Project report – Journey to the sun roguelike

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

Rohan Sim

Contents

[1 Introduction 3](#_Toc156657566)

[2 Stating the Problem 3](#_Toc156657567)

[2.1 The Problem 3](#_Toc156657568)

[2.2 Computational Methods 4](#_Toc156657569)

[2.3 Stakeholders 4](#_Toc156657570)

[3 The Approach 5](#_Toc156657571)

[3.1 Game Engine 5](#_Toc156657572)

[3.2 IDE 5](#_Toc156657573)

[4 Assets and Tilemaps 6](#_Toc156657574)

[5 Requirements Specification 7](#_Toc156657575)

[6 Development 7](#_Toc156657576)

[6.1 Iteration 1 – Player Movement 7](#_Toc156657577)

[6.1.1 Design 7](#_Toc156657578)

[6.1.2 Implementation 8](#_Toc156657579)

[6.1.3 Testing 9](#_Toc156657580)

[6.1.4 Evaluation 9](#_Toc156657581)

[6.2 Iteration 2 – Player Attack 9](#_Toc156657582)

[6.2.1 Design 9](#_Toc156657583)

[6.2.2 Implementation1 10](#_Toc156657584)

[6.2.3 Testing 10](#_Toc156657585)

[6.2.4 Evaluation 10](#_Toc156657586)

[6.3 Iteration 3 – Map Generation 10](#_Toc156657587)

[6.3.1 Design 10](#_Toc156657588)

[6.3.2 Implementation 11](#_Toc156657589)

[6.3.3 Testing 11](#_Toc156657590)

[6.3.4 Evaluation 11](#_Toc156657591)

[6.4 Iteration 4 – Enemy Movement 11](#_Toc156657592)

[6.4.1 Design 11](#_Toc156657593)

[6.4.2 Implementation 12](#_Toc156657594)

[6.4.3 Testing 12](#_Toc156657595)

[6.4.4 Evaluation 12](#_Toc156657596)

[6.5 Iteration 5 – Enemy Attack 12](#_Toc156657597)

[6.5.1 Design 12](#_Toc156657598)

[6.5.2 Implementation 12](#_Toc156657599)

[6.5.3 Testing 12](#_Toc156657600)

[6.5.4 Evaluation 12](#_Toc156657601)

[6.6 Iteration 6 – User Interface 12](#_Toc156657602)

[6.6.1 Design 12](#_Toc156657603)

[6.6.2 Implementation 12](#_Toc156657604)

[6.6.3 Testing 12](#_Toc156657605)

[6.6.4 Evaluation 12](#_Toc156657606)

[6.7 Iteration 7 – Player Upgrades 12](#_Toc156657607)

[6.7.1 Design 12](#_Toc156657608)

[6.7.2 Implementation 12](#_Toc156657609)

[6.7.3 Testing 12](#_Toc156657610)

[6.7.4 Evaluation 12](#_Toc156657611)

[7 Bibliography 12](#_Toc156657612)

# 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, we 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, we will be able to reference this document and come to a fix faster as we 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 we are 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, we can be sure that the systems work as intended before fully implementing them into the game.

## Stakeholders

The stakeholders of this project are the people who we must create the game in mind for. We 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, we 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. We will also decide on the development methodology that we 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 easy to create in our short timeframe.

Another benefit of Unity was that we 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 we 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 we are modifying.

## IDE

For this project we 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 we will be able to use pre-made functions specific to what we 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 we may make ourselves for the same purpose.

Visual Studio as an IDE has other features, 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.

# 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 we had to utilise assets for a regular dungeon, while still attempting to keep close to the theming that we were aiming for.

A screenshot of a video game

Description automatically generated

Figure - Tilemap for room layout

A video game screen with a dark background

Description automatically generated with medium confidence

Figure - Tilemap for obstacles

# Requirements Specification

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Requirement | User Requirement | Solves | Justification | Evidence | Success Criteria |
| 1.1 | Player movement | Inability to navigate the map | User needs to be able to move around the map |  |  |
| 1.2 | Player Attack | Player not being able to fight back against enemies | Player must be able to kill enemies to continue to the next room |  |  |
|  |  |  |  |  |  |

# 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 we 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.

### Implementation

For a project with a timeframe such as this

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.

### Implementation1

### Testing

### Evaluation

## Iteration 3 – Map Generation

### Design

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.

#### 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, and have the player simply move between them, 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.

#### 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, we 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 the 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.

#### Placing Rooms into Scene

#### 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 we have 4 different door presets, corresponding to the left, right, top, and bottom doors, this means that we have to find a way to affect all of these doors simultaneously. The best way to accomplish this would be to

### 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 then 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, we 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 we were 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, we 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 we 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, we 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.

### 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. | N (Evidence in 6.3.3.1) |
|  |  |  |  |  |

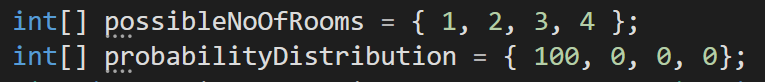
#### Setting number of Child Rooms

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 – Enemy Movement

### 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, and playing a set number of enemies within that area.

We 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.

As we are creating the first level of the game, this will likely be any number between 0 and 6 enemies, with the percentage chances of both fewer and higher numbers of enemies having a lower chance to occur than the median number. These enemies will be instantiated from a prefab, much like the player projectiles, and will be picked from a list of potential enemy types. The enemy types in the first version of the game will be two varieties of skeleton, vampires, and skulls. Each enemy type will move in the same way, but will have different attack patterns and ways of attacking the player.

Movement will be created

### Implementation

### Testing

### Evaluation

## Iteration 5 – Enemy Attack

### Design

### Implementation

### Testing

### Evaluation

## Iteration 6 – User Interface

### Design

### Implementation

### Testing

### Evaluation

## Iteration 7 – Player Upgrades

### Design

### Implementation

### Testing

### Evaluation

# Bibliography