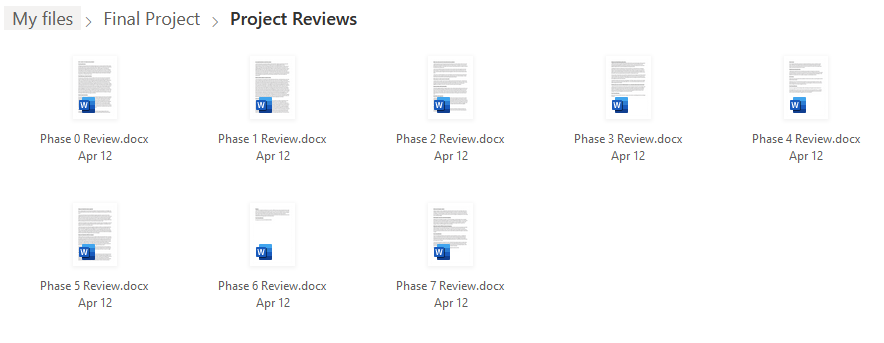
Figure 7.8 – Local Version Control:

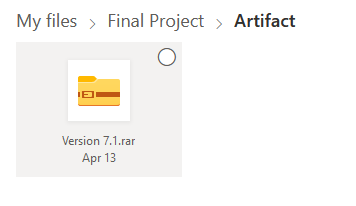


A backup was also created on OneDrive; this was in the form of a folder called Final Project. This folder contained: the Gantt Chart that was being used as a checklist with the completed sections highlighted in green, a Project Reviews folder which contained review documents for each Phase of the project, and an Artifact folder which contained the zipped up Unity project file of the most recent version of the project. This is shown below in Figure 7.9 below.

Figure 7.9 – OneDrive Backup







The next section of this report is the Project Evaluation section to evaluate the project made and the learning achieved through the development of this project.

# 8.0 Project Evaluation

This project will be evaluated through the following criteria: personal evaluation, research evaluation, and artefact evaluation.

8.1 Personal Evaluation

This evaluation section goes over the learning achieved during the development of this project, as listed below:

* Learnt how to develop and a design a game using professional techniques and standards from the industry.
* Learnt how to critically evaluate existing games and then that knowledge was used to design this project based on the project objectives.
* Learnt more about the C# programming language and the Unity engine, and applied this knowledge to complex problem solving during development.
* Gained an understanding on player motivation and how to design the game and game mechanics so that they keep the player motivated to play.
* Gained understanding on evaluating games and learnt how to effectively evaluate existing games.
* Learnt how to effectively and efficiently manage time for bigger and more complex projects

The lessons learnt during this project are important as I want to pursue a career in a similar field to this project and this will help when making big projects with the lessons on effective time management and the more technical lessons learnt will help with projects that are a lot more similar to this project.

8.2 Research Evaluation

The research undertaken has been important for this project to gain an understanding on pre-existing games and how they dealt or did not deal with the main three project objectives. The game design and game mechanic research was important for designing good mechanics that can achieve the projects main objectives. The player motivation research was important to take into account during the design phase so the mechanics designed not only achieve the projects main objectives but achieve these objectives effectively. The algorithms research was important for figuring out how to randomly generate the dungeons and the AI research was important for designing the enemies and boss so good difficulty balancing can be achieved as one of the projects main objectives. The methodology being important for figuring out what methodology to use as a structure for this projects development process.

8.3 Methodology Evaluation

The test driven development methodology was used which was a good choice as it allowed for lots of testing which was needed with the amount of different mechanics and such that would be added. Having lots of tests also meant that bugs and errors where a lot easier to catch, especially as there where some bugs that were not caught from earlier phases but were found in later phases thanks to the amount of testing done.

8.4 Artefact Evaluation

This project has implemented the sub objectives as they were mechanics and features that were to be implemented to achieve the three main objectives, an evaluation on each main objective is shown below.

*8.4.1 Main Objective: Difficulty Balancing*

The sub objectives for Difficulty Balancing have all been implemented and work well, with these mechanics, enemies, and such also being able to be easily added to so the player can have more variety with harder enemies and stronger upgrades.

*8.4.2 Main Objective: Save and Load Feature*

This main objective was achieved as the save and load mechanic was implemented with three different save slots with a save and quit option in the HUB World, autosave is also a feature that happens when entering or leaving a dungeon run and when upgrades and such are brought.

*8.4.3 Main Objective: Clear End Goals*

The sub objectives for Clear End Goals have all been implemented with short and long term goals available for the player to complete with room to add more variety to these goals, especially the upgrades, weapons, and boss trophies.

*8.4.4 Conclusion*

Overall all the main objectives have been achieved with the solutions, in the form of sub objectives, work together mechanically to make sure these main objectives are achieved.

For future development more variety can be added for specific mechanics such as:

* Stronger player upgrades for later on in the game
* More HUB World upgrades for more customisation
* More enemy types that get harder as the sector and dungeon levels increase
* More styles for the dungeons in each sector level
* Add more powerups for the player to use
* Add more bosses for each new dungeon
* Add a final boss as a special type of dungeon for the last sector
* Manipulate the spawning of the different room types in the dungeons as the sector and dungeon levels increase

All these points have the potential to add onto the solutions I have already provided by just adding more to the already existing mechanics and features.

In conclusion this project achieved all of its objectives and proved the hypothesis correct, that these three main problems: Missing Save Features, No Clear End Goals, and Poor Difficulty Balancing, can be solved with the solutions used in this project.

# 9.0 Bibliography

Aspnes, J. (2020, October 29). *Notes on Randomised Algorithms.* Retrieved from yake.edu: http://www.cs.yale.edu/homes/aspnes/classes/469/notes.pdf

Blackthornprod. (2018, March 13). *RANDOM DUNGEON GENERATOR - EASY UNITY TUTORIAL - #1*. Retrieved from YouTube: https://www.youtube.com/watch?v=qAf9axsyijY

Bowes, J. (2014, July 17). *Agile vs Waterfall: Comparing project management methods*. Retrieved from manifesto: https://manifesto.co.uk/agile-vs-waterfall-comparing-project-management-methodologies/

Brackeys. (2018, December 2). *SAVE & LOAD SYSTEM in Unity*. Retrieved from YouTube: https://www.youtube.com/watch?v=XOjd\_qU2Ido&t=679s&ab\_channel=Brackeys

CellarDoorGames, & Abstraction. (2013). Rogue Legacy. Cellar Door Games.

cprime. (Undated). *What is Agile? What is Scrum?* Retrieved from cprime: https://www.cprime.com/resources/what-is-agile-what-is-scrum/

DodgeRoll. (2016). Enter the Gungeon. Devolver Digital.

Fabricatore, C. (2007). *Gameplay and Game Mechanics: A Key to Quality in Videogames.* University of Huddersfield.

GeeksforGeeks. (2020). *Shuffle a given array using Fisher–Yates shuffle Algorithm*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/shuffle-a-given-array-using-fisher-yates-shuffle-algorithm/

Hunicke, R., LeBlanc, M., & Robert, Z. (2004). *MDA: A Formal Approach to Game Design and Game Research.* Northwestern University, Electrical Engineeing and Computer Science Department.

Kulkarni, K. (Undated). *What is Test Driven Development (TDD)? Tutorial with Example*. Retrieved from Guru99: https://www.guru99.com/test-driven-development.html

McMillen, E., & Himsl, F. (2011). The Binding of Isaac. Headup Games.

NewMediaRights. (2020). *Video Games and the law: Copyright, Trademark and Intellectual Property*. Retrieved from New Media Rights: https://www.newmediarights.org/guide/legal/Video\_Games\_law\_Copyright\_Trademark\_Intellectual\_Property

Przybylski, A. K., Ryan, R. M., & Rigby, C. S. (2010). *A Motivational Model of Video Game Engagement.* American Psychological Association.

Rabin, S. (2013). *Game AI Pro: Collected Wisdom of Game AI Professionals. 1st Edition.* CRC Press.

Rabin, S. (2015). *Game AI Pro 2: Collected Wisdom of Game AI Professionals. 2nd Edition.* CRC Press.

Rogers, S. (2010). *Level Up! The Guide to Great Video Game Design. 1st Edition.* John Wiley & Sons.

Schell, J. (2019). *The Art of Game Design: A Book of Lenses. 3rd Edition.* CRC Press.

SubsetGames. (2012). FTL: Faster Than Light. Subset Games.

SupergiantGames. (2018). Hades. Supergiant Games.

Suter, B., Kocher, M., & Bauer, R. (2018). *Games and Rules: Game Mechanics for the Magic Circle.* Transcript Verlag.

Unity. (2020). *Unity User Manual 2020.3 (LTS)*. Retrieved from Unity Documentation: https://docs.unity3d.com/Manual/index.html

VeewoGames. (2020). Neon Abyss. Team17; Yooreka Studio (China).

# 10.0 Appendices:

## 10.1 Appendix A – The Game Design Document

Introduction:

This is a game I am making for my final project, to make a roguelike game that solves the major problems with most roguelike games out on the market.

Marketing:

Genre(s): Roguelike

Similar Games: Neon Abyss, HADES, , FTL: Faster than Light, the Binding of Isaac, and Enter the Gungeon

Setting: Space with planets and space stations, space stations as the hub worlds for each sector and planets as the dungeons

Look: 3D

Technology: Unity and Blender

Gameplay:Interactions: The player will have upgrades they can get from the hub world, each hub world containing the same upgrades. The player will be able to switch between different weapons that have different stats and playstyles. The HUB World will allow the player to buy weapons, player upgrades, and cosmetic HUB World upgrades. There will also be an object for boss trophies that are collected when bosses are defeated.

Simulation: There will be different enemy types that have different weapons and abilities. There will a boss at the end of every dungeon. The dungeons will be random, and so will the enemies you encounter.

Environments: There will be planets, stations, or ships that act as dungeons and space stations that act as HUB Worlds.

Navigation: Each sector has multiple dungeons and a HUB World, to complete the sector you must complete the dungeons within said sector. You can save your progress after completing a dungeon (it will auto save that as well). Each run of a dungeon will be random. Always start each sector in the space station.

Quests: Complete each sector, collect all the boss trophies, complete the dungeons in each sector (optional quest to buy all the upgrades and weapons, or just buy the ones needed).

Goals: Same as quests.

Consequences: Powerups and other reward pickups in the dungeon will be taken away once the player has exited the dungeon (apart from specific items that help with the purchasable upgrades in the HUB World, which are Hyper Cores and Upgrade Chips).

Currency:HUB World upgrades can be brought using hyper cores, and player upgrades and different weapons can be brought using upgrade chips. Gold is used to buy things from the workshops in the dungeon.

Game Flow:Menus: Main menu, choosing dungeon and sector menus, pause menu, workshop and bot haven menu, save slot menu (new game, load game, delete save), weapons menu, boss trophies menu, HUB World and player upgrade menus, and tutorial menu.

Hub world: space station.

Dungeons: planets, stations, or ships.

Maps: Will show what type of room it was (if it has a type) and will appear as the player discovers the dungeon.

Story:

You are part of the resistance and must defeat the empire.

Types of Rooms:

Workshop – for buying items, special abilities, upgrade chips, and hyper cores.

Enemy Rooms – the most common room of enemies where the player will defeat enemies and get a reward at the end, also known as encounters.

Bot Haven – a room that has a NPC robot, the player can talk to this NPC to give you one of these things:

* Health
* Hyper Cores
* Gold
* Upgrade Chips

Boss rooms – every dungeon has one, this is where the boss of the dungeon is. The player must defeat this boss to complete the dungeon.

Beat Chart:

Design notes taken during the implementation of the project:

The boss that will be implemented is called Solar Guardian. This boss will have four different phases with each activated when the bosses health bar reaches a certain value. The phases are detailed below:

* Phase 1 – Is the first phase of the boss where the boss follows the player and shoots enemy bullets at them.
* Phase 2 – Activated at 350 out of 400 health, after a certain amount of time the boss will move to the centre of the boss room and rotate while firing loads of bullets from every direction before going back to following the player.
* Phase 3 – Activated at 200 out of 400 health, after a certain amount of time clones will spawn around the boss that must be defeated before the player can deal more damage to the boss.
* Phase 4 – Activated at 150 out of 400 health, buffs movement speed and damage while reducing all the bosses’ cooldowns.

The two types of enemies will be Head Jars that are fast and deal melee damage to the player when in range, and Drones which are slower enemies that fire bullets at the player.

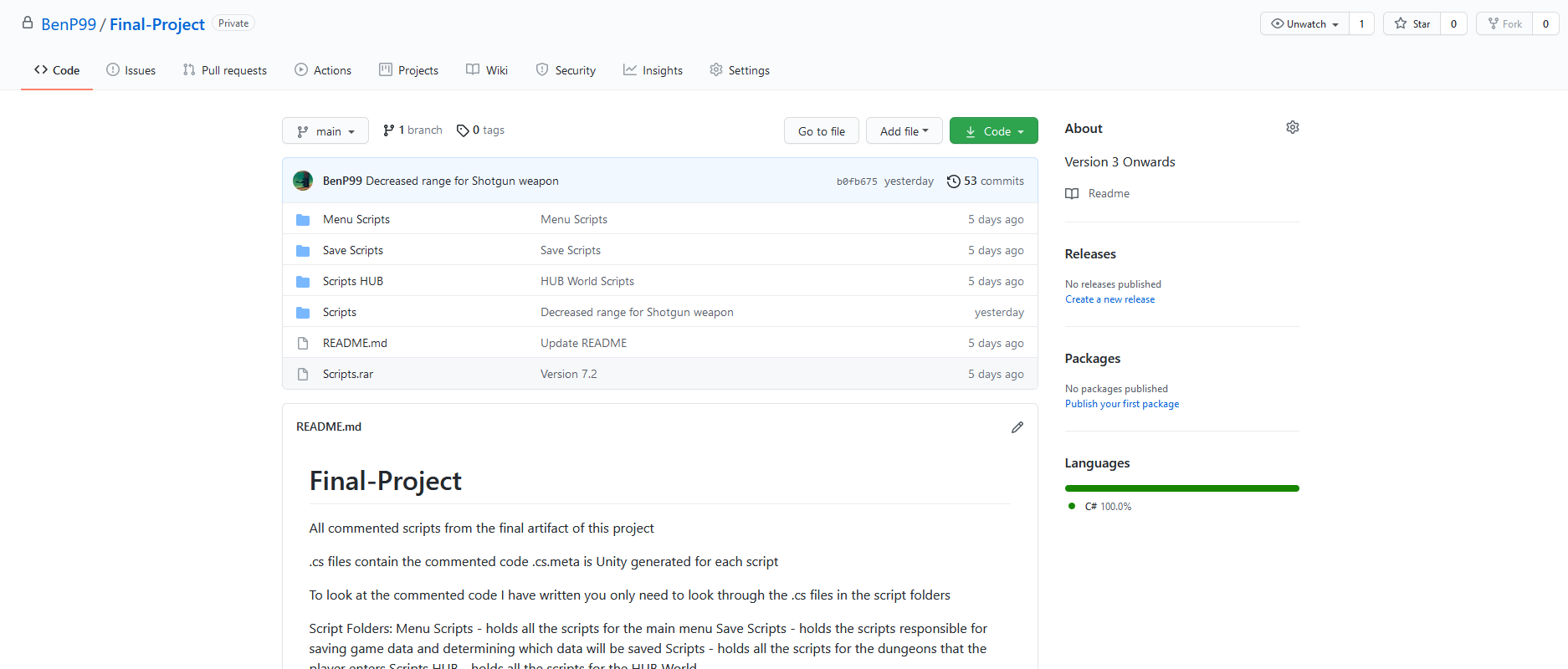
Special Room Types:

* The stock for the workshop will be restocked whenever the players buys everything that was in the previous stock.
* Only one workshop and one bot haven room will spawn per dungeon.

5 Gold will be rewarded after every encounter when all the enemies have been defeated.

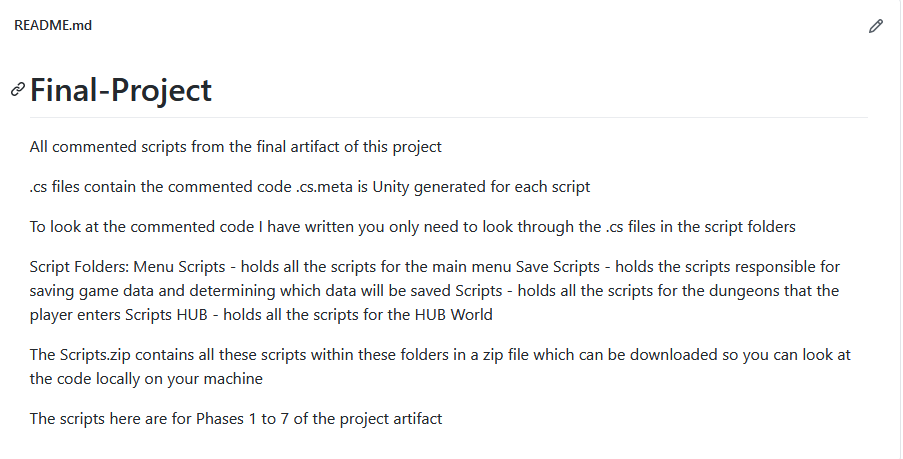
## 10.2 Appendix B – GitHub Repository for the Project Code

The link and screenshot below shows the GitHub repository which includes all the commented code made for this project.



Link: <https://github.com/BenP99/Final-Project>

The README attached to this repository is shown below:



The commits that were pushed onto this GitHub are shown below:

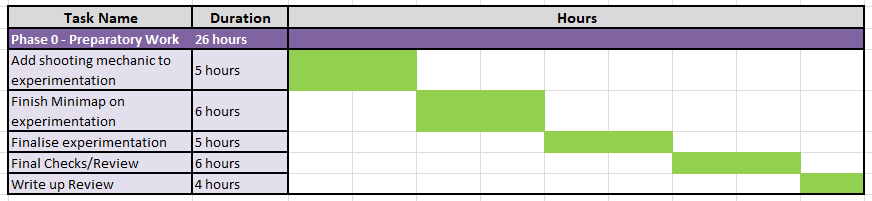


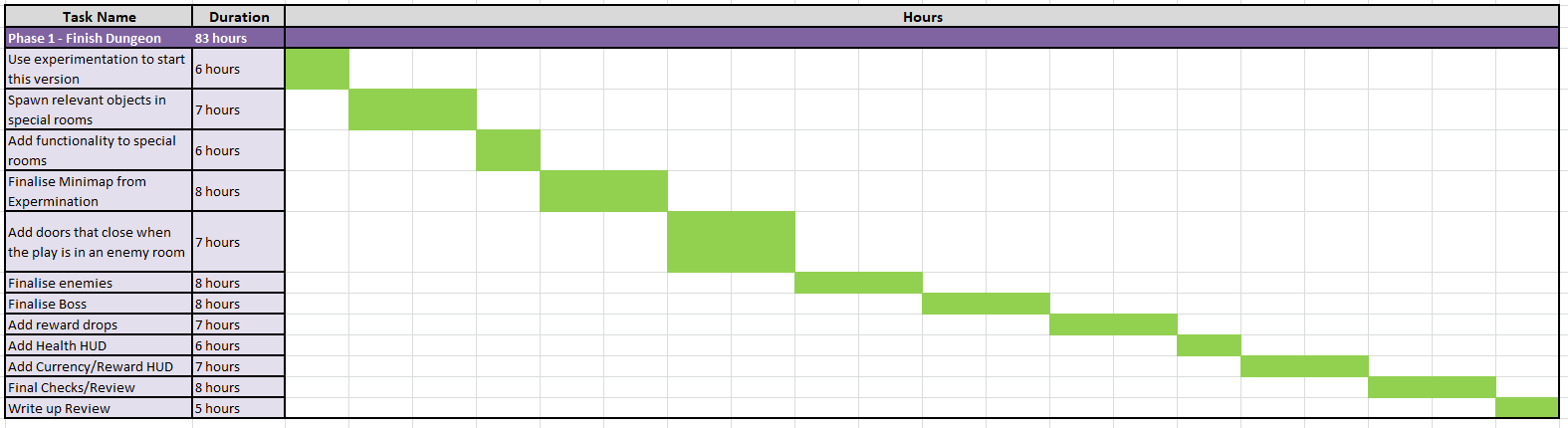


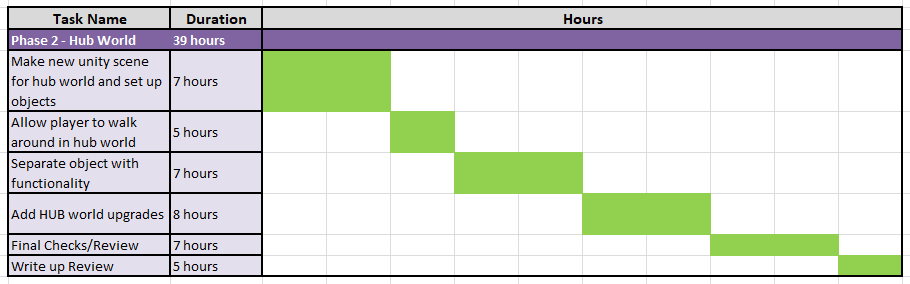
The reason April 13th has loads of commits is because this was when the scripts where added as their own files as well as having them zipped into a Scripts zip folder which was how the script where committed during development.

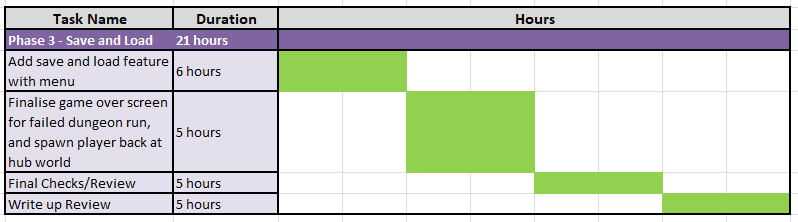
## 10.3 Appendix C – Gantt Chart

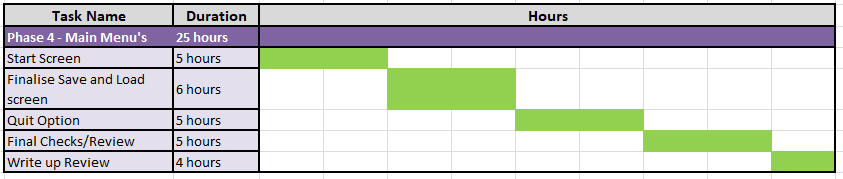
Shown below is the Gantt Chart that was created and used for this project, split into different phases to effectively and clearly show each planned phase within this Gantt Chart.

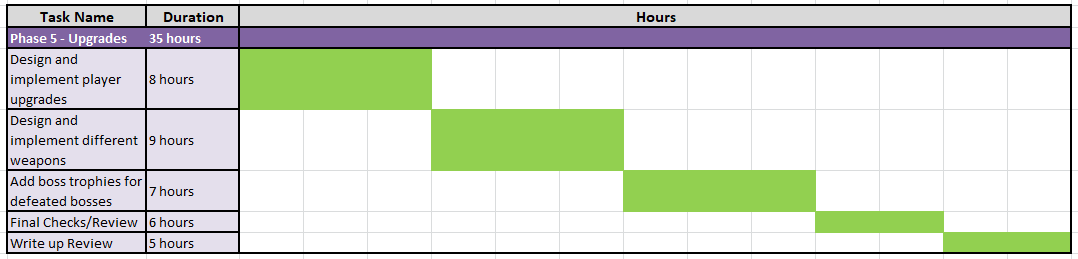


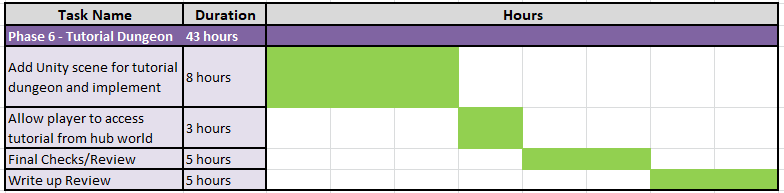


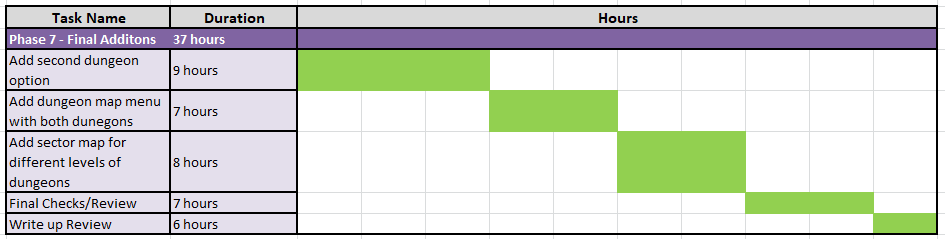












## 10.4 Appendix D – Project Tests

The tests done for each project objective is shown below:

Random Dungeon Generation Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Dungeon Rooms spawn |  | N/A |
| Dungeon is closed |  | N/A |
| No room openings lead to a wall |  | Some openings still lead to a wall even though they shouldn’t |

To fix the room openings, the CheckedBlockOpenings script was modified so the script checked the room names, which held the different openings the room had, and compared it with the rooms it collided with to check that the counterpart to each opening existed in the other room. For example: a room with a top opening needs the room it collided with to have a bottom opening so the top opening of the room does not lead to a wall.

Enemies and Boss Tests

The tests below were the tests that were done on: the drone, head jar, and boss enemies.

Drone Enemy Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Drone Movement works |  | N/A |
| Heath bar decreases when damage is taken |  | N/A |
| Enemy shoots at player |  | N/A |

Head Jar Enemy Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Head Jar Movement works |  | N/A |
| Heath bar decreases when damage is taken |  | N/A |
| Player takes damage when Head Jar is within melee range |  | N/A |

Boss Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Boss Movement works |  | N/A |
| Heath bar decreases when damage is taken |  | N/A |
| Phase 1 works as intended |  | N/A |
| Phase 2 works as intended |  | N/A |
| Phase 3 works as intended |  | N/A |
| Phase 4 works as intended |  | N/A |

Reward Drop Tests

The tests below were the tests that were done on: health pickups, gold pickups, upgrade chip pickups, hyper core pickups, the boss key, and powerups.

Health Pickup Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Player can interact with this pickup |  | N/A |
| Give the player 12 more health, if the player is not already at full health |  | N/A |

Gold Pickup Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Player can interact with this pickup |  | N/A |
| Give the player 10 gold |  | N/A |

Upgrade Chip Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Player can interact with this pickup |  | N/A |
| Gives the player 8 upgrade chips |  | N/A |

Hyper Core Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Player can interact with this pickup |  | N/A |
| Gives the player 8 hyper cores |  | N/A |

Boss Key Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Player can interact with this pickup |  | N/A |
| Player can now use the boss key to open the boss room door |  | N/A |

Powerup Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Player can interact with this pickup |  | N/A |
| PowerupStock generates a valid powerup stock generation for each of the powerup pickups |  | Did not properly choose two powerups and would sometimes only choose one |
| Number of equipped powerups shown on HUD |  | N/A |
| Test EMP Powerup effect |  | EMP Cooldown bar wouldn’t disappear when the player no longer had the EMP Powerup |
| Test Attack Drone Powerup effect |  | N/A |
| Test Overdrive Powerup effect |  | N/A |
| Test Shields Powerup effect |  | N/A |
| Test Extra Plating Powerup effect |  | N/A |

To fix the PowerupStock generation the loop within the Choose Powerup method so that when the game is going through the loop it will stay in that loop, and restart the loop if needs be, to find a valid powerup starting from the random index.

To fix the EMP cooldown bar an extra check was added to make sure that when the player no longer has the EMP powerup the EMP cooldown bar will be set to not active if it is active.

Minimap Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Minimap rooms spawn |  | N/A |
| Player Icon successfully shows which room the player is in |  | N/A |
| Room Icons show on the relevant special room types |  | N/A |
| Current room changed successfully in player so the minimap rooms can spawn |  | N/A |

HUB World Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| HUB World scene added, and player can walk around in said scene |  | N/A |
| Player can transition from the HUB World to the Dungeon |  | N/A |
| Player can transition from the Dungeon to the HUB World |  | N/A |

Save and Load Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Save Game works |  | N/A |
| Load game works |  | N/A |
| Delete Save works |  | N/A |
| Check all data successfully loads |  | N/A |

Player Upgrade Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Menu opens successfully |  | N/A |
| Menu closes successfully |  | N/A |
| Upgrades are set active when brought |  | N/A |
| Player upgrades brought are active in the dungeon scene |  | N/A |

Weapon Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Menu opens successfully |  | N/A |
| Menu closes successfully |  | N/A |
| Upgrades are set active when brought |  | N/A |
| Weapons spawn properly in the dungeon scene |  | N/A |

Boss Trophy Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Menu opens successfully |  | N/A |
| Menu closes successfully |  | N/A |
| Boss trophy appears in menu once the boss has been defeated |  | N/A |

Tutorial Menu Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Menu opens successfully |  | N/A |
| Menu closes successfully |  | N/A |
| Relevant tutorial descriptions show up in the info panel when the buttons are hovered over |  | N/A |

Dungeon and Sector Select Menu Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Menus opens successfully |  | N/A |
| Menus closes successfully |  | N/A |
| Choosing the active dungeon button will take the player to the relevant dungeon |  | N/A |
| Choosing the active sector button will take the player to the relevant sector |  | N/A |
| Button is no longer able to be clicked and sprite changed when dungeon is completed |  | N/A |
| Button is no longer able to be clicked and sprite changed when sector is completed |  | N/A |
| Next sector button shows when all dungeons in the current sector are completed |  | N/A |

Different Dungeon Room Types Tests

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Task Succeeded/Failed** | **Fixes Needed** |
| Room types are being assigned |  | N/A |
| Rewards are being assign to the enemy/encounter rooms |  | N/A |
| Room objects spawn successfully depending on the room type assigned |  | N/A |