BSc Project Report

Procedural Level Generation and VR

George Hanks

Computer Games Programming BSc

2019/2020

Khawla Al Hasan

Abstract

This project is investigating the use of procedural generation techniques within games to create levels / spaces for play. The project has conducted research into the methodologies that can be used to algorithmically produce data in order to produce a piece of software to demonstrate the methodology. The software has two parts, the first is a virtual reality environment, so a user can engage more immersively with the levels, and the second is a set of developer tools designed to make it easier to use the technology.

Key Words

Procedural Generation, Virtual Reality, Unity

Contents

1.0 Introduction

1.1 Brief

Video games are a popular form of entertainment, with an ever-growing consumer base that always expects the next best thing with new releases, however this means it will be increasingly difficult for video game developers to meet this higher demand. In computer science, procedural generation is a method of algorithmically producing data as opposed to manually, usually through a mixture of human-generated resources and algorithms, combined with the randomness and processing power of a computer. It is most notably used within video games in order to make content for the game, such as levels, environment details, and sounds. However, despite the potential benefits offered by using such methods, many video game developers choose not to employ such methods. This Project looks to improve the accessibility of procedural generation techniques and determine why there isn't widespread adoption. It will focus mainly on using procedural generation for making levels, but the techniques discussed can be transitioned to different areas of games.

1.2 Motivation

Using procedural generation within games can have many benefits for a development team, it can reduce workload on team members, and it can also allow the team to expand the scope of their project. However, many teams are hesitant to adopt procedural generation into their projects, improving the knowledge available to teams and providing tools for them will make adopting procedural generation techniques easier and less of a risk to the project. Furthermore, given the quick turnaround times expected by consumers it seems like procedural generation would be an appropriate technique for developers to employ as it can greatly increase output from the team. As an individual who spends a great amount of their spare time playing games, the idea of being able to have more content to play is a very interesting proposition. Because procedural generation can be used to make more game content, it is therefore great motivation for research and development of an application.

1.3 Research Questions

The primary research question for this project is 'How easy is it to develop procedural generation for use in games?', by answering this question we should be able to infer how effective procedural generation can be for modern games, Who should or should not be using procedural generation, and are there any unforeseen shortcomings to using procedural generation techniques. By answering these questions, it should hopefully reassure and encourage developers considering procedural generation for their games and or projects, which should hopefully make it more common place in modern games, which can only benefit the consumer and provide them with more interesting games to play.

1.4 Aims & Goals

The aim of this project is to investigate the ease of utilizing procedural generation within the context of games, and how they can be improved upon with a focus on making it easier for people with less programming knowledge to be able to use the system.

An initial goal of this project is to investigate the current research on procedural generation and to evaluate common trends. After the research is completed, a Windows application will be developed inside the Unity engine with the purpose of exploring the development process of a game that uses procedural generation. This application will serve both to explore methods of making development tools more friendly, and as a demonstration of the capabilities of such as system in the form of a game.

- The game will need to include movement mechanics for the player, these will allow the user to explore the level they have been placed into and interact with objects in the level. The player needs to be able to look around the level and move through it.
- The player needs to be able to interact with enemies in the level, the player will need to be able to attack the enemies and be attacked by the enemies.
- The levels need to be based on or created by procedural generation. This could be a purely procedural or a mixture of hand made and procedural.
- The game will be a functional prototype, it will not be feature complete or finished product.
- The software must be compatible with Windows 1903 or later.
- The software must be compatible with Steam VR 1.8 or later.
- A test plan will need to be created and executed in order to confirm features are functioning correctly.

1.5 Finished Product

The finished product of this project comes in two parts, the first is a virtual reality experience/game designed to demonstrate the procedural generation techniques investigated in this project, and the second is a set of tools built into the Unity game engine to aid with development, as part of the investigation into procedural generation workflow.

The virtual reality experience / game is a standalone windows application integrated with SteamVR and using the HTC Vive virtual reality hardware kit. This product as per the specification is not a polished product but demonstrates what procedural generation can be used for and the advantages of using it over other techniques.

The tools that are built into the Unity engine were not planned at the start of the project, they exist as a result of investigating what it is like to develop with procedural generation, as one of the aims of the project is to make procedural generation easier to access, these tools were seen as a solution to that problem. The tools developed for the project allow the management of assets relating to the level generation and creating new themes for levels.

2.0 Design & Planning

2.1 Overview of the Software

The software for this project is divided into two sections, the first is a collection of developer tools intended to simulate what a developer might see inside a game studio, and the second is a virtual reality environment used to explore what it is like to create with procedural generation and demonstrate what it can be used for. For the project to be successful the purposes and requirements for the project need to be established, and when each stage of development will take place. This was done using a functional requirements document to plan out a specification for the project and a Gantt chart to plan how much time should ideally be allocated to tasks.

2.1.1 VR Experience

The primary purpose of the VR experience is to explore what it is like to develop a VR game and investigate the advantages and limitations of the platform. Furthermore, it should be a showcase for the potential of procedural generation and virtual reality. The VR experience should allow the player to explore a level made by the procedural generation system in the project. Within the VR experience there should be things for the player to interact with such as enemies and props.

2.1.2 Developer Tools

As part of the development of the VR experience a set of tools was created in order to make it easier to design sections of levels. These tools were not planned from the beginning of the project, so no functional requirements exist for them. These tools are integrated into the Unity engine workspace so they should feel familiar with anyone who has used the software. They allow the user to control the parameters of the procedural generation and introduce handmade elements. These tools offer an insight into what it would be like developing a VR game.

2.2 Unity

The application was developed using Unity, a real time development platform primarily targeting games, but can also be used for films, cinematics, and architectural visualizations. Applications developed within Unity make use of C#, a multipurpose programming language, developed as an improvement to C and C++. Unity allows it users to code in C# with their preferred integrated development environment (IDE), Visual Studio was used for this project. Unity supports developing applications for nearly all platforms with 3D/2D graphics, this application will be developed for Windows desktop. Unity provides many tools within its development environment, providing an easy workflow for its users, which made it the ideal choice for this project.

2.3 GitHub & Source Control

To help manage and document the project, the source control software GitHub was used, source control software allows the user to save the project and keep track of the changes that where made with each new save so that if needed the project can be rolled back to a working save. GitHub was used for the project as it stores the project in the cloud, allowing for access from multiple workstations and in case of data loss easily retrievable.

2.4 SteamVR Plugin

To aid in the development of the virtual reality experience, the ‘SteamVR’ plug for Unity was used, this was done because the plugin includes scripts that make accessing and remapping inputs from the hand controllers more streamlined, it also manages the tracking of 3D models to the position of the hand controller in the Unity scene. With the plugin several assets are also provided which are free to use, these include some sounds, textures, and models, some sounds where used in the project for the spells. It is common for developers to use 3rd party plugins, and as one of the goals of the project is to emulate a developer workflow it seems to be an appropriate tool to use.

2.5 Virtual Reality Hardware

The virtual reality portion of the software was developed and tested with the ‘HTC Vive’, a high performance commercial virtual reality headset, which makes use of Valve’s SteamVR hardware allowing it to effortlessly communicate with virtual reality applications. Furthermore, the HTC Vive allows for room scale experiences which allow the user to walk around their environment, it also uses hand tracked controllers that provide more realistic interactions. For these reasons the HTC Vive was chosen for this project. However, as the HTC Vive uses the SteamVR standard, any virtual reality headsets that support this standard should work with the project.

2.6 UML (Unified Modelling Language)

The following diagrams represent the class relationships and dependencies using the unified modelling language, they were created using a web app that would output them as a PDF, but there are many other ways of doing this.

UML was used as part of the planning stage as it can make it easier to understand the relationships and dependences of a large system, such as the one being built in this project, furthermore it clearly outlines what data and functions are in each class, making implementation easier and faster. However, they represent an initial idea of what the system will look like, they will likely differ from the finished system. UML was effective in this project as a large portion of it is data focused, nevertheless changes were made to the original plan, but it is obvious that the final system is a variation of this plan.

See 'Figure 1' in appendix for UML example

3.0 Implementation

3.1 Development Methodology

A development methodology can be applied to clarify what will happen at various phases of the project and make the project management simpler, using such a process will enhance design, ease of development and finished product. The development method used for this project was inspired by the iterative waterfall method, that simplifies the development lifecycle by splitting it into five main phases.

The first phase of the lifecycle is where the goals/aims of the project should be compiled and established, so that they can be reference at later times to make decisions and evaluate performance. For this project it is covered in multiple documents, including the project contract and the functional requirements.

The second phase is where the goals/aims established in the first stage are used to draw up plans for the software/ application, these plans are created to make the development process easier and to ensure that the finished product meets the requirements of the project. UML diagrams was used in this project as part planning stage to plan out systems of the software, and a Gantt chart was used to plan time allocation for the project.

The third phase is where the plan goes into action, this is where the software is made using the plans established in the second phase. This phase usually takes up most of the project, and if the planning is insufficient can lead to time wastages that would take longer than it would have taken to plan properly.

The fourth phase is where the product is compared to the requirements set out in the first phase to check that it meets it requirements, test plans are also created to test systems that were not outlined in the planning phase. The outcomes of these tests are recorded and are used to plan the final phase.

The final phase is where the results from the fourth phase are used to fix any problems with the product and create any missing functionality. A small run of test is then conducted at the end to check that the changes have the desired effect.

Having a development plan based on this style of iterative design, allows retroactive development and changes to be made, moving away from a linear style means that the project is not fixed on its initial goal, and it can evolve naturally. Furthermore, it means that any unforeseen obstacles with scheduling and limitations on resources can be worked around.

3.2 Development Preparation

For a project to be successful it is important to prepare and make certain design decisions at the start that will be more difficult to implement or enforce later as the project grows. These decisions are made at the start of the project to alleviate confusion and to hopefully assist progress. These types of decisions would take place in a development studio, therefore offering insight into what developing a game or piece of software is like.

3.2.1 File Structure

An organized file structure can greatly improve efficiency within a project as developers won't have to be searching though directories to find files. Furthermore, it should make merging and committing of the project easier with source control as it divides the project into easier to understand chunks. For this project the file structure was based on the types of assets used within a game, and whenever possible loose files were never in a folder alongside other folders, this was done to keep the file structure cleaner, and make it easier to manage assets.

3.2.2 Naming Conventions

Standardization of names are common and sometimes required in large projects to reduce confusion, there are many variations of naming conventions that can be argued for and against. For this project the naming was kept simple, member and static variables used the Pascal Case standard and were prefixed with 'm_' and 's_' respectively. Local variables and parameters use the Camel case standard, but parameters are prefixed with an underscore. Functions are like Static and Member variables and are named with Pascal case but with no prefix. Using this standard makes it easier to find variables with a code editor and was easy to follow without becoming overly complicated.

Type	Examples
Member Variables	m_Health, m_Range
Static Variables	s_Instance
Local variables	rayCastHitInfo, timeLeft, selectedObject
Function parameters	_Target, _Size, _Speed
Function	GetDistance(), DoDamage()

'Table 1' Naming Convention Examples

3.2.3 Default Template Script

When creating new classes and scripts Unity provides a default script template that implements some commonly used functions, this template can be customized by the developer to their preferences. At the start of the project the template was modified to include regions to help organize the code, and an abstraction of Unity's console debug function to also include the name of the script the message is coming from, this function can also be disabled using a Boolean called m_Debug. These changes make development and debugging easier, as the code would be faster to read, and any messages could be taken straight back to the source without opening a code editor.

See 'Figure 2' in appendix for New Default Script

3.2.4 Scriptable Objects

The ScriptableObject class in implemented by the Unity engine, it acts as a data container that is independent from a class instance and is saved to the project folder. This means that the same data can be used in multiple places without having to copy the data, saving memory. Furthermore, with scriptable objects it is very easy to create and save multiple configurations of the same base data container, making development with these objects very easy. An example scenario would be if you wanted a large number of items within a game, with scriptable objects it would be a very easy process to create these items, as they would just be configurations of a single base object. Scriptable objects can also have functions which allow for even more possible scenarios. Because of its usefulness scriptable objects are used multiple times throughout this project.

3.3 Dungeon Generator - Definition of terms

In order to be consistent through development and testing, a set of terms where defined for the dungeon generator, these names were chosen as they are simple and contain enough detail to inform a person about the functionality / purpose of an object.

- Dungeon – A dungeon is all of the objects that make up a level, it is sometimes known as a maze/labyrinth, it can contain one or more rooms and zero or more connections. A dungeon can be as simple as the data that describes the layout, or it can contain all the 3D models to make a playable environment.
- Room – A room is a section of a dungeon, a room is made up of one or more cells and one or more doors. For a room to exist it must have at least one connection to the rest of the dungeon.
- Door – A door is an entrance/exit of a room, it will be connected to another door with a connection, it also has a direction which faces outside from the room it is a part of, it takes up one cell.
- Connection – A connection is link between two doors of different rooms, it allows the player to move between rooms.
- Cell – A cell is a single section of the grid that makes up the dungeon, all rooms, doors, and connections are aligned to a grid.
- Palette – A Palette is a collection of rooms that the dungeon generator can pull from when building the dungeon.

3.4 Dungeon Generator - Algorithm

All objects within the dungeon are aligned to a grid, the grid is made up of many cells and can be infinitely large. The grid is used to check for overlaps when placing rooms and connections, and for debugging the layout of a dungeon. The data inside the grid can be almost anything, this algorithm uses an enum. Having a grid reduces the number of possible configurations of the dungeon but makes the outcome more controllable and consistent, furthermore it makes designing rooms easier as they must conform to a standard.

Overlap checking is done by iterating through all the cells that make up a room/connection, if a cell already exists in the grid at a location then an overlap has been found, and the algorithm can respond appropriately. Overlaps need to be checked for as we don't want rooms inside one another, we want each room to be its own distinct environment, this makes the job for designers easier, as they can more easily control what happens inside a room and make it feel unique.

The dungeon is made by connecting rooms, which are defined with a width and height and a list of doors, a room can be any size and can have any number of doors if there is space around the edge of the room for it in the dungeon. A room can also have what is known as an empty cell, these are cells that fit within the bounds of the room but don't take up any space, when the room is added to the grid these empty cells are removed afterwards, this is done so that rooms can be more closely placed to one another and allows more interesting shapes.

1. The first step in the algorithm is to place a random room at the centre of the grid, all rooms in the dungeon will be connected to this room. This room can act as the start point of the dungeon for the player, but the start point can be any place in the dungeon if the designer chooses.
2. The second step is to pick a random room from the dungeon and try to spawn a room off it. This is done by picking a random door that doesn't already have a room connected to it, this is now an exit door, and then pick a random connection length.

3. The third step is to pick a random room from the palette, and then pick a random door that is on the opposite side to the exit door, this door is now an entrance door.
4. The fourth step is to check that the new room is not overlapping any other rooms in the dungeon, if it is then the algorithm goes back to step two. If there are no overlaps then the data about the room can be added to the dungeon data, then the two doors need to be marked as connected and data about the connection needs to be added to the dungeon data.
5. Steps two to four then repeat until the desired number of rooms has been reached.

3.5 Dungeon Generator – Classes & Data Structures

3.5.1 DG_DungeonGenerator.cs

The dungeon generator class is the biggest class in the project in terms of data and contains the most lines of code in a single file, it contains many public and private functions. Due to the size of the class very little of the code will be presented in this report but will be available in the appendix. This class implements the dungeon building algorithm and stores all the data the describes the dungeon. This code can be run either inside the Unity editor as an editor script, or at run time during a release build of the application. The class has several parameters that can be modified inside the Unity editor, they allow the user to change how the dungeon generator will behave, examples of parameters are the number of rooms, the length of connections, and how much space must be around a room. This class also handles the visual debugging of the dungeon, this is done by drawing coloured squares at the location of cells, the colour indicates what type of cell it is, these squares can be seen in the Unity editor scene window. Keeping in line with emulating a developer workflow, much of the class it organized into functions, this helps reduce the number of lines of code and makes the document easier to read. The two most important functions in this class are 'GenerateDungeon' and 'AddRandomRoom'.

See 'Figure 3' in appendix for 'DG_DungeonGenerator.cs' class

'GenerateDungeon' is a public function that is the highest abstraction of the dungeon building algorithm, its first operation is to check that all the required data to build a dungeon has been provided, and then it will try to generate the first room in the dungeon. The function then runs a 'while' loop that adds rooms to the dungeon, using the 'AddRandomRoom' function, and maintains how many rooms have been added. The while loop also keeps track of how many times the algorithm has failed, and when a threshold has been reached it breaks the while loop, this is done as if the algorithm keeps failing it is effectively an infinite loop, which are bad as they can take up all the processing time of a computer.

See 'Figure 4' in appendix for 'GenerateDungeon' Function

'AddRandomRoom' is likely the most complicated function in the class, it is a private function which returns a bool depending on whether it can add a room to the dungeon. This function handles picking the room and door to add a connection to, the new room that is going to be added and the entrance door to the new room. At every stage of the function there a check to ensure that the current attempt to add a room is compatible with the setting of the dungeon generator. If an attempt is successful then all the relevant data is added to the class, this includes the cell data, the room data and the connection data.

The other major task this class completes is adding all the models and game objects to the scene, these objects are stored as prefabs in the Unity file system and are referenced in the information about the rooms, this part of the class will also add the meshes for the connections at the same time.

This function makes its random decisions using dice rolls or pseudorandom number generation, these dice rolls also can have a predefined seed, the seed is the number used to initialise the number generator, this means that if the same seed is used the number generator will always give the same result. This means if you

wanted to transfer a level all you would need is the seed rather than a large multi megabyte file. Dice rolls are the foundation of the dungeon generator and are the primary method of procedural generation.

See 'Figure 5' in appendix for 'AddRandomRoom' Function

3.5.2 DG_Palette.cs

This class is a simple data container that stores all the data about a set of rooms, this is data can be filled out manually or preferably using palette designer tools. This class inherits from the scriptable object class so can be easily reconfigured and stored within the project directory and be used within multiple places in the project. A refence to a palette is used in the dungeon generator as the pool of rooms to choose from.

3.6 Dungeon Generator – Unity Editor Tools

3.6.1 DG_RoomVolume, DG_DoorVolume, & DG_EmptyCell

These classes are the main tools for designing sets of rooms, they allow the user to edit parameters and move objects around in the 3D workspace to design in unique ways. All three classes have a debugging functionality to show the user how big the object is and the direction it is facing, making it easier for the user to get the outcome they want. This debugging is implemented using the Unity Gizmos system and draws a wireframe box around the object.

See 'Figure 6' in appendix for 'Unity Editor Room Design'

DG_RoomVolume is the most complicated out of the three classes in terms of functionality and data stored, the purpose of this class is to collect all the data from the scene to create the configuration for a room. It achieves this by first recording the data about the doors and the empty cells within the bounds of the room, the bounds of the room are defined by the m_Size public variable. It stores the position and direction of the doors and only the position of the empty cells in a cache. The second purpose of the room volume is to collect all the relevant gameObject data, the room volume is only concerned about the gameObjects that are within the bounds of the room. It then separates out the gameObjects that are also within doors, the gameObjects that are within doors are only used when a door has no connection.

DG_DoorVolume & DG_EmptyCell are very simple classes, they are used as markers for the room volume class to represent the position of the objects within the room. They don't store any data and have no functions, there only purpose is to be a visual debugging tool.

3.6.2 DG_PaletteDesignerWindow.cs

The palette designer window class is the main class in the tools that are used to design the rooms that the dungeon generator can use when making levels. This class inherits from the Unity class EditorWindow which allow it to appear as a menu inside the Unity editor. The menu is opened using a button in a dropdown that is added to the menu bar at the top of the Unity editor. This class handles setting up a new scene for room design, spawning in prefabs that define rooms and doors, and saving the information about the rooms as prefabs. Many of the functions a user needs to run are accessed via buttons that appear in menu, these buttons are added using built in Unity editor functions.

The main function of this class is to compile all the data from room volumes in the scene and then store that data in the project directory. It achieves this by going through every room volume in the scene and creating a scriptable object for that room, it then fills out the object with the relevant data from the room. After it has collected the data it will create and save a prefab for the base objects in the room and all the doors inside the room. The scriptable object for the room is then told the location of these prefabs so that they can be loaded in later. Once all the rooms have be compiled, the references to them are all saved in a palette so that it can be used by the dungeon generator.

See 'Figure 7' & 'Figure 8' in appendix for 'DG_PaletteDeignerWindow' class and '*Palette Designer Window Unity Editor*'

3.7 Player Movement & Interaction

3.7.1 SteamVR Input Manager

The InputManager class acts as an interface for all the SteamVR functions inside the project, it retrieves the input values from the SteamVR API and stores them inside the class to be accessed. Inputs for SteamVR are mapped to buttons using InputActions, Input Actions are a flexible way of assigning buttons to actions that can be easily changed by the developer or the end user. The actions can also be easily mirrored over both controllers so that only one set of actions is needed, however which hand needs to be specified when retrieving the input data. The input data is stored as a mixture of floats, bools, and vectors, and is updated every frame. The input data can be accessed publicly, this allows for clearer code that is like the built in Unity input manager so should feel familiar.

The class makes use of the singleton design pattern so that it can be accessed from any script within the project, this is achieved by having a static InputManager reference store in the class, which is set to itself when the project runs. Being a singleton also ensures that there is only one instance of the class at a time, which could lead to undesirable behaviour as this class handles user input and should not be ambiguous.

See 'Figure 9' in appendix for 'InputManager.cs'

3.7.2 Movement & Interaction

The player can move and interact within the game using spells, spells are actions that the player can trigger using the hand controllers. All spells in the game inherit from the base 'Spell' class, allowing for polymorphic class design, the base class contains data and functions that all spells have/ can use. The base class inherits from the ScriptableObject class, allowing for easy configuration of new spells.

Spell are stored within the SpellManager class, which manages equipping and unequipping spells to the players hands, the spell manager allows the player to switch between 4 different spells, changing of spells is triggered by the user interface. The SpellManager allows the player to have any combination of spells for both hands, spells are equipped using a function that takes an integer as parameter which is the number of the spell to be equipped.

The user interface for the spell manager is triggered by touching the touchpad on the controllers, when triggered a radial menu will appear above the controller and a moving dot will represent the players thumb position on the touchpad. The radial menu has four section on it representing the 4 spells the player can equip, when the player presses down on the touchpad, they will trigger the event to equip the corresponding spell.

See 'Figure 10' in appendix for 'Radial Menu'

3.7.3 Enemies

There are two types of enemies within the project, melee and ranged, both are based upon the same base class with handles movement and attack timing. This AI is very simple and is the minimum viable solution for AI in a game, they were designed this way as they are not meant to be the focus of the application. The AI navigates the world using the Unity NavMesh and NavMeshAgent systems, this allows them to navigate around the level calculating paths to the player. In order to make the application fair the AI must have line of sight on the player before they can start pursuing, but once they have started, they will not return to their previous state. This line of sight is calculated using a ray cast and is a very efficient solution to this problem. However, using the built in components does mean the navigation data does need to baked in the editor, this means it was not possible without building a custom navigation system to have levels generated at run time, the current solution is to generate a level in the editor and the bake the navigation data to the level. The 3D

models for these enemies were taken from the free website ‘Mixamo’, Mixamo is a website that provides character models and animation for projects.

3.8 Walkthrough of the Application

3.8.1 Main Menu

When the user starts the application, they will be presented with the main menu, the main menu displays information about the project and how to play. This information is presented on a large screen that is easy to read, the user can activate the buttons on the screen using the right-hand controller and pulling the trigger. Clicking on the ‘Play’ button will start the game, the ‘Controls’ button will show the controls for the game, and the ‘Quit’ button will exit the application.

See ‘Figure 11’ in appendix for ‘Main Menu’

3.8.2 Dungeon Level

When the user starts the game, they will be placed into the dungeon level, they will be able to look around the level by rotating their head in real life. The user should be able to see two controllers that represent the user’s hands, these controllers will follow the user’s hands when they move them.

The player can move about using the teleport spell, which can be equipped by touching the touchpad and moving their thumb to the top segment and pressing down, there should be a sound to confirm the spell has been equipped. The user can then point and pull the trigger on the controller to show the target for the teleport spell, and when the trigger is released the user will move to the target’s location. The teleport spell has a short cooldown and can be equipped to both hands if needed, so is easy to use.

See ‘Figure 12’ in appendix for ‘Teleport Spell’

After exploring the level for a while, the user should see some enemies, these enemies can be attacked with the fireball spell, which can be equipped the same way as the teleport spell but instead pressing down on the segment on the right. The user can then hold down the trigger and they will continually cast fireball spells, it will take four fireballs to kill an enemy.

The user can receive damage from these enemies which is indicated by the colour of the screen changing to red, if the user takes too much damage the screen will turn black, this indicates the user is dead and then they will be taken back to the main menu. If the user can avoid damage for a short period of time, they will gain health and the screen will return to normal.

See ‘Figure 13’ in appendix for ‘Player Health’

As the user is exploring the level, they should be able to interact with props in the level, they can do this with the grab spell, which is on the left in the spell menu and the push spell which is on the bottom of the spell menu. The grab spell has a long range and will pull the object to the player’s hands, and the push spell has a short range and will add force to an object. If the user can kill all the enemies within the level, they will be taken back to the main menu where they can play again or quit the application.

3.9 Testing

3.9.1 Testing Overview

Developing or testing software and games is an important part of the process, its primary goal being to uncover defects before reaching the end user. Defects in a game or software may include accidental functionality, freezing, stuttering, and crashing, all of which may spoil a user’s experience and may harm a product's reputation, therefore catching them during development is very important.

During the bulk of development there will be lots of ad hoc testing, it is common for this to be unplanned and not recorded, this is because as a developer or programmer it is easier to solve problems on the fly. This kind of testing is common when implementing a feature for the first time, as it is not necessary to test partially complete systems or functions as the code base could change dramatically by the end.

When it comes to planned testing it will be based on the functional requirements of the project, however as the goals of the project can change this is not always the case, and new tests will need to be planned to accommodate these changes. At the start of the project the descriptions of these tests will be broad as the specifics of the how the code would function will not have been ironed out. Towards the end of the project it will be possible to design more specific tests as the goals of the project won't change significantly, and specifics about the inner working of the code will be known.

3.9.2 Black Box Testing

Black box testing is a very simple way to test if a system works, however it does not concern itself with how the systems works, all it's concerned about is the inputs and outputs. During this testing valid and invalid inputs will be used to create positive and negative test cases. A set of expected outputs are then created by the designer of the test to compare with the outputs. It is then very easy for someone with little knowledge of the system to test and record the outputs and compare them with the expected results. This form of testing will take place when it is believed that a part of the project is finished, it acts as a final check to find any missing features or bugs.

3.9.3 Functionality Testing

Functionality testing is done to check if the game or project is performing against the specification, it is a very easy way to tell if a system works as what should happen is predefined. It will investigate errors that will affect user experience such as audio, visuals, and gameplay. This kind of testing will occur when a significant change has been made to the game or project, it is unnecessary to run these tests on minor changes as they are unlikely to affect the games performance.

3.9.4 Play Testing

Play testing is done by having exposure to the game, it tests abstract features of the game such as fun, balance, and difficulty. This testing will be mostly opinion based and anecdotal, but it is still valuable in order to build a successful project. This testing will occur all throughout the project, as it is an easy and efficient way to test incomplete systems, however this type of testing cannot be run in isolation, other forms of testing will need to be conducted to support the findings of play testing, as a game passing these tests doesn't automatically pass other more systematic tests.

3.9.5 Regression Testing

Regression Testing is done after a code fix, or upgrade to a system or function, this is done to check that the fix has not altered the existing code. These tests will be reruns of previous tests to confirm the changes has had the desired effect and not negatively affected the project. Depending on how often testing is done, regression testing may not be necessary as the error will be picked up in the next set of tests.

3.9.6 Testing Results

Most of the testing for the software happened at the end of the project or when a system was considered finished, however these two events would usually coincide with each other, so most of the testing fell within the allocated development time for testing. The primary method of testing was black box testing and was recorded in a testing table, most tests were successful on the first run through, however some tests did need to be rerun due to small bugs. Limited regression testing was then conducted to see if any changes had occurred which all

passed, so the previous black box test results were not changed. It is likely that most tests passed on the first attempt due to the constant 'ad hoc' testing that happened throughout the project, however another factor could be that the tests are too broad in their requirements, so some bugs cannot be seen with the tests. However, the goal of the project was not to produce a bug free finished product but a functional prototype, so the testing conducted was acceptable.

See 'Table 2' in appendix for black box testing table

4.0 Conclusion

4.1 Evaluation of the Software

This development project aimed to investigate the ease of utilizing procedural generation within the context of games, and how they can be improved upon with a focus on making it easier for people with less programming knowledge to be able to use the system. Based on qualitative analysis of the software produced by this project, it can be concluded that the software has completed its goals and fulfils all the requirements set out in the specification for the project or software. While the visual quality and user friendliness of the VR experience may not meet the standards expected by consumers, it demonstrates that procedural generation can be practical, and employed for use within games.

4.2 Further Improvements to the Software

Based on the results from the software and despite the success of the project, it is obvious that many improvements can still be made to increase the capabilities of the software and make it more accessible. Further improvements could address the solution of storing data, as the current solution whilst functional is not very scalable and could slow down larger projects. A solution to this problem would be to find a suitable data format such as XML and store the data in a single file. Furthermore, improvements could be made to the modularity of the project, for example the current software can only design levels in 2D, a considerable improvement would be the inclusion of 3D level design, this would widen the amount of options to developers and generally give better results from the algorithm.

4.3 Evaluation of the Research

The primary research question of this project is 'How easy is it to develop procedural generation for use in games?'. Informed by qualitative analysis and reinforced by anecdotal evidence, it can be concluded that procedural generation has a place in the games industry and can produce experiences for consumers that can meet the high standards they have, however this conclusion does come with some caveats. Procedural generation will not automatically make a game better, it should not be employed for the sake of just using it or having it as a buzz word for your marketing team, it should be implemented in a way that will objectively make the game better or by a narrative driven need. For example, there is little need for procedural generation to exist in a football game, the rules and setting of football are predefined and would not benefit from randomness.

Furthermore, depending on the size of the development team and the scope of your game, utilizing procedural generation may overwhelm your development team and remove them from developing more vital systems in the game, if the scope of your game is small and the project has a small turnaround time, you are unlikely to reap the benefits of procedural generation, as the time spent implementing the technology could take up more time that it would have taken to finish the game. On the other hand, if the scope of you game is large and you have a long lead time for completion, procedural generation should be seriously considered, especially if you

plan to continue supporting the game after the release, the benefits may not feel large over a short amount of time, but these benefits will stack up over the duration of the project.

4.4 Further Research

To further understand the implications from these findings, further research studies could investigate the applications of procedural generation for other areas of games. While this study focused on levels, there are many parts of games that could see improvement if procedural generation techniques where applied. An example would be quests or stories in role playing and adventure games, if stories and activities could be developed using existing assets by the game rather than a developer it would greatly widen the variety of content for players, and allow developers more time to focus on parts of the game they want to produce.

4.5 Reflection on the Entire Project

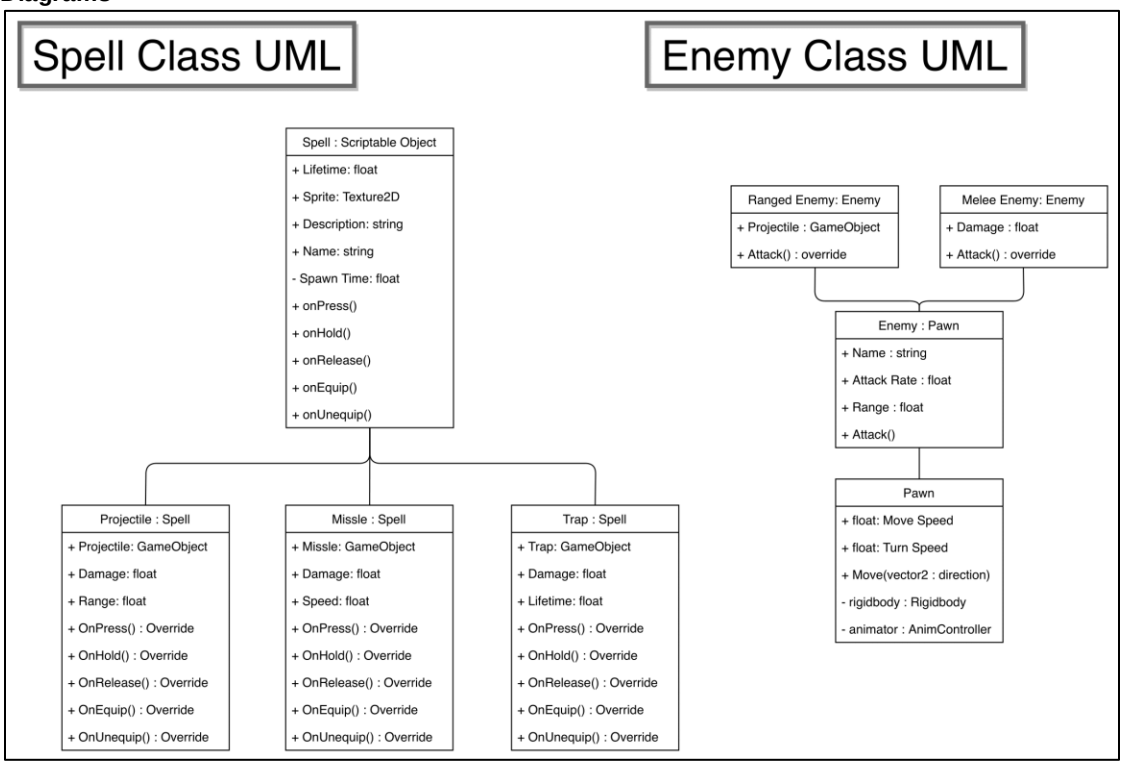
The software produced by this project meets all expectations, it implements the functionality set out in the designs and works as envisioned. Therefore, despite certain development distractions and maybe the visual quality not being as good as desired, it was able to reach its requirements, therefore it can be deemed effective. A feature of significance is the speed at which the development tools operate, in this regard they exceed expectation.

The project management despite the success of other parts of the project was not effectively enforced, more could have be done to ensure that small objectives were completed on time, a contributing factor to this is the scope of the project, which grew as the project went on, a smaller more easily achievable goal perhaps should have been chosen for an individual project. The setback at the start of the project due to supervisor management, and the uncertainty to access to resources towards the end of the project did cause some concerns, but even with these factors the project was able to be completed.

Once the project had got going the supervisor support was excellent, regular meeting and contact through other channels meant that the project stayed mostly on track. There was some hindrance due to miscommunication, which meant things being explained more than once, on both sides of the conversation, however it did not block progress being made. The best solution for this would be to have more regular meetings, either face to face or online, as more progress was generally made at these.

Appendix

Figure 1 UML Diagrams



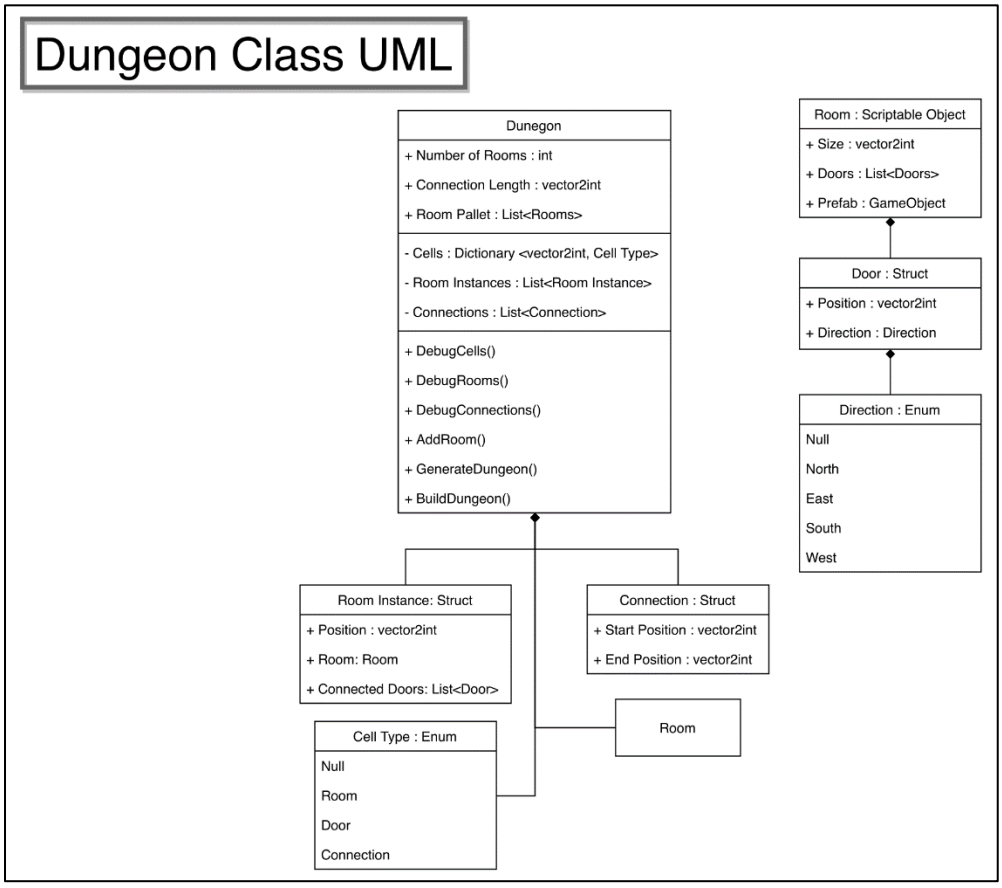


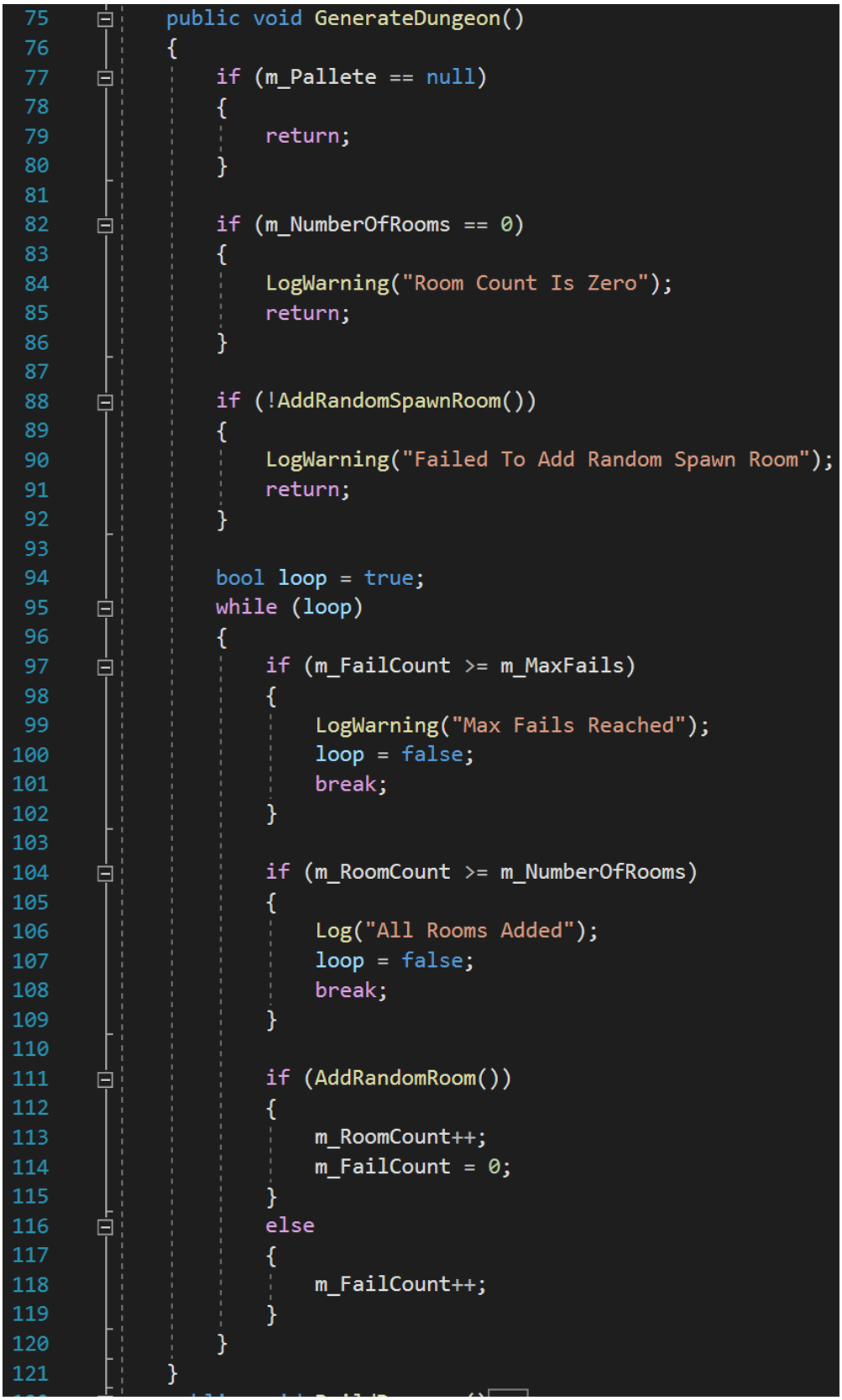
Figure 2 Default Template Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class #SCRIPTNAME# : MonoBehaviour
{
    [Header("#SCRIPTNAME# Options")]
    #region Public Variables
    public bool m_Debug = true;
    #endregion

    #region Private Variables

    #endregion

    #region Unity Functions

    #endregion

    #region Public Functions

    #endregion

    #region Private Functions
    void Log(string _msg)
    {
        if (m_Debug)
        {
            Debug.Log("[#SCRIPTNAME#][" + _msg + "]");
        }
    }
    #endregion
}
```

Figure 3 DG_DungeonGenerator.cs Class

```
4   public class DG_DungeonGenerator : MonoBehaviour
5   {
6       #region Public Variables
7       public int m_Seed;
8       public bool m_Seeded = false;
9       public bool m_Debug;
10      public int m_NumberOfRooms;
11      public Vector2Int m_ConnectionLength;
12      public DG_Palette m_Pallete;
13      public int m_RoomPadding = 0;
14      public float m_DrawSquareSize = 0.45f;
15      public int m_MaxFails = 100;
16      #endregion
17
18      #region Private Variables
19      private Dictionary<Vector2Int, DG_CellType> m_Cells = new Dictionary<Vector2Int, DG_CellType>();
20      private List<DG_RoomInstance> m_RoomInstances = new List<DG_RoomInstance>();
21      private List<DG_Connection> m_Connections = new List<DG_Connection>();
22      private int m_FailCount;
23      private int m_RoomCount;
24      #endregion
25
26      #region Unity Functions
27      private void OnEnable()...
31      private void Awake()...
34      private void Start()...
37      private void Update()...
40      private void FixedUpdate()...
44      #endregion
45
46      #region Public Functions
47      public void DebugCells()...
67      public void DebugRooms()...
74      public void DebugConnections() { }
75      public void GenerateDungeon()...
122     public void BuildDunegon()...
127     public void ClearDungeonData()...
135     public void ClearDungeonMeshes()...
142     #endregion
143
144     #region Private Functions
145     private bool AddRandomRoom()...
234     private bool AddRandomSpawnRoom()...
247     private void AddRoomToDungeonData(DG_Room _Room, Vector2Int _Position)...
273     private void AddConnectionToDungeonData(Vector2Int _StartPosition, DG_Direction _Direction, int _Length)...
304     private void BuildRoomMeshes()...
322     private void BuildConnectionMeshes()...
345     private bool CheckRoomOverlap(DG_Room _Room, Vector2Int _Position)...
361     private bool CheckConnectionOverlap(Vector2Int _StartPosition, DG_Direction _Direction, int _Length)...
398     private void DrawSquare(Vector2Int _Position, Vector2Int _Size, Color _Color)...
409     int RandomNumberInRange(int max, int min = 0)...
422     Vector2Int DirectionToVector(DG_Direction _Direction)...
443     List<DG_Door> FilterDoorsByDirection(List<DG_Door> _Doors, DG_Direction _Direction)...
455     DG_Direction GetOpositeDirection(DG_Direction _Direction)...
477     Vector3 GridToWorldAxis(Vector2Int _gridPosition)...
481     float DirectionToZRotation(DG_Direction _Direction)...
489     Vector3 DirectionToEulerRotation(DG_Direction _Direction)...
497     private void Log(string _msg)...
502     private void LogWarning(string _msg)...
507     #endregion
508 }
```

Figure 4 GenerateDungeon Function



```
75      public void GenerateDungeon()
76      {
77          if (m_Pallete == null)
78          {
79              return;
80          }
81
82          if (m_NumberOfRooms == 0)
83          {
84              LogWarning("Room Count Is Zero");
85              return;
86          }
87
88          if (!AddRandomSpawnRoom())
89          {
90              LogWarning("Failed To Add Random Spawn Room");
91              return;
92          }
93
94          bool loop = true;
95          while (loop)
96          {
97              if (m_FailCount >= m_MaxFails)
98              {
99                  LogWarning("Max Fails Reached");
100                 loop = false;
101                 break;
102             }
103
104             if (m_RoomCount >= m_NumberOfRooms)
105             {
106                 Log("All Rooms Added");
107                 loop = false;
108                 break;
109             }
110
111             if (AddRandomRoom())
112             {
113                 m_RoomCount++;
114                 m_FailCount = 0;
115             }
116             else
117             {
118                 m_FailCount++;
119             }
120         }
121     }
```

Figure 5 AddRandomRoom Function

```
private bool AddRandomRoom()
{
    if (m_Pallete.m_Rooms.Count < 1)
    {
        LogWarning("No Rooms In Palette");
        return false;
    }
    if (m_RoomInstances.Count < 1)
    {
        LogWarning("There Are No Room Instances");
        return false;
    }
    DG_RoomInstance startRoomInstance = m_RoomInstances[RandomNumberInRange(m_RoomInstances.Count)];
    if (!(startRoomInstance.m_ConnectedDoors.Count < 4))
    {
        Log("Start Room Has Too Many Connections");
        return false;
    }
    if (startRoomInstance.m_Room == null)
    {
        LogWarning("Start Room Has No Room Info");
        return false;
    }
    DG_Room startRoomData = startRoomInstance.m_Room;
    if (startRoomData.m_Doors.Count < 1)
    {
        LogWarning("Start Room Has No Doors");
        return false;
    }
    DG_Door exitDoor = startRoomData.m_Doors[RandomNumberInRange(startRoomData.m_Doors.Count)];
    DG_Room newRoom = m_Pallete.m_Rooms[RandomNumberInRange(m_Pallete.m_Rooms.Count)];
    if (newRoom.m_Doors.Count < 0)
    {
        LogWarning("New Room Has No Doors");
        return false;
    }
    List<DG_Door> potentialDoors = FilterDoorsByDirection(newRoom.m_Doors, GetOpositeDirection(exitDoor.m_Direction));
    if (potentialDoors.Count < 1)
    {
        LogWarning("New Room Has No Potential Doors");
        return false;
    }

    DG_Door entryDoor = potentialDoors[RandomNumberInRange(potentialDoors.Count)];

    int connectionLength = RandomNumberInRange(m_ConnectionLength.y, m_ConnectionLength.x);

    Vector2Int connectionStart = Vector2Int.zero, connectionEnd = Vector2Int.zero, newRoomLocation = Vector2Int.zero;

    if (connectionLength > 1)
    {
        connectionStart = startRoomInstance.m_Position + exitDoor.m_Position + DirectionToVector(exitDoor.m_Direction);
        connectionEnd = connectionStart + DirectionToVector(exitDoor.m_Direction) * connectionLength;
        newRoomLocation = connectionEnd + DirectionToVector(exitDoor.m_Direction) - entryDoor.m_Position;
    }
    else if (connectionLength == 1)
    {
        connectionStart = startRoomInstance.m_Position + exitDoor.m_Position + DirectionToVector(exitDoor.m_Direction);
        connectionEnd = connectionStart;
        newRoomLocation = connectionEnd + DirectionToVector(exitDoor.m_Direction) - entryDoor.m_Position;
    }
    else if (connectionLength == 0)
    {
        newRoomLocation = startRoomInstance.m_Position + exitDoor.m_Position + DirectionToVector(exitDoor.m_Direction) - entryDoor.m_Position;
    }


    if (!CheckRoomOverlap(newRoom, newRoomLocation))
    {
        if (CheckConnectionOverlap(connectionStart, exitDoor.m_Direction, connectionLength))
        {
            Log("Found Connection Overlap!");
            return false;
        }
        Log("Added Room");
        AddRoomToDungeonData(newRoom, newRoomLocation);
        startRoomInstance.m_ConnectedDoors.Add(exitDoor);
        m_RoomInstances[m_RoomInstances.Count - 1].m_ConnectedDoors.Add(entryDoor);

        if (connectionLength > 0)
        {
            AddConnectionToDungeonData(connectionStart, exitDoor.m_Direction, connectionLength);
        }

        return true;
    }

    return false;
}
```

Figure 6 Unity Editor Room Design

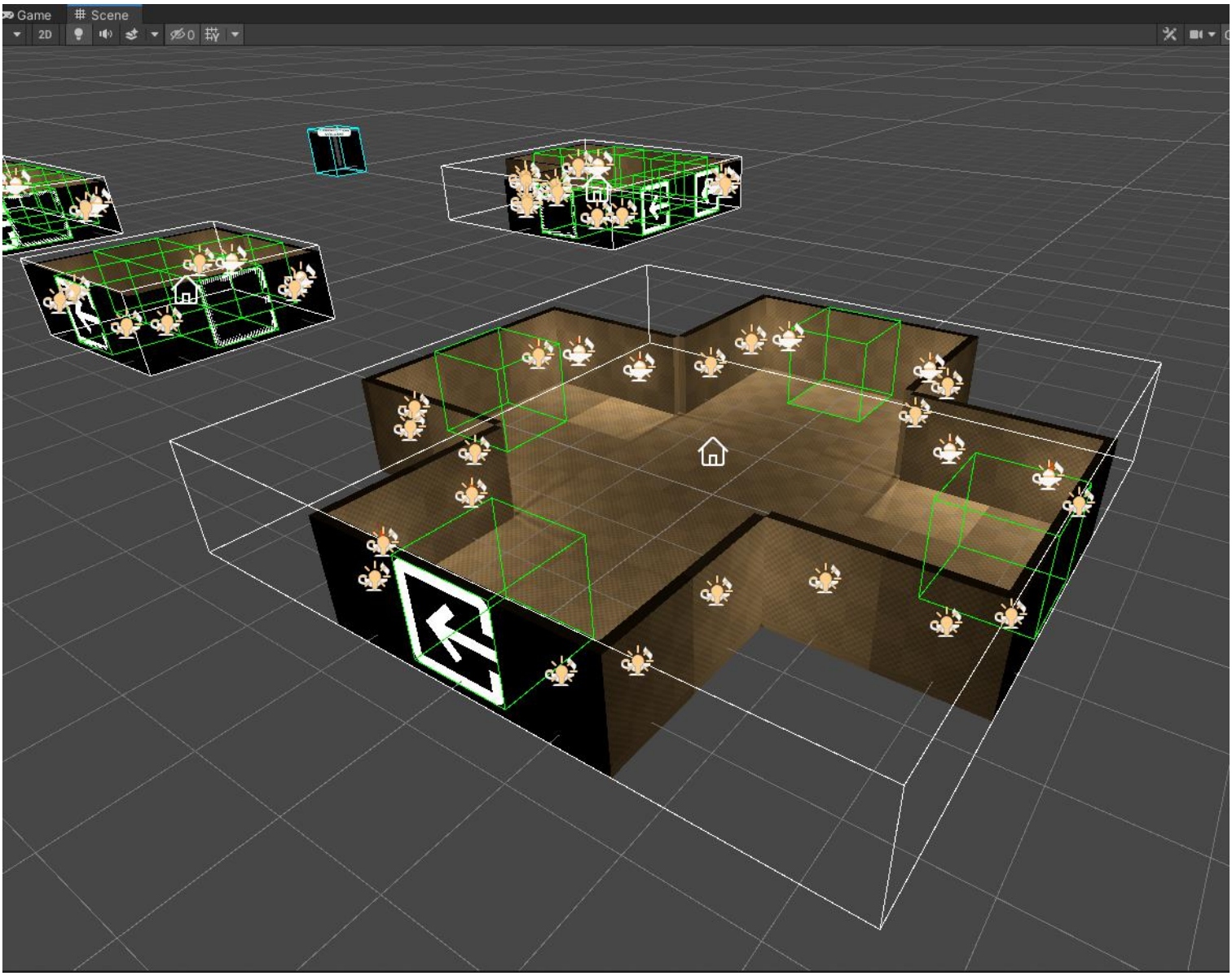


Figure 7 DG_PaletteDesignerWindow.cs Class

```
8   public class DG_PaletteDesignerWindow : EditorWindow
9   {
10  #region Private Variables
11      private Object roomPrefab;
12      private Object doorPrefab;
13      private Object emptyCellPrefab;
14      private Object connectionPrefab;
15  #endregion
16
17  #region Unity Functions
18  #endregion
19
20  #region Public Functions
21      [MenuItem("Dungeon Generator/Palette Designer")]
22      public static void ShowWindow()...
28  #endregion
29
30  #region Private Functions
31      private void OnGUI()...
87      private void SaveRooms()...
135     private void SaveConnection()...
160     private void CheckVolumePrefabs()...
167     private List<DG_RoomVolume> GetAllRoomVolumes()...
178     private GameObject CreateObjectFromTransformData(List<Transform> _Transforms, Vector3 _RootOffset)...
190     private void Log(string _msg)...
194     private void LogWarning(string _msg)...
198 #endregion
199 }
```

Figure 8 Unity Editor Palette Designer Window

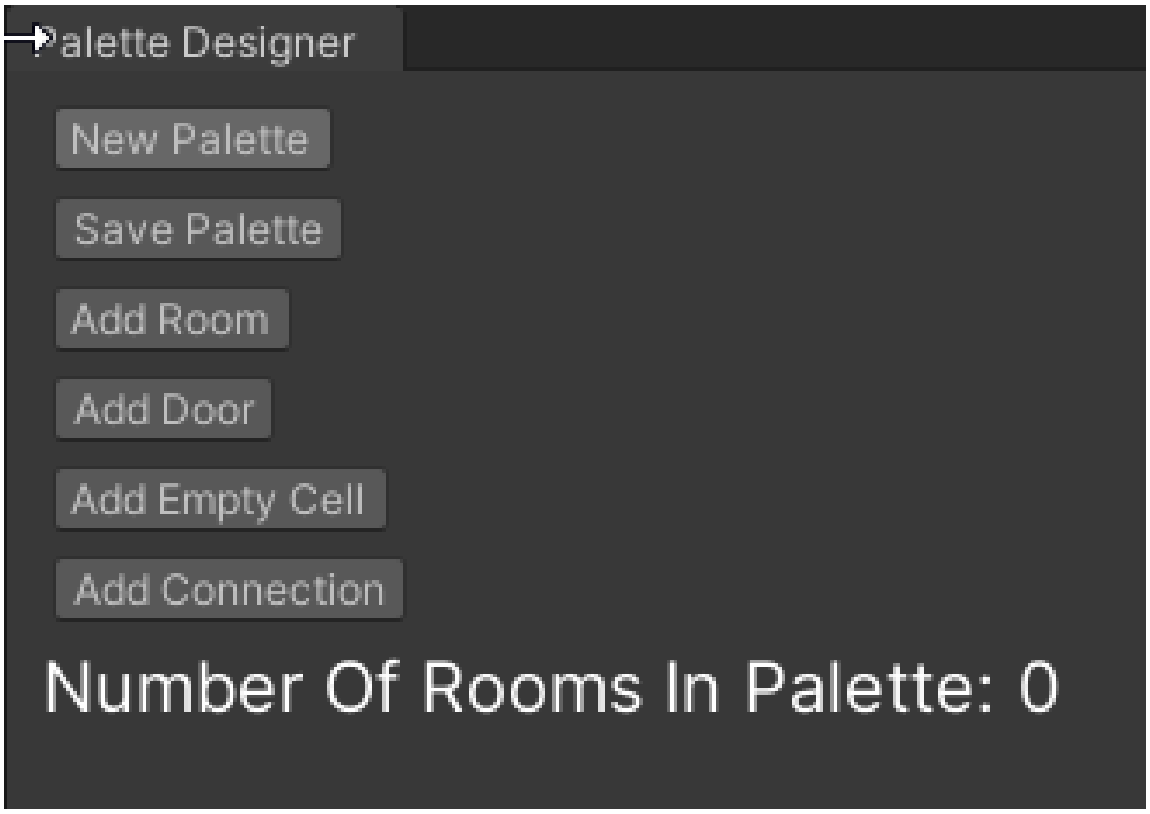


Figure 9 InputManger.cs

```
6   public class InputManager : MonoBehaviour
7   {
8       #region Public Variables
9       [Header("Input Manager Options")]
10      public bool m_Debug = true;
11      public static InputManager s_Instance;
12      [Header("Input Data")]
13      public bool m_touchpadTouchLeft = false;
14      public bool m_touchpadTouchRight = false;
15
16      public bool m_touchpadPressDownLeft = false;
17      public bool m_touchpadPressDownRight = false;
18
19      public bool m_touchpadPressUpLeft = false;
20      public bool m_touchpadPressUpRight = false;
21
22      public Vector2 m_touchpadPositionLeft = Vector2.zero;
23      public Vector2 m_touchpadPositionRight = Vector2.zero;
24
25      public float m_triggerPositionLeft = 0;
26      public float m_triggerPositionRight = 0;
27      #endregion
28
29      #region Unity Functions
30      private void Awake()
31      {
32          if (!s_Instance) s_Instance = this;
33      }
34      private void Update()
35      {
36          m_touchpadTouchLeft = SteamVR_Actions.default_TouchpadTouch[SteamVR_Input_Sources.LeftHand].state;
37          m_touchpadTouchRight = SteamVR_Actions.default_TouchpadTouch[SteamVR_Input_Sources.RightHand].state;
38
39          m_touchpadPressDownLeft = SteamVR_Actions.default_TouchpadPress[SteamVR_Input_Sources.LeftHand].stateDown;
40          m_touchpadPressDownRight = SteamVR_Actions.default_TouchpadPress[SteamVR_Input_Sources.RightHand].stateDown;
41
42          m_touchpadPressUpLeft = SteamVR_Actions.default_TouchpadPress[SteamVR_Input_Sources.LeftHand].stateUp;
43          m_touchpadPressUpRight = SteamVR_Actions.default_TouchpadPress[SteamVR_Input_Sources.RightHand].stateUp;
44
45          m_touchpadPositionLeft = SteamVR_Actions.default_TouchpadPosition[SteamVR_Input_Sources.LeftHand].axis;
46          m_touchpadPositionRight = SteamVR_Actions.default_TouchpadPosition[SteamVR_Input_Sources.RightHand].axis;
47
48          m_triggerPositionLeft = SteamVR_Actions.default_TriggerPosition[SteamVR_Input_Sources.LeftHand].axis;
49          m_triggerPositionRight = SteamVR_Actions.default_TriggerPosition[SteamVR_Input_Sources.RightHand].axis;
50      }
51      #endregion
52
53      #region Private Functions
54      void Log(string _msg)
55      {
56          if (m_Debug)
57          {
58              Debug.Log("[InputManager][" + _msg + "]");
59          }
60      }
61      #endregion
62  }
```

Figure 10 Radial Menu

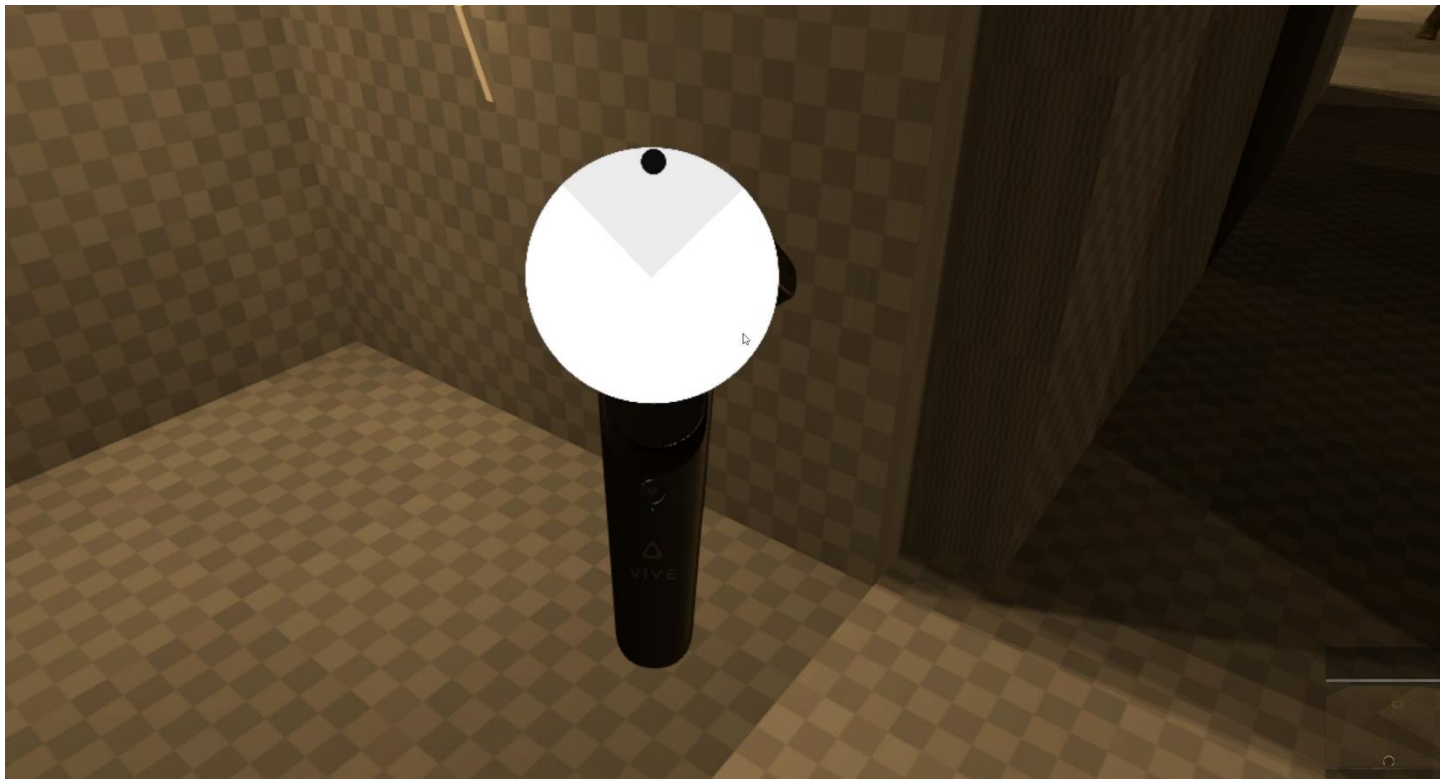


Figure 11 Main Menu

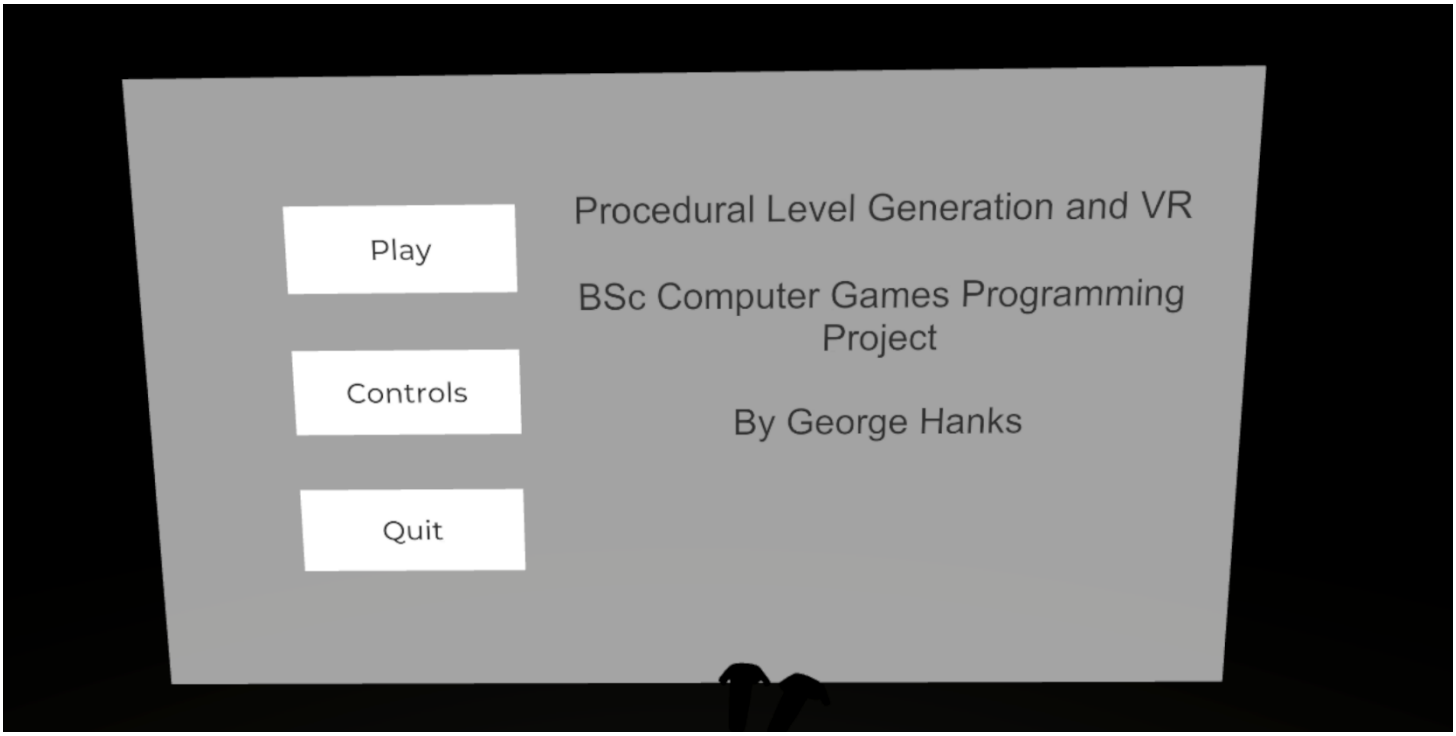


Figure 12 Teleport Spell

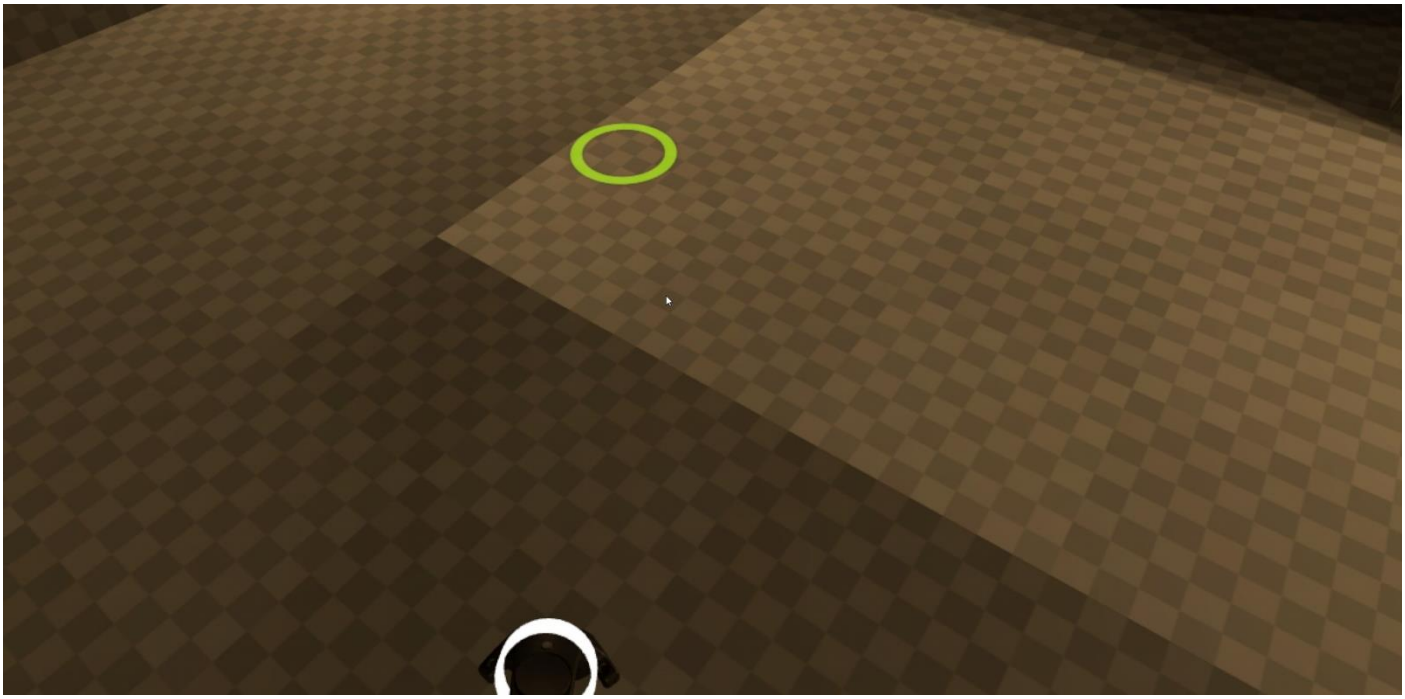


Figure 13 Player Health

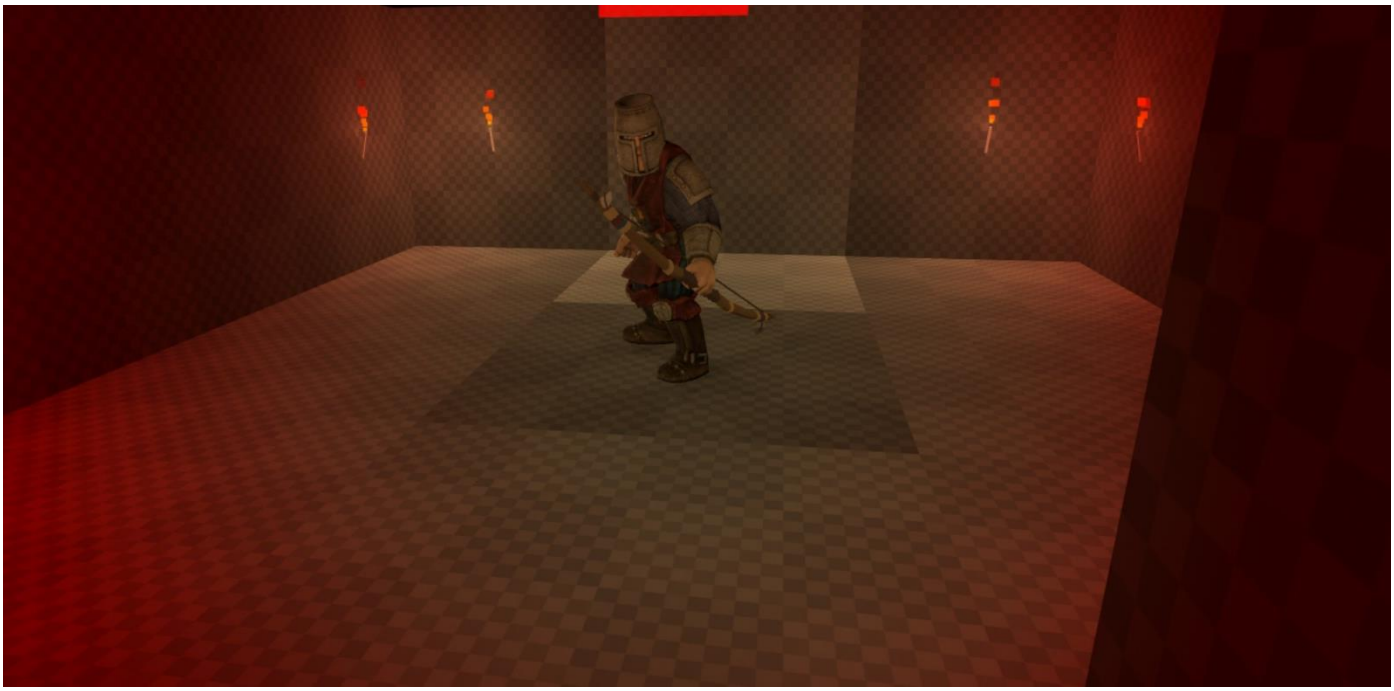


Table 2 Black Box Testing Table

HTC Vive Input Test Cases

Test Name	Expected Outcome	Last Test Outcome	Additional Comments
Head Rotation	• When the player rotates their head, the camera rotates as well • The rotation is in the correct direction • The rotation is the same distance	20/04/20 – The Camera smoothly followed the rotation of the user's head and did not lose synchronisation.	N/A
Head Position	• When the player moves their head, the camera moves as well • The movement is in the same direction • The movement is the same distance	20/04/20 – When the player moved around the play space the camera smoothly followed the position whilst maintain the same rotation.	N/A
Controller Left / Right	• The Left and Right Controller Should Work independently from one another	20/04/20 – When buttons on the controllers where pressed, they where received independently from each other.	This test failed previously, the input manager script was missing the input source parameter.
Controller Position	• When the player moves the controller, the hands/controllers in the game should move • The movement is in the same direction • The movement is the same distance	20/04/20 – When the user moved the controllers the controllers in the game, moved the same distance, they did not feel out of place for the user.	N/A
Controller Rotation	• When the player rotates the controllers, the hand/ controllers in the game should rotate • The rotation is in the same direction • The rotation is the same distance	20/04/20 – When the player rotated the controllers the controllers in the game matched the rotation.	N/A
Controller Button Input	• When the player presses an input on the controller an event should trigger • A button press should trigger a single event • Each button has its own event	20/04/20 – When the user pressed buttons on the controller they updated the data in the input manager script.	N/A
Controller Rebinding	• The user should be able to rebind inputs using the SteamVR interface	20/04/20 – The user was able to customize the binding of actions on their controllers.	When a release version of the game was built the bindings where not exported, to fix this they need to be exported separately with the SteamVR Binding GUI.

Spell Test Cases

Test Name	Expected Outcome	Last Test Outcome	
Spell Equip	• The unequip function for the old spell is executed • The old spell is unequipped from the correct hand • The new spell is equipped to the correct hand • The equip function runs on the new spell	30/04/20 - When a new spell was equipped the old spell was unequipped and the new spell was bound to the correct hand.	N/A
Spell Unequip	• The unequip function for the spell is executed • The spell is removed from the correct hand	30/04/20 – When a spell was unequipped, the unequip function is called and the spell is removed from the correct hand.	N/A
Spell on press event	• When the player starts pressing the input the on-press function is call a single time	30/04/20 – When the player starts pressing the trigger for a single frame the OnPress event is called.	Failed on previous test, was being called every frame, error with input manager class.
Spell on hold event	• When the player holds the input the on-hold function is called every frame	30/04/20 – When the player holds down the trigger, every frame the OnHold event is called.	N/A

Spell on release event	• When the player stops pressing the input the on-release function is called a single time	30/04/20 - When the player releases the trigger for a single frame the OnRelease event is called.	N/A
Spell Cooldown	• When the player casts a spell, it should begin cooldown • When a spell is in cooldown the player should not be able to cast that spell • When the spell has finished cooldown, the player should be able to cast the spell	30/04/20 – When the player casts a spell the time is recorded so the cooldown time can be calculated. The player was not able to cast the spell while it was cooling down.	N/A
Spell Cooldown Ring	• The Ring should appear on the players controllers only when a spell is equipped in that hand • The ring should be white when the spell is ready to cast • The Spell should transition from red to green when the spell is in cooldown • When the player unequips a spell the ring should disappear	30/04/20 – The cooldown ring only appeared when a spell was equipped and was the correct colours for each stage of the cooldown.	Failed on a previous test, the rings where changing colour in reverse, error with parameters in the function.

Trap Spell Test Cases

Test Name	Expected Outcome	Last Test Outcome	Additional Comments
Trap Target	• When the spell is activated a target should appear where the player is pointing • The target should only appear if there is an object to point at • The target should be green when the player can cast the spell • The target should turn red if the player is pointing at an invalid surface • The target should turn red if the target is out of range from the player	01/05/20 – When the spell as activated and the player was pointing at a valid surface a target appeared and was green. When the target was out of range or pointing at an invalid surface it turned red.	Failed on a previous test, the target was also changing size, which is not a desired behaviour, issue was with texture scaling.

Missile Spell Test Cases

Test Name	Expected Outcome	Last Test Outcome	Additional Comments
Spawn Missile	• The missile prefab is spawned • The missile has spawned at the correct location • The missile has the correct rotation • The missile has the correct properties	01/05/20 – When the spell is activated the prefab was spawned with the correct orientation, it also had the correct properties.	N/A

Projectile Test Cases

Test Name	Expected Outcome	Last Test Outcome	Additional Comments
On Update	• The projectile should move forward the correct amount • The projectile should be facing the correct direction • If the missile has gone past its range, it should destroy itself	01/05/20 – Every frame the projectile would move forward with the correct delta and was also facing the correct direction. When the spell had gone past its range it was destroyed.	N/A

On Collision Enter	• If the missile has hit a collider that has health properties it should deal damage to that collider • If the missile has one, an explosion particle should be spawned • If the missile has one, a sound should be played	01/05/20 – When the Projectile collided with a valid surface, it spawned its particle and played the sound. If the collider it hit was a player or an NPC it deducted health.	N/A

Radial Menu Test Cases

Test Name	Expected Outcome	Last Test Outcome	Additional Comments
Visibility	• When the player touches the touchpad, the menu should appear • When the player releases the touchpad, the menu should disappear	02/05/20 – When user touched the touchpad the radial menu appeared and stayed attached to the controller, when the user released there thumb the radial menu disappeared.	N/A
Segment Selection	• When the moves their thump around the touchpad, the correct segment should be activated • Only one segment at a time should be activated	02/05/20 – When the users moved there thumb the correct segment would be activated, and only one segment was active at one time.	N/A
Activate Segment	• When the player presses down on the touchpad, it should activate the event bound to the segment • The activate should only be called once per press • A sound should play to indicate the event being invoked	02/05/20 – When the user pressed down on the touchpad, the event attached to the segment was invoked, and the event was called only once, the correct sound was also played when the user pressed down.	Failed on a previous test because no sound was played when the user pressed down on the touchpad, this was due to a missing reference.

Enemy Test Cases

Test Name	Expected Outcome	Last Test Outcome	
Collison	• The collider for the enemy should collide with other objects in the world • The collider should be in the correct position	25/04/20 - When the software was run the enemy would fall and hit the ground, and the collider was in the correct position	N/A
Character Model	• The model for the enemy should be visible • The model should be in the correct position • The model should be facing the right direction	25/04/20 – When the software was run the character model was visible and animated, it was in the correct position and it was facing the correct direction.	Failed on a previous test as the character model was facing the wrong direction, the offset in the rotation was incorrect so was changed to fix the issue
Movement	• The object for the enemy should move • The movement should be in the correct direction • The movement should be correct speed	25/04/20 – When the software was run and the enemy had a target it would calculate a path to the target and move along that path, it moved along the path at a constant speed and face the direction of movement.	N/A
Turning	• The object for the enemy should rotate • The rotation should be in the correct direction • The rotation should be the correct speed	25/04/20 – When the program was run, and the enemy had a path and not facing the correct direction it would turn to face the current direction. It would turn at a constant speed turned the shortest way to the correct direction.	N/A
Target Acquisition	• If a potential target is within the range of the enemy, and the enemy has line of sight it should make it its target.	25/04/20 – When the program was run the enemy would start looking for targets, when one was within range the enemy would start checking for a line of sight	N/A

		between the target and the enemy, if there was a line of sight, the enemy would set the target as its target.	
Attack	• If the enemy has a target it should try to attack • The enemy can only attack if the target is within range • The enemy can only attack if it has been long enough since its last attack • If the target is out of range the enemy should try to reduce the distance	25/04/20 – When the enemy had a target it would check if it was in its attack range, if it was it would attack the target, it would then record the time of this attack and wait unit enough time had passed for another attack to occur. If the enemy was not within range, it moved to reduce the distance.	N/A
Health is Zero	• If the enemy's health is zero it should run the onDeath function • The enemy should no longer be able to attack • The enemy should no longer be able to move • After the on-death function the enemy should not be visible • The enemy should no longer exist as an object in the game	25/04/20 – After the enemy had been attacked it would check how much health it had and if had not health it would run the onDeath function and was unable to attack the player.	N/A
Navigation	• When the enemy has a target and is moving it should create a path using the nav mesh agent and the nav mesh	25/04/20 – When the enemy was active, it would communicate with the nav mesh component to calculate a path to the target.	

Dungeon Generation Test Cases

Test Name	Expected Outcome	Last Test Outcome	
Dungeon Debug	• A wireframe square should be draw at each occupied tile/ cell of the dungeon. • The size of the square represents the real size of a tile/ cell • The colour of the square should be different for each cell type	26/04/20 – When the dungeon was debugged a square was draw at each cell of the dungeon, all the cells were in the correct place and were the correct colour for the type of cell.	N/A
Adding a Room	• The room should be added to the dungeon data • The information is transferred correctly • The room cannot overlap another room or corridor • The room should be aligned to corridor/ hallway it was added from • The entrance door is marked as an entrance	26/04/20 – When a room was added to the dungeon, all the correct data was stored inside the dungeon data, the overlap checking prevent rooms from being inside one another, the entrance doo for the room was marked as connected and the room was aligned to the grid.	N/A
Adding a Corridor/ Hallway	• The Corridor should start from the door of an existing room • The length of the corridor should be within the limits set by the dungeon settings • The corridor cannot overlap any other rooms or corridors • The Information about the corridor is added to the dungeon data.	26/04/20 – When a corridor was added to the dungeon, it would start from an existing door and attach to another door, it would be the correct length to connect the two rooms and it did not overlap either of the rooms. All the information about the corridor was added to the data about the dungeon.	N/A
Spawning Room Prefabs	• The 3D models for each room is added to the level • The correct set of models are added • The models are alleged to the tile/ cells of the dungeon	26/04/20 – When the 3D models for the dungeon are spawned into the scene, the correct models would be spawned for each room, there were no duplicates, and each model had the correct orientation and position	N/A

	• The models are in the correct position • The models are facing the correct direction		
Spawning Corridor/ Hallway Prefabs	• The model for the corridor is added to the level • The correct number of corridor section are added to match the length • The sections are aligned to the tiles/cells of the dungeon • The sections are in the correct position • The sections are facing the correct direction.	26/04/20 – When adding the models for the corridor, the correct number of segments were added, and they were all facing the correct direction.	N/A
Spawning Door Prefabs	• The model for the door is added to the level • The model is in the correct position • The model has the correct rotation • The correct model for the door is added	26/04/20 – When spawning the prefabs for the doors, the correct prefabs were spawned, and the position and orientation were correct.	N/A

Main Menu Test Case

Test Name	Expected Outcome	Last Test Outcome	Additional Comments
Laser Pointer	• When the player holds the trigger on the right controller a beam should appear and collide with the world and the user interface • When the player lets go of the of the trigger the beam disappears.	04/05/20 – When user pressed down the trigger on the right-hand controller, a laser beam appeared, this beam would end where the player was pointing, when the player let go the beam disappeared.	N/A
Button	• When the laser beam is over a button the button should change colour • When the laser is over a button and the trigger is clicked the button should activate and invoke its event	04/05/20 – When the user pointed at a button it would change colour, and when the user clicked the trigger on the right-hand controller it would activate the button and invoke the event attached to the button.	N/A