Project report – Journey to the sun roguelike

Candidate Number:Centre Number:Qualification Code:Date: 17/12/23

Rohan Sim

Contents

[1 Introduction 3](#_Toc158218657)

[2 Stating the Problem 3](#_Toc158218658)

[2.1 The Problem 3](#_Toc158218659)

[2.2 Computational Methods 4](#_Toc158218660)

[2.3 Stakeholders 4](#_Toc158218661)

[3 The Approach 5](#_Toc158218662)

[3.1 Game Engine 5](#_Toc158218663)

[3.2 IDE 5](#_Toc158218664)

[4 Assets and Tilemaps 6](#_Toc158218665)

[5 Requirements Specification 7](#_Toc158218666)

[6 Development 7](#_Toc158218667)

[6.1 Iteration 1 – Player Movement 7](#_Toc158218668)

[6.1.1 Design 7](#_Toc158218669)

[6.1.2 Implementation 8](#_Toc158218670)

[6.1.3 Testing 9](#_Toc158218671)

[6.1.4 Evaluation 9](#_Toc158218672)

[6.2 Iteration 2 – Player Attack 9](#_Toc158218673)

[6.2.1 Design 9](#_Toc158218674)

[6.2.2 Implementation1 10](#_Toc158218675)

[6.2.3 Testing 10](#_Toc158218676)

[6.2.4 Evaluation 10](#_Toc158218677)

[6.3 Iteration 3 – Map Generation 10](#_Toc158218678)

[6.3.1 Design 10](#_Toc158218679)

[6.3.2 Implementation 12](#_Toc158218680)

[6.3.3 Testing 25](#_Toc158218681)

[6.3.4 Evaluation 28](#_Toc158218682)

[6.4 Iteration 4 – Enemies 28](#_Toc158218683)

[6.4.1 Iteration 4.1 – Enemy Spawning 28](#_Toc158218684)

[6.4.2 Iteration 4.2 – Enemy Movement 35](#_Toc158218685)

[6.4.3 Enemy Properties 38](#_Toc158218686)

[6.4.4 Implementation 40](#_Toc158218687)

[6.4.5 Testing 40](#_Toc158218688)

[6.4.6 Evaluation 40](#_Toc158218689)

[6.5 Iteration 5 – Enemy Attack 40](#_Toc158218690)

[6.5.1 Design 40](#_Toc158218691)

[6.5.2 Implementation 40](#_Toc158218692)

[6.5.3 Testing 40](#_Toc158218693)

[6.5.4 Evaluation 40](#_Toc158218694)

[6.6 Iteration 6 – User Interface 40](#_Toc158218695)

[6.6.1 Design 40](#_Toc158218696)

[6.6.2 Implementation 40](#_Toc158218697)

[6.6.3 Testing 40](#_Toc158218698)

[6.6.4 Evaluation 40](#_Toc158218699)

[6.7 Iteration 7 – Player Upgrades 40](#_Toc158218700)

[6.7.1 Design 40](#_Toc158218701)

[6.7.2 Implementation 40](#_Toc158218702)

[6.7.3 Testing 40](#_Toc158218703)

[6.7.4 Evaluation 40](#_Toc158218704)

[7 Bibliography 40](#_Toc158218705)

# Introduction

This report will serve as the primary destination for everything related to this project. Its purpose will be to provide information for the progress of the project throughout its lifecycle. This report will also discuss the requirements for the project and analyse the different features and computational techniques that will be required before even beginning development. This will take up a large proportion of the document, as it is important that the requirements have been fully analysed and investigated, so that they are clear, unambiguous, and able to be easily implemented into the solution when development begins, with minimal issues and delays due to unforeseen events that could have been mitigated had more time been spent planning.

Once development has begun, I will be using this report to document the process from start to finish. This may come in the form of screenshots taken from the solution itself, with descriptions on the purpose of the code shown, and information on any issues that may have been encountered when writing it and how they were resolved. This will be beneficial as if similar issues are encountered in the future, I will be able to reference this document and come to a fix faster as I will know the process required to fix it. It will also be useful to show the stakeholders how the project is developing to keep them informed and have something to reference when discussing future improvements or testing.

# Stating the Problem

This section will overview the problem that I am trying to solve with this project. It will discuss the things relating to the problem such as the gathering of requirements in order to get a full view of what the solution will entail and how it will be accomplished, along with identifying the key stakeholders of the project who will have the best idea as to what they want to gain from the solution.

## The Problem

This game will be a 2D top-down roguelike twin-stick shooter game, where the player will complete levels on each of the planets in the solar system, starting on Neptune, and ending the game in the final level which will take place on the Sun. Each level will start the player with a basic gun, firing projectiles at nearby enemies. Items that power up the player can be found throughout the level in randomised locations that may give the player a better weapon, or improve their stats. At the end of each level, the player will collect an armour piece that will be able to absorb the energy of the sun, that will prevent it from becoming a supernova. All armour pieces will have to be collected from each planet in order to achieve this and beat the game. To my knowledge, this game concept has not been created before, and for that reason I believe it will be an interesting and unique project that has a solid selling point.

## Computational Methods

The core of this game will be the different planets that the player will explore, and the maps that will be used to represent them. In order to create these levels procedurally, while still keeping correct theming, object-oriented programming will be used. This will allow us to create classes for important things such as enemies, weapon archetypes, and the maps themselves that will be generated based on a class containing the generic features and functions for each level. This will result in code that is much easier to read, and be more expandable to make the creation of new levels much simpler and easier to implement into the game while keeping them unique and allowing the developers to add new content to the levels to make them different and unique to each other.

Other methods, such as abstraction, may be used during the development of the game, and can be used in cases such as the spawning of enemies. It may be beneficial, for initial testing, to create code that allows for entities to spawn outside of a given radius, at random intervals, and in random spaces. This will get the core functionality of the enemy spawning system in place, which can then be later expanded upon to include the sprites and AI that will be used to make the enemies move, follow, and attack the player. The method of abstraction can be used throughout the project to ensure that all systems implemented work on a purely functional level, such as having a sprite move around a plane with the WASD keys, and colliding with a wall to prevent further movement in that direction. This will be able to be expanded upon by implementing the player sprite, along with the required perspectives for the player character, and creating walls around the player to define the play-space. By abstracting these functions, I can be sure that the systems work as intended before fully implementing them into the game.

Encapsulation will also be an integral part of the game to protect the different variables that will be present throughout. Some of these variables might be the player’s health, as we do not want it to be able to go below 0, or the enemies’ speed, which will also have the same issue as if their speed goes below 0, then they will be moving backwards.

## Stakeholders

The stakeholders of this project are the people who I must create the game in mind for. I will base all requirements off of their feedback, and must ensure that throughout development, that they are being catered for in the content that is added to the game. For example, if one of the stakeholders believes that the projectiles fired by the enemies are too fast, I must conduct testing on the speed, and possibly allow the play-testers to run through some levels with projectile speeds adjusted to different speeds, from which they can then help us to finetune the gunplay of the game to be as fair as possible while still keeping the player engaged, and with a certain level of challenge. The stakeholders for this game will be my classmates, as they are a part of the key demographic for this game, as well as my future colleague, who will provide aid and feedback on the technical aspects of the project, as their primary requirements for the project are that is runs optimally on all hardware, so that it remains accessible to all players, regardless of their PC specifications.

# The Approach

The following headings will discuss how the project will be approached, and the software-based requirements that will be used, such as the IDE (integrated development environment) and the programming languages that will help us to create the game. I will also decide on the development methodology that I will use for the project, between ones such as Waterfall, Agile, and Spiral.

## Game Engine

A couple of game engines were considered for use in this project. Those were Unity and MonoGame. MonoGame was considered due to it being easy to set up, simply installing it onto Visual Studio and creating a solution with it like any other project. It would allow us to create a simple 2D game, which would be beneficial for this project, however one of the drawbacks that made us unlikely to use it was the fact that there was not a preview window of the game available outside of runtime, where objects and assets could be moved graphically, rather than by code. This was available in Unity, making the creation of levels and layouts much more manageable and easier to create in our short timeframe.

Another benefit of Unity was that I would be able to easily import assets from the Unity store directly into our project, which would not be possible with MonoGame, with us either having to create them ourselves, or find them online and then import them ourselves. The Unity Asset Store will help greatly with intuitive imports, and also allow us to purchase higher quality assets than those I would likely come across searching elsewhere.

Unity also allows us to modify the properties created in the code directly from the engine, and modify numeric values such as speed and attack damage without having to go into the code itself for the object I am modifying.

## IDE

For this project I will use Visual Studio for the IDE. This is because it is well integrated with both Windows machines, which this game will be created in mind with, as well as Unity, the game engine. It has a wide range of libraries that can be imported and used that will make various aspects of development significantly easier, as I will be able to use pre-made functions specific to what I want to achieve instead of having to create them from scratch. This will greatly cut down on development time, and there is also a good chance that the functions from the libraries will be better optimised and less prone to issues and bugs than ones I may make myself for the same purpose.

Visual Studio also allows for great C# functionality, which is the language that unity uses, meaning that development will be much easier due to the tools available for this language that may not be available in other editors or IDEs.

These tools include things such as IntelliSense, break points, and unit testing capabilities that will make general development and code-writing much easier. IntelliSense can be used in all aspects of development to autocomplete lines of code. As you begin writing code, IntelliSense will attempt to guess the code you are trying to write, to which you can then press the tab key to automatically enter it into the code. This minimises the risk of misspelling words or missing out other syntax, as the computer writes the code itself, and can be beneficial in suggesting likely functions that will be used that you may not have known about to achieve the goal you were working towards.

# Requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement | User Requirement | Justification | Evidence | Success Criteria |
| 1 | Player movement | User needs to be able to move around the map | [Iteration 1 – Player Movement](#_Iteration_1_–) | Player can move in all 8 directions |
| 2 | Player Attack | Player must be able to kill enemies to continue to the next room | [Iteration 2 – Player Attack](#_Iteration_2_–) | Player can shoot projectiles in 4 directions |
| 3 | Map Generation | Procedural generation means higher replayability with map changing each time you play | [Iteration 3 – Map Generation](#_Iteration_3_–) | Rooms generate procedurally with multiple branches leading off each room for the player to explore |
| 4 | Enemies | Increases player enjoyment, primary focus of game is combat | [Iteration 5 – Enemy Attack](#_Iteration_5_–) | Enemies spawn in, move around, and attack player |
| 5 | Enemy movement | If enemies stay still, game becomes easy and boring | [Iteration 4.2 – Enemy Movement](#_Iteration_4.2_–) | Enemies move around their room, targeting the player if they get too close |
| 6 | Enemy Spawning | If enemies do not spawn, the player would simply walk around rooms and have no goal in the game | [Iteration 4.1 – Enemy Spawning](#_Iteration_4.1_–) | A set number of enemies spawn from a set list of presets |
| 7 | Enemy Attacking | Enemies otherwise walk around the room and do not pose direct threat to player | [Iteration 5 – Enemy Attack](#_Iteration_5_–) | Enemies attack based on their preset |
| 8 | Health | Player needs to have health that can decrease when they get hit to add reason to avoid enemy attacks | [Iteration 6 – Player and Enemy Properties, Player Upgrades](#_Iteration_6_–) | Player has 6HP that decreases by 1 each time they get hit |
| 9 |  |  |  |  |
| 10 |  |  |  |  |
| 11 |  |  |  |  |
| 12 |  |  |  |  |
| 13 |  |  |  |  |
| 14 |  |  |  |  |
| 15 |  |  |  |  |
| 16 |  |  |  |  |
| 17 |  |  |  |  |
| 18 |  |  |  |  |
| 19 |  |  |  |  |
| 20 |  |  |  |  |
| 21 |  |  |  |  |

# Assets and Tilemaps

The first part of project development is importing all of the assets that will be used in the project. Given the low budget and timeframe for the game, obtaining assets relating specifically to Neptune (first level of the game) was not possible, and instead I had to utilise assets for a regular dungeon, while still attempting to keep close to the theming that I was aiming for.

Unity did not have free assets that were usable for this project, so I had to look externally, and found these asset packs on Itch.io, a website for publishing tools for game creation, as well as complete games for donations.

A screenshot of a video game

Description automatically generated

Figure - Tilemap for room layout

The tilemap shown above is what will be used for the general layout of the rooms, and will primarily use the wall, floor and door tiles to create them. This has been chosen as it works well for the kind of layout that the rooms will have, as a lot of other tested assets did not work very well, or required a lot of finetuning to get it to look right in the game. This pack also works will with the general theming of the game, with the enemies and player sprites that I am going to use.

A video game screen with a dark background

Description automatically generated with medium confidence

Figure - Tilemap for obstacles

The tilemap below will be used for general things such as obstacles that the player and enemies will have to navigate around, to make the gameplay more interesting and add some more strategy to the game in blocking and avoiding enemy attacks.



Figure - Enemy and player sprites

I have chosen to use this asset pack for the enemies and player sprites in this game. This is primarily due to the lower budget of the game, making it difficult to find sprites that properly fit the sci-fi theming initially set out early in the report. In a future version I may choose to update the theming of the game with different sprites that more accurately reflect the theme of the game, travelling through different planets, however for the initial version, getting the core gameplay loop and mechanics working as intended is more important.

# Development

## Iteration 1 – Player Movement

### Design

Player movement is a core component of any top-down game, and ensuring that it works well can make or break the experience the user has when playing the game. For this reason, it is important that I get the factors such as responsivity and speed just right so that the player can easily manoeuvre the game area avoiding enemy attacks, while not feeling overly slippery from unresponsive controls.

The player movement will be 8 directional by using WASD inputs and allowing for diagonal inputs to be made by using 2 adjacent keys at a time. Due to the nature of the code that will be written to implement this, it is likely that some extra code will have to be added to prevent the player from moving faster diagonally as the code will be adding forces for both the horizontal and vertical inputs, making the diagonal speed faster due to Pythagoras theorem.

As I am still learning Unity at this stage, and have very limited knowledge on how C# game development flows, I have chosen to use Brackey’s movement guide to give me some basic ideas on how components, methods and libraries work within Unity, to help with this part of development. (Brackeys movement guide, <https://youtu.be/whzomFgjT50?feature=shared>). However, as I want to ensure that my game feels unique to play, I will make some changes to the script to modify it slightly, changing the values and the way the player moves slightly.

The player movement will work by getting the raw axis values for the horizontal and vertical axis’, which are either positive or negative 1 in each direction. These will then be multiplied by the movement speed, with a force added to the players sprite using a 2D RigidBody component placed on the player, moving the player forward with a constant force.

**[INSERT FLOWCHART FOR MOVEMENT]**

Before even thinking about movement however, before even creating the player’s movement,

### Implementation

The first

A screen shot of a computer program

Description automatically generated

Figure - Original Movement Script

A screen shot of a computer program

Description automatically generated

Figure - New Movement Script

The player movement script has been renamed to “PlayerBehaviour” as it has expanded past player movement to also include the attacking and projectile instantiation.

### Testing

### Evaluation

## Iteration 2 – Player Attack

### Design

Player attacking will work in 4 directions, regardless of what direction the player is facing or moving in. The attack will work by pressing any of the arrow keys, with projectiles being fired in the respective direction in set intervals. These shots will be created from a prefab object, consisting of the projectile asset, a glow effect, and a collision hitbox.

This prefab will then be instantiated as a clone with the required forces based on the direction it is being fired in, and will be destroyed when it meets with another collider.

Once enemies are added, further functionality will be added to either the projectiles themselves, or the enemy class, detecting whenever a projectile collides with the enemy’s hitbox, triggering an event to reduce the enemy’s HP.

A screen shot of a computer program

Description automatically generated

### Implementation

### Testing

### Evaluation

## Iteration 3 – Map Generation

### Design

The design for this section of the game was partially collaborated on with my classmate. Together, we discussed the general flow of the algorithm, of which some design choices made it into the final design, and others were modified slightly or removed in favour of alternative approaches.

The room generation system will draw a lot of inspiration from the game “The Binding of Isaac”. That game consists of multiple rooms linking off of one another, each with 1-4 doors branching off into other rooms. I figured that the best way to create this system was to look at how rooms were generated in that game and attempt to replicate the process in my own design.



Above is an example of a map generated in the game. Each room has between 1 and 4 rooms connected to it. However, as you can see in this game, some of the rooms are larger than the others, taking up either 2 or 4 rooms total. I thought that for this initial version of my game, it would be best to stick to generating only a single size of room, as the developers of Binding of Isaac stated that it was a very difficult task to implement properly, and is something that would be out of my timeframe for this project.

#### Coordinate System

The first step in the design of this system is to create a coordinates system for each of the room placements. As each room has the same size, it means that it can fill into a box, with each room taking up a coordinate value of that box. For example, the first room the player spawns into will have coordinates (0,0). The room to the right will have coordinates (1,0), the room above that will have coordinates (1,1) and so on. This will make the building of the map much easier as rooms can simply be instantiated into those coordinates without having to directly worry about where they will be placed in the engine’s coordinates, with the camera following them as they do so.

#### Stages of Map Generation

To make map generation simpler, it will be carried out in two major steps. The first step will be the creation of the layout, without creating the physical rooms that the player will explore. This will be in the form of multiple coordinate vectors that will each represent the location of a room. Once this has been completed, and the maximum number of rooms for that level has been reached, then the next step can begin.

The next step will be the implementation of the physical rooms into the scene. Multiple clones of the room prefab will be instantiated onto the coordinates that were made previously, with checks being carried out as they are implemented, checking for any rooms that land directly next to them. These checks will determine whether or not a door should be placed on any one of the sides of the room, allowing the player to navigate to the next room once all of the enemies have been defeated.

Once these rooms have all been created and placed into the scene, then a script will run that removes all of the doors from the scene that are not required to move to another room. This will make it so that the player can not escape the map and go out of bounds.

#### Creation of Map Layout

The first step in creation the layout will be to create the starting room, which will always be at (0,0) using the rooms coordinate system. Alongside the instantiation of this room, a number will be picked from a list of numbers, ranging from 1 to 4 (these numbers will be weighted so that rooms with only 1 child and rooms with 4 children are less common. While useful for if the newly created room has no children, if it does, then it may cause issues with overlapping rooms as the algorithm will attempt to add rooms onto coordinates where there are already existing rooms. An example of this is a room that is to be generated where it will already have 4 children. Once it has been instantiated, the algorithm will decide how many child rooms it should have, and if it chooses 4, then 4 rooms will be created on top of the already existing ones. To remedy this, I will run a check to see how many rooms already exist next to the coordinate the new room will be generated in. If the number of pre-existing rooms is greater than or equal to the generated number, then no new rooms will be spawned from the new room. If it is less, then the algorithm will have to place the new rooms in areas where no rooms already exist.

The process of creating the coordinates where rooms should be placed will end once the maximum number of rooms has been reached. This will be determined by a variable that will increase as the player progresses through the game and the different planets, as each planet will consist of one iteration of the map generation, with each consecutive level becoming more difficult to encourage the player to improve and stay engaged in the game.

Each level in the game will have a single boss room that, once defeated, will allow the player to progress to the next level. These boss rooms will be spawned once the maximum room number has been reached, and will be assigned the room furthest away from the origin coordinates, encouraging the player to explore more of the level before it is completed.

At the end of this process, there will be a set of clones created in the project hierarchy with their associated coordinates. These objects will be placed into the scene in their respective coordinate, to allow for the room to be placed on top of them in the next stage of generation.

#### Removing doors after generation

Once all of the rooms have been creating for the level, I will have to remove any doors that do not lead to other rooms so that the player can not walk out of the map. This will be done by running a check on all of the rooms, much like during the initial room generation, to see if they do or do not have a room connected to them. If they do, then the door will remain, if not, then a wall will be placed on top of the door, likely by enabling the renderer and collision for and already placed wall included in the prefab of the object. While not disabling the door, it will prevent the player from moving through as an extra layer of collision will be present that will not be disabled when the doors are removed after defeating all of the enemies.

#### Enabling and disabling doors

Whenever the player enters a room, all of the doors will be closed, to force the player to defeat all of the enemies in the room. Once this is done, all doors will open. Taking into consideration the fact that I have 4 different door presets, corresponding to the left, right, top, and bottom doors, this means that I have to find a way to affect all of these doors simultaneously. The best way to accomplish this would be to add a tag called “doors” and have the code search for all objects with the same tag, and disable both the renderer and hitbox of the object, so that the player can walk through.

### Implementation

#### Opening/Closing Doors

As enemies have not yet been created, there is no way to control the opening and closing of the doors in the game. For testing purposes, it would be beneficial to allow the player to open and close doors using a keybind to allow them to navigate the map easily to view the different rooms and the layout that has been created.

A computer screen shot of a program code

Description automatically generated

As shown in the image above, this algorithm works by creating an array of all the GameObjects that have the tag “Door” and enabling and disabling the renderers and colliders of every object in the array.

One issue with this code, however, was that the doors would open and close every single frame for as long as the “k” key was held down. This was due to the condition for executing the procedure was met for as long as the key was held down, as it was only checking for an input.

A screen shot of a computer program

Description automatically generated

This error was fixed by changing the “GetKey” method, to “GetKeyDown”, which simply checks whenever the input is initially made, and returns true on the first instance of the key being pressed, and false for all the frames after until it is released and pressed again. This made it so that the doors did not constantly open and closed, and only did it when I pressed the key.

#### Coordinate System

The next thing I wanted to implement was the coordinate system that would be used to determine where rooms would be placed in the scene. To do this I had to create various rooms attached to the first test room to find their respective x and y values.

A screenshot of a computer

Description automatically generated

As shown above, the first room has an overall position vector of (0, 0, 0), meaning it is centred at the origin of the scene.

A screenshot of a computer

Description automatically generated

Room 2 shows that the centre of each room along the x axis will be 22 units greater than the previous. Room 3 also shows this but for the y axis, which each room being 16 units great than the one below it.

This meant that I had to make a way for any vector entered in regard to room location, was multiplied by 22 and 16 in the x and y axis respectively whenever it was ready to be interpreted by the engine.

A computer screen shot of a program

Description automatically generated

The image above shows the RoomController class. This class will be responsible for the creation of the rooms and the overall map of the level. In the initial test, to ensure the coordinate system works properly, I simply created 3 rooms using the system to test that they were all placed in the right location. Executing this code, however, had some interesting results.

A computer screen shot of a program

Description automatically generated

As shown above, I decided to modify the code to make it more easily readable how rooms are created. Each room is created by calling the “CreateRoom” method, followed by the currentRoomCoord and the direction the new room should be created in relative to the current room.

A screenshot of a video game

Description automatically generated

First of all, in the scene view, the 3 rooms were created and placed on the map in the correct locations relative to each other, but not at the origin where the player is. This is peculiar, because both the starting room and the player supposedly have the same coordinates according to the Inspector tab.

This was fixed by going into the parent object for the room, and changing the position of that to the origin of 0, as it was not previously set to that, resulting in all of the instantiated rooms being offset by that amount.

A pixelated image of a person

Description automatically generated

The second interesting event was that in the game view, all of the rooms are invisible to the player, despite still being rendered in the scene. In the end, this was due to the camera being positioned at the z value -1000, which was what I was using in the initial creation of the room at the start of development. After changing the z coordinate to 0, the rooms were visible on the game view.

A screen shot of a computer program

Description automatically generated

A switch case statement was used here to multiply the index of the first array by a chosen index in the second array. While this worked mostly as intended, there was an issue where the number 1 did not appear anywhere in the array, with the other numbers being selected an incorrect number of times, as shown below.



This was fixed by reducing the number on the cases in the switch case statement, as it was using the case for the next index value, which was not intended. Corrected code can be seen below, along with the result of the algorithm when it is run.

A computer screen with white and purple text

Description automatically generated



#### Creating Camera Movement

In order to navigate the rooms created when the game begins, I have to make sure that the camera is able to follow the player across the map. This will be done by changing the camera’s position to the centre of the new room that the play enters.

The first step in this process was to get updates on the room coordinate relating to the players position in the world.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated  
While this worked mostly as intended, updating the coordinates of the player, the coordinates ended up updating and increasing by 1 for every half room the player explored. If I changed the values to 22 and 8 respectively, then it would result in the coordinate updating whenever the player entered the centre of the new room.

A screenshot of a video game

Description automatically generated

As you can see above, the player is in room (1,0), while the Player Room Coord variable is still showing as (0,0) on the bottom right of the screen.

A screenshot of a video game

Description automatically generated

To fix this, I had to think of the world coordinate to room coordinate as an equation. This ended up with the form roomCoord.x = ((worldCoord.x + 11) / 22 for the x axis, and roomCoord.y = ((worldCoord.y + 8) / 16 for the y axis. This was due to the offset of the rooms being placed in the centre of the origin, so this had to be artificially corrected by adding half of the width/height of the room to the coordinate.

A screen shot of a computer program

Description automatically generated

The above code works by first setting the cameras world coordinate to the world coordinate of the room the player is currently in. Once that has been set, for testing purposes a debug.log line is ran to confirm the coordinates, before setting the position of the camera equal to its target coordinates. The “MoveTowards” method is used here to move the camera gradually to its destination, over the time defined by the third parameter of the method call.

##### Creating generation procedure

Due to the complexity of the generation algorithm, any planning ahead of time would only work for logic, and the basic flow of the algorithm, as shown in the planning section with the method flowchart. This meant that implementation of the algorithm had to be primarily creating a section of code, testing how the game reacted to it at runtime, and making modifications accordingly to work towards a working solution. The image below shows the first attempt at implementing the logic created for the algorithm.

This algorithm worked as intended for the first iteration, but would not create any rooms past that. In the end, this due to the start method and the while loop contained within it.

A screen shot of a computer program

Description automatically generated

An issue arised where rooms were all made at (0,0). This was due to the CreateRoom method taking the room coord of a value that was already in room coord form. This meant that the method for getting room coordinates from a coordinate in world coordinate form was ran twice, truncating all the values to 0. This was simply fixed by removing the “GetRoomCoord” method from the parameter, as shown below.



The next issue was the parent room variable not updating between iterations. This was fixed by modifying the Start() method.

Once I had fixed this, the procedure began to loop correctly, however once the game was run to be tested, Unity would crash.

The way I ended up finding the cause was by waiting for user inputs before executing the next iteration of generation, so that the engine had time to execute the instructions and finish what it was doing before continuing.

A computer screen shot of a program code

Description automatically generated

As shown above, this was done by moving the while statement to the Update method, and waiting for the user to press c before continuing.

A black background with white numbers and numbers

Description automatically generated

This showed that the room coordinates were getting exponentially bigger. On top of this, the coordinates in the name show the room coordinates, which are further multiplied by 22 and 16 to get the world coordinates, resulting in unimaginably high or small numbers, which could understandably cause a crash.



A black background with white text

Description automatically generated

This was fixed by only getting the room coord of the parent room position, as shown below.



The final issue I had was with the collision system. Often the collision system would not detect collisions and create a room in that position anyway. This was due to the DirectionList value being checked as a room coordinate, instead of applying the direction to the parent room and checking if that room exists. Once that had been fixed, the system would work as intended.

Once the based generation was working as intended, there were still a few small kinks to iron out with the algorithm. While not an error per se, as the room generation still generated 10 rooms in a random order, it was not working as intended relative to what I had initially hoped for with the child rooms.

To go into more detail, the issue was regarding other rooms generating rooms where other rooms had rejected due to not having the coordinates in their direction list. What this would result in, is that rooms could be made directly next to another room, ending in that room having a number of child rooms that is not equal to their intended amount. This meant that the generated map was much more central to the origin than I had hoped, with many rooms having 4 rooms connected to them, rather than being spread out, so the player would have to explore further.

To remedy this, I added a list for null rooms, for rooms which did not already exist, and were not a part of the direction list for the parent room. Whenever a new room would be created, it would check against this list to see if the coordinate was inside it. If it was, then the room would not be created.

### Testing

This table shows all of the major issues that were encountered during the iteration. Any tests that required retesting have their solutions shown in the Implementation section of the iteration, below a detailed description of the error.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No | Test Name | Expected Outcome | Experienced Outcome | Retest Required? (Y/N) |
| 1 | Door Check | Doors open/close whenever “K” is pressed | Doors | Y |
| 2 | Creating weighted list | Number in the first array containing numbers 1-4 are multiplies by the respective index in the probability array | Numbers are skewed by the wrong amount, number 1 is not shown anywhere in the list | Y |
| 3 | Setting number of childRooms | Whenever a room is created, a random number between 1 and 4 is set to the childRooms variable. | Intended outcome experienced. | Y (Evidence in 6.3.3.1) |
|  |  |  |  |  |
|  |  |  |  |  |

#### Retest No 1 - Doors

A video game screen with a purple square with a person in it

Description automatically generated

A screenshot of a video game

Description automatically generated

Doors mismatched, some open while some closed.

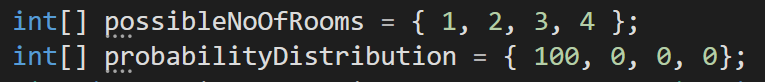
#### Setting number of Child Rooms

A small test was carried out here to ensure that the percentage chances for each potential number of children were working as intended.

This was done by setting up a key bind to reroll the number of child rooms, so that it could be tested multiple times.

A screen shot of a computer code

Description automatically generated



As shown above, the distribution of probabilities was changed to make it a 100% chance to generate 1 child room. This tested the integrity of the code by guaranteeing that it was being interpreted as intended and producing the correct result.

As shown below, the result of this was that the variable was only able to be set to 1, verifying the functionality of the code.

A screenshot of a black screen

Description automatically generated

### Evaluation

## Iteration 4 – Enemies

### Iteration 4.1 – Enemy Spawning

#### Design

The first step of creating enemy movement is the spawning of enemies. This will be done by creating an area where enemies will be able to spawn within each room, and placing a set number of enemies within that area.

I can get the maximum and minimum values for the x axis by adding or subtracting half of the horizontal length to the x coordinate of the object respectively. The same can be done for the y axis by using the vertical height and the y coordinate of the object. The enemies will be picked randomly from the list of prefabs and instantiated into the room at a random coordinate, between 1 and 6 times as to not overcrowd the room.

Each of these enemies will be instantiated during map generation, with the enemies for a particular room being generated right after the room has been created. This will make it easier to implement as I can simply write the spawning code inside the Room class and have it create in the same area as the room, instead of having to calculate it afterwards, which would require checking through all of the rooms again and placing in enemies one at a time. This way, the code will be both easier to implement, and more efficient and well-optimised.

A diagram of a room

Description automatically generated

Above is a flowchart of the initial design of the process for instantiating enemies into a room. As with all other code in this project, the implementation will likely be much more complicated than this, and begin to require additional classes and methods, as we are dealing a lot with prefabs, which makes it difficult for referencing instantiated game objects, as the prefab isn’t actually initiated and does not exist within the scene.

The different scripts we will need to make this work will likely be contained within the RoomController class, for conversions between Room and World coordinates, and the game object for the play area that the enemies will be spawned into in each room.

#### Implementation

Within the Room class, I began by creating references to the required game objects and scripts. This was done as shown below.

A screen shot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

It took a small amount of time to get this working, as the Room class is only instantiated at runtime, meaning that existing game objects cannot be linked in the inspector. The way it ended up working was by creating two variables, one for the game object that holds the script we want, and one for the script itself. We then set the game object to the game object in the scene, and the script to the script component within that object.

Once that was working, the next thing to implement was the random coordinate system.

A computer screen shot of a program

Description automatically generated

A screenshot of a computer

Description automatically generated

As shown above, this was implemented using some logic for the minimum and maximum values of x and y. This is the based on the fact that the playArea game object has a scale of (18, 12). The minimum value of x and y will be the full scale of the object, minus half of the scale, and the maximum value will be plus half of the scale.

#### Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No | Test Name | Expected Outcome | Experienced Outcome | Retest Required? (Y/N) |
| 1 | Get Components | Components are referenced within room | Can see components for room controller and enemy prefabs in the | N |
| 2 | Random Spawn Vector | Able to get a random spawn vector usable for instantiating enemies | Incorrect logic resulted in unusable vectors, however realised the need for a redesign. | Y |

#### Design – Test No 2

As per the failed test no 2, the random vector procedure required a design update that would allow it to be used with the world coordinate system, as the enemies will be instantiated based on the world coordinate of the room, plus the spawn vector. The vectors in the previous implementation did not return usable vectors.

The goal in this design is to create a procedure that can create positive and negative values from a fixed coordinate 0, that will be applied to the rooms centre coordinate to get a coordinate that resides within the room that an enemy can spawn in.

One way to implement this would be by having a constant vector 3 of (0, 0, 0), and applying either -9 or +9 to the x value for the minimum and maximum x values, and -6 or +6 for the minimum and maximum y values. This eliminates the need for unnecessary calculations when the calculation will be applying these minimum and maximum values to the rooms world coordinate to get a range of world coordinates that the enemy can spawn in for that room.

#### Implementation

A computer screen shot of a black screen

Description automatically generated

This is the updated version of the procedure. It uses 4 variables, primarily for ease of understanding, that have a random integer picked between them for the x and y values, those values are then plugged into a vector, and returned to be instantiated in the scene.

A screenshot of a video game

Description automatically generated

This is the result of the instantiation. Once the code has been applied to spawn them within their own rooms, they should not all spawn in the origin room.

A screen shot of a computer

Description automatically generated

The code we used for the instantiation is shown above. To get them to spawn in their own rooms, we added the transform.position of the room to the spawn vector of the enemy. This gave the following result in game:

A screenshot of a video game

Description automatically generated

As you can see, each room has 1-6 enemies, with the maximum in this case having been 4 on the far left room. One minor alteration to this that I wanted to make was to ensure that the enemies did not spawn partially inside the walls. To do this I simply reduced the minimum and maximum values by 1 so that they would not spawn on the far edges.

A computer screen shot of a program

Description automatically generated

A screenshot of a video game

Description automatically generated

The enemies now spawn fully inside the rooms, without partially clipping through the walls.

### Iteration 4.2 – Enemy Movement

#### Design

When brainstorming ideas for how to manage enemy movement in the game, my initial idea was to use a package from GitHub that utilised different scripts on both the surface and sprite to create meshes that would allow for AI pathfinding to be implemented. While this would have worked, I decided against it in the hopes that I would be able to design my own algorithms to accomplish similar functionality. While it will likely not be as foolproof as the package, which allows for pathfinding past obstacles and blockages, that will not be an issue in my game as it current does not have that feature implemented.

This will mean that in the future if I wanted to add obstacles, I would have to reevaluate the use of pathfinding tools, as through research I have found that creating your own will be too time-consuming and will almost always be worse than one that has already been created considering how much time it would take to create yourself.

How it will be accomplished in this version of the game, will be setting a variable for the enemy’s room to the value of the parentRoom when the enemy is instantiated. This will be called on the start method of the enemy when it is created, so should not update even after the value of parentRoom has changed. The GetRandomVector will then be called, and the enemy will move towards that vector in the room over a period of time. Once it has done that, it will wait a short amount of time before choosing a new vector and moving to that. The random wait time will ensure that enemies move at all different times to make their movement less predictable for the player.

#### Implementation

The first thing I did in this iteration was move the methods relating to enemies into their own utility helper class.

A computer screen shot of a program

Description automatically generated

This allowed me to reference them both inside and outside already instantiated objects. This was necessary due to the fact that they were initially held inside a prefab. As I wanted to access them inside another prefab as well, this was the simplest way to make that possible, as I could not create a reference to non-instantiated object inside another non-instantiated object.

A computer screen shot of a program code

Description automatically generated

A short script was created for the initial implementation of enemy movement. This script started by setting the world coordinate of the parent room at the time the enemy was instantiated to the general coordinate of the enemy. A target coordinate was then created to set where the enemy will attempt to move to. The enemy’s position will then be set to move towards that target coordinate over a certain period of time, modifiable by the enemies move speed.

#### Testing

When testing this implementation, I found that all enemies were selecting coordinates at the origin room, causing them all to move, regardless of their location on the map, to the room at 0, 0. Once there, they would move very quickly around that coordinate as if they were selecting new target coordinates relative to the origin.

A screenshot of a video game

Description automatically generated

Aside from the issue of using the wrong coordinates, and the fact that there seems to be a conflict between the usage of room coordinates and world coordinates, the algorithm seems to be mostly functional. The issues will simply have to be fixed in a redesign and implementation.

#### Design

A black screen with white text

Description automatically generated

After some more testing, it was discovered that the quick changes of direction before meeting their set positions was due to the update function being called multiple times, not allowing the enemy to get to their new position before having a new one be selected. This took a while to figure out, as it was not a problem that I had had with the camera, as the player was unable to move fast enough to make the camera move again before it had reached its intended position.

The best way to approach a fix for this method is to run a check to see if the enemy has reached their destination before selecting a new coordinate to move to, preventing new coordinates from being generated every frame.

Alongside this, I wanted to add a wait time between movements where the enemies would be idle, as I found that they would immediately move to their next position once they had reached their target. While this is not an issue code-wise, I thought this implementation would improve the player experience significantly, as it gives them a chance to react to the enemy’s previous movement before they move again.

#### Implementation

#### Testing

#### Evaluation

## Iteration 5 – Enemy Attack

This version of the game will have 4 enemy types, Skeleton1, Skeleton2, Skull, and Vampire, as shown below.

A screenshot of a video game

Description automatically generated

Each enemy will have its own respective colour projectile, so that the user knows exactly what enemy is firing what, and to help with memorising attack patterns, as each enemy will have their own style of attack. Skeleton1 will shoot a single projectile, similarly to the player’s projectiles, but will aim them in the direction of the player at a given point. Skeleton2 will shoot 3 projectiles, at slower intervals, with the centre being set towards the direction of the player, with the 2 either side being shot 45 degrees to the left and right of the centre projectile. The skull will shoot projectiles in al 8 directions, and the vampire will shoot much slower projectiles, less frequently, that will instead deal 1 full heart of damage instead of the usual half a heart that the other enemies deal.

Each enemy will have their own specific enemy class for attacking, that will instantiate their respective projectiles in the patterns that they fire in. Each projectile will inherit the EnemyProjectileBehaviour class as a general projectile class for enemies, that will cause damage to the player when they collide with one. Each class will also have a specific projectile class that will determine the things such as the speed of the projectile, and any overrides for damage dealt if the projectile deals more damage than a usual projectile, such as the vampire’s.

### Iteration 5.1 – Skeleton1 Attack

#### Design

As mentioned, Skeleton1 will shoot a single yellow projectile towards the player’s current vector at the time of firing.

#### Implementation

A computer screen shot of a program code

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

A screenshot of a video game

Description automatically generated

A computer screen shot of a program

Description automatically generated

Always shooting in same direction towards the top left of the screen.

#### Testing

#### Evaluation

### Iteration 5.2 Skull Attack

#### Design

Since creating the first skeleton’s projectiles, I have realised that it would be better for the player’s experience if the skulls only shot projectiles in 4 directions, instead of 8. This is because there will be too many projectiles on the screen, especially once the other enemies have had their projectiles coded as well.

Here is a basic flowchart for the process of creating the skull’s projectiles.

A white rectangular sign with black text

Description automatically generated

The simplest way to work this enemy’s projectiles is to create a prefab that already has the 4 projectiles in it, each with their own name for their direction. The behaviour class for the projectiles can then have variables within, referencing the children of the object, which will be each projectile. These projectiles will then be moved in their correct directions.

#### Implementation

First, a prefab of 4 skull projectiles was created.

A blue circle with white dots

Description automatically generated with medium confidence

Initially, these were created all on top of each other in the prefab, however this made some issues arise when instantiating then on top of the skull, as they would all push away from each other as a result of the colliders, meaning I had to move them away from each other like I did with the player’s projectiles, with each one being 1.5 units away from the enemy in each direction.

#### Testing



#### Evaluation

### Skeleton2 Attack

### Vampire Attack

## Iteration 6 – Player and Enemy Properties, Player Upgrades

For time management purposes, I have decided to create the Player and Enemy Properties before finishing enemy attacks, due to the attacks being completable at any point in the project separate from the other features. By doing this, it will allow me to better sense the time I have left to complete the project, and how much time I have to allocate to each section to ensure it is all completed in a good amount of time.

### Design

A screenshot of a computer game

Description automatically generated

The initial player properties are as shown above. This should be all that is needed for both general gameplay, and upgrading the player’s stats as they progress through the game. Enemies will also be given a health property, with that being the only modifier that will change throughout the game, as the player hits them with projectiles. In order for the player to tell if the enemy has been hit, then the enemy will turn red once a collision with the player’s projectile occurs. This will be done utilising the enemy’s spriterenderer.

### Implementation

### Testing

### Evaluation

### Player Upgrades

<https://medal.tv/games/binding-of-isaac/clips/1VAKoei0cZD1Ne/d13377piQn0S?invite=cr-MSxBRE8sMjU2NTUwMTks>

A screen shot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

Fixed by adding a check to see if the name of the room matched the name of the room the player was in.



A screen shot of a computer program

Description automatically generated

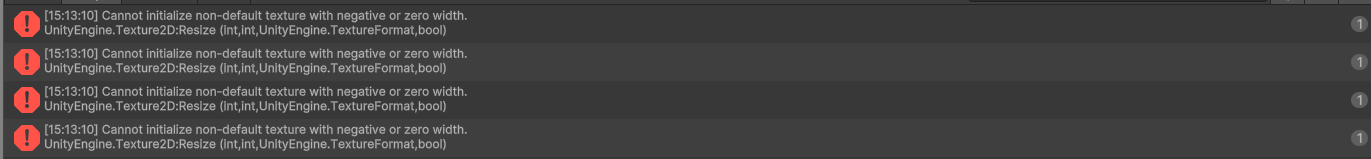
When increasing shot rate, the player’s shots would collide with each other and be destroyed due to the collision destroying the projectiles if they collided with anything other than the player. This was fixed by adding another condition, so that if they collided with another player projectile, they wouldn’t be destroyed.

A screen shot of a computer screen

Description automatically generated

## Iteration 7 – User Interface

### Design



This error was fixed by changing the version of Unity that the project was running on. It turned out that this was a known bug in version 2023.1. After I updated this to 2023.3, and reimported the text packages, this issue no longer occurred.

A computer screen shot of a program code

Description automatically generated

A screenshot of a video game

Description automatically generated

### Implementation

### Testing

### Evaluation

## Iteration 8 – Modifying Door behaviour

A computer screen shot of a program code

Description automatically generated



# Bibliography

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated