Uses Of Artificial Intelligence In Video Games

Kane Thompson-Hall
33385792

Goldsmiths
University of London

BSc. Computer Science

May 15, 2017

Abstract

This dissertation reports based on a Unity developed game that will serve as a tech demo that explores and presents the use of AI techniques in video games. In particular I will showcase a use of Perlin noise in procedural generation and intelligent agents.

Uses Of Artificial Intelligence In Video Games

Kane Thompson-Hall
33385792

Goldsmiths

University of London

BSc. Computer Science

May 15, 2017

Contents

List of Figures

List of Tables

List of Abbreviations

IA . . . . . Intelligent Agents.

AI . . . . . Artificial Intelligence.

FSM . . . . . Finite-State Machines.

PCG . . . . . Procedural Content Generation.

VR . . . . . Virtual Reality.

NPC . . . . . Non-Player Character.

RTS . . . . . Real Time Strategy.

IDE . . . . . Integrated Development Environment.

2D . . . . . Two-Dimensional Space.

3D . . . . . Three-Dimensional Space.

1 Introduction

Contents

1.1 Overview of the System

This project will consist of a Unity developed game in which players will take control of a character, and using the directional keys will navigate a procedurally generated level. The objective of the game is to reach the goal before the IA (Intelligent Agent) catches the player. Each level will be procedurally generated, meaning it will be created algorithmically, so that no two user experiences will be the same. Each level will contain IA that will use various path finding algorithms and attempt to stop the player from reaching the exit. Each IA will also feature FSM (Finite-State Machines). This will enable it to appear intelligent and perceive and act on it's environment like an IA should.

1.2 Motivation

During my final year, I studied four modules. Two of these modules AI (Artificial Intelligence) and Game AI Programming had a big influence on my decision to build this project. In AI one of the topics I studied was about IA where 'An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.'[1] I also studied several topics in Games AI Programming which I was really interested in, such as state machines and procedural content generation. Unfortunately, due to the scope of both courses I never had an opportunity to implement everything I was interested in to a substantial project. In addition to my already established interest in building game projects. This computing project now gives me that opportunity to be able to combine and utilize the overlap between the courses and combine the knowledge from both, to build a project that features content covered in both modules. Furthermore, the game will also serve as a demonstration for how content can be algorithmically generated, serve as a practical use of noise in procedural generation and demonstrate various path finding algorithms. This motivates me as the project could prove to be a good learning resource demonstrating several algorithms.

1.3 Report Structure

This report is structured as follows: In Chapter 2 a detailed background research in the area of *Practical Uses of specific AI techniques in Game development* is given. In Chapter 3 I discuss the system requirements and the criteria that will be needed to consider the game fully functional. In Chapter 4 the system design part is described. In Chapter 5 the implementation part of the system is given. Chapter 6 consists of the description of the testing techniques and the testing results. Chapter 7 will consist of an evaluation of the implemented software and my findings. Finally, a conclusion and a direction of future work is given in Chapter 8.

2
Literature Review

Contents

2.1 Introduction

The Literature Review section will consist of a detailed discussion surrounding the background research that was conducted in building this project.

2.2 Research

2.2.1 Intelligent Agents

As stated in my motivation an IA is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through

effectors. IA have had a long history in video games dating back to the first video games ever released.

> "In 1970, Nutting Associates released the first video arcade game, "Computer Space". And it was not until then that game designers began their first attempt to incorporate AI into their games. AIs were designed primarily for arcade games with the purpose of ensuring people kept feeding quarters into the game machine. (Tozour) Pong, Space Invaders and Donkey Kong were among the seminal games. These games were running under very simple rules and scripted actions. The agents didn't have the ability to make decisions. Sometimes decisions were designed to be made randomly so that the behaviours looked more unpredictable. Therefore, the so called intelligence was actually coded into the game and was not able to act at runtime."

According to **(Xu,n.d)**[**2**] Since IA first introduction in video games they have expanded and become increasingly more popular and complex. In modern video games such as shooting games NPCs (Non-Player Character) are able to assess a players position and take cover or shoot back. Moreover in RTS and other genres IA are now able to make complex decisions in order to overcome human opponents.

According to the book **Artificial Intelligence: A Modern Approach** Russell and Norvig consider there to be five classes of agents. The five classes consist of simple reflex agents, model- based reflex agents, goal-based agents, utility-based agents and learning agents. Each agent receive percepts from the environment and perform actions. A percept is the input that an IA is currently perceiving.

> "Simple-reflex agents act on the basis of the current percept, they work by finding a rule whose condition matches the current situation defined by the current percept and then the agent will do the action associated with that rule."
>
> "Model-based reflex agents expand on simple-reflex agents, with an internal state that depend on the percept history and actions are done by those defined in the percept and the stored internal state."

[1]

Goal-based agents further expand on the idea of model-based reflex agents with "goal" information. Goal information dictates which situations are desirable. This means goal-based agents are able to choose among multiple desirable possibilities

and choosing the one that reaches a goal state. Goal-based agents are more flexible than reflex agents with respect to reaching different destinations. Simply by specifying a new destination, we can get the goal-based agent to come up with a new behaviour. The reflex agent's rules for when to turn and when to go straight will only work for a single destination.

Utility-based agents follow the same principal as goal-based agents but goal-based agents are only able to distinguish between goal states and non-goal states. Utility-based agents therefore utilizes function that maps a state to a measure of the utility of the state where the 'utility' is a quality measure of the agent.

Learning agents can be divided into four conceptual components. The learning element is responsible for improvements this can make a change to any of the knowledge components in the agent. The performance element is responsible for selecting external actions. The critic is designed to tell the learning element how well the agent is doing. The critic employs a fixed standard of performance. The final component the problem generator, is responsible for suggesting actions that will lead to new and informative experiences. As described by myself and according to the book **Russel + Norvig,1995** [**1**]

2.2.2 Finite-State Machines

FSM is a device that I am going to implement into the IA that is featured in this project. "A FSM is a rigidly formalized device used by mathematicians to solve problems". One use of a FSM is a hypothetical device that Alan Turing a Mathematician created in 1936. The machine featured 3 states and it foresaw modern day computers which could perform logical operations such as reading, writing and erasing symbols on a strip of tape. **(Coppin,2004)**[**3**] An example of FSM is flicking a light switch off and on. It transitions between each state off and on by the input of flicking a light switch. When the light switch has the 'on' state it allows electricity to run, and power the light bulb when it's off no action occurs. Implementing a FSM in the IA in this project will enable it to act on and perceive its environment. An example on how FSM are used in video games is

in the game Pac-Man in which the ghosts utilize states. The first main state is the chase state in which the implementation is slightly different for each ghost on the screen. The next main state occurs when the player picks up a power pill in this state all the ghosts will transition to the evade state. **(Buckland,2005)**[**4**] I am going to implement something similar for this project in which, in each IA the chase state will consist of a path finding algorithm such as A*, and that state will be changed to an Evade 'like' state by the input of the player's actions attacking the IA or the type of tile it is surrounded by. Like Pac-Man, the player in my game will also have multiple states which will be affected by the IA attacking the player or the player being affected by the surrounding tiles. Another example of states being used in games is the NPCs (non-player characters) in RTSs (real-time strategy games) such as Warcraft make use of finite state machines. They have states such as MoveToPosition, Patrol, and Follow Path. Optionally I could also implement something like this for implementing an FSM for the player controlled character. States like those in the example will allow me to navigate by use of the mouse if I decide to implement another movement input. Each state will change by an input of a mouse click and will move the player around the level

2.2.3 Pathfinding Algorithms

An AI technique commonly used in AI is path finding algorithms which essentially finds a path from a starting node to a goal node. My game will include procedurally generating navigation graph which will be used as the search space utilizing the path finding algorithms in this project. Each intelligent agent featured in my game will feature a path finding algorithm with a goal.

A combination of traditional uniform path finding algorithms and breadcrumb path finding is a feature that will be present in my game. The concept of breadcrumb pathfinding is the act of a user unknowingly creating a path for other computer-controlled characters. When a user moves they will create a string of breadcrumbs that can act as a path for other controlled characters thus following the user **(Bourg + Seeman,2004)**[**5**]. Breadcrumb pathfinding technique will be implemented when

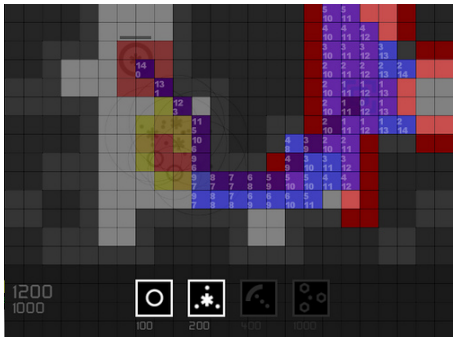

Figure 2.1: You can see all the paths A* explored in blue. The optimal path to the target is in purple. Each cell evaluated has two numbers the first is the cost to get to that tile and the second is the weight of that tile.**(Schearer, 2009)**[8]

the IA has reached a location close to the goal the player and it is within sight of the player and thus will follow it's trail.

A common pathfinding algorithm for IA used in video games is the A* algorithm this will be a pathfinding algorithm that I focus on in my game. The A* algorithm is an improvement of the Dijkstra's 1959 algorithm. The A* algorithm was first described in 1968 by researcher Nils Nilsson who was trying to improve the path finding of shakey the robot. The A* algorithm improves Dijkstra's algorithm by the addition of heuristics, the A* algorithm iteratively checks all the best unexplored locations and if an explored location is the goal the algorithm is complete, else it keeps going. The reason I chose to focus on this algorithm is because A* is the most popular choice for path finding, because it's fairly flexible and can be used in a wide range of contexts.**(Patel,2017)**[6]

In terms of it's use in the video game development as the game industry continues to gain momentum, path finding is becoming quite popular and also a frustrating problem in the industry. It is commonly used in RTS (real-time strategy) games in finding the most efficient path on different terrain **(Cui + Sh, 2011**[7]. Due to most commercial games source code being private it was difficult to locate an example of pathfinding algorithms being used in games. However, 'Null OP' an academically / independently developed tower defence based game gives an example in fig 2.1 as to how it is used.

Figure 2.2: Pathfinding used in the game Civilization 5**(Sid Meier's Civilization V,n.d)**[10]

Reinforcing what was said earlier about RTS games [2.2.3] according to the book **(Artificial Intelligence For Games,2009)** "With Dune II [Westwood Studios, 1992], Westwood created a new genre", "the genre is one of the strongest on the PC platform." "Key AI requirements for real-time strategy games are: 1. path finding and 2. Group movement."[**9**] It is clear that path finding algorithm are crucial in modern game development.

2.2.4 Perlin Noise

First examples of noise are on old TV sets or Radios if you were tuned to a channel that that didn't have a station. In digital terms, noise most of the time can be a nuisance. In terms of audio noise, noise is unpleasant sound that is added to a desired sound. Audio noise is represented by an assortment of random numbers in 1D.In terms of images it is presented as unwanted brightness etc and is represented by random numbers arranged in 2D. Noise can also be present in higher dimensions.**(Red Blob Games, 2013)**[**12**]. In most circumstances it is preferred to subtract noise from a situation, however in a lot of applications of noise the natural situations tend to be noisy so if you want to procedurally generate something it would be a good idea to add noise, for it to appear more natural as it adds variation. Perlin noise is a type of gradient noise that was developed by Ken Perlin in 1983. Perlin noise is the foundation of many procedural texture and modelling algorithms. It can be used to create marble, wood, clouds, fire, and height maps for terrain. It is also very useful for tiling grids to simulate organic regions, and blending textures for interesting transitions **(Tulleken, 2008)**[**11**]. Perlin Noise is

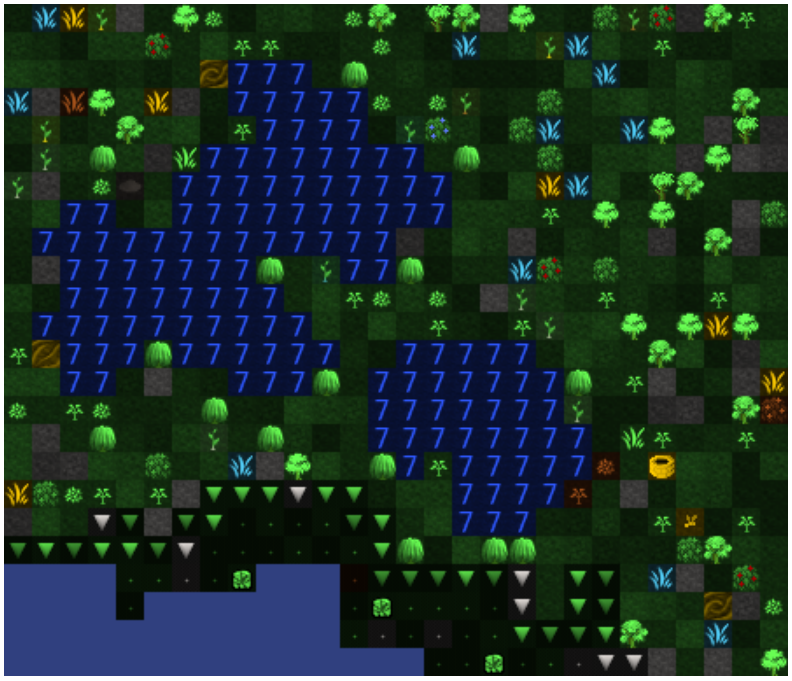

Figure 2.3: Image from a mod that loads tree sprites on top of Dwarf Fortress gameplay terrain.[13]

also frequently used in PCG (Procedural Content Generation), It can be utilized by using the noise values and comparing them to a threshold. For example you can create an if clause and the condition could be if the noise value returns a value greater than 0 then do one thing, else if the value is smaller than 0 then do another. An example of this is using 3D Perlin noise to generate caves. You can say that if a noise value is beyond a certain value then build solid earth else it is open air.[12]

In terms of game development, a mod designed for a game called Dwarf fortress (2006) is an example of noise being used in PCG to generate natural environments. The mod generates a bitmap image with a grey scale coloured pixel corresponding to each floating number on the 2D array, makes the array the same size as the screen and coloured each pixel in a grey scale going from 0 to 1 **(Mazzini, 2016)[13]** In order to create an environment like this that generates nature sprites on the map. In Ken Perlin's paper on 'Improved noise' it is stated that

> 'Noise is determined at point **(x,y,z)** by computing a pseudo-random gradient at each of the eight nearest vertices on the integer cubic lattice and then doing splined interpolation. Let **(i,j,k)**denote the eight points on this cube, where **i** is the set of lower and upper bounding integers

on **x: | x|,| x|+1**, and similarly **j = | y|,| y|+1** and **k = | z|,| z|+1**.The eight gradients are given by $g_{i,j,k} = G[P[P[P[i]+j]+k]]$ where precomputed arrays **P** and **G** contain, respectively, a pseudo-random permutation, and pseudo-random unit-length gradient vectors. The successive application of **P** hashes each lattice point to de-correlate the indices into **G**. The eight linear functions $g_{i,j,k}$ **sum((x-i,y-j,z-k))** are then trilinearly interpolated by **s(x-|x|),s(y-| y|)**and **s(z- z|)**, where

$$s(t) = 3t^2 - 2t^3 \tag{2.1}$$

(Perlin, 2002)[**14**].

2.2.5 Navigation Graphs

When it comes to handling the navigation in game development navigation graphs or navigation meshes are commonly used. In a navigation graph, the graph is based on nodes which are not necessarily meant to cover every navigable location in a search area,but rather serve as navigable way-points.A set of 2D primitives (circles, quads and the like) can be used to represent nodes in a navigation graph.**(Borovikov, 2011)**[**15**]In addition a path can be considered from anywhere in the environment to any one node in the navigation graph. For example an agent located anywhere in an environment is able to create a path a navigate to any node in the navigation graph under the condition that no constraints are applied such as obstacles obstructing the path.

In comparison to navigation meshes navigation graphs offer various advantages and disadvantages. Navigation graphs are useful when it comes to placing objects into the environment. For example if we placed a door into an environment where the user needs to be in a certain orientation to open it. Navigation graphs will allow us to position a door such that there will be one node in which the door will allow us to pass through.[15]It is also not restricted in flexibility to the level design like meshes hence it can offer better level design as levels do not need to fitted to suit the graph. However navigation graphs can also be time consuming to implement if the graph contains a lot of nodes, it will also be difficult to alter the graph if the geometry of the environment changes. This solution could be solved if the geometry of the environments remain the same and if the objects that will represent the nodes

are procedurally generated some of these disadvantages can be eliminated as the navigation graph will naturally build itself and adjust to the environment.

2.3 Existing Market Research

2.3.1 Commercial Market: Pac-Man

Pac-Man is a iconic video game which released in 1980 and is one of the first commercial examples that employs similar features and ideas that I am planning on implementing in this project. The goal of Pac-Man is to eat all the dots positioned on each level while avoiding ghosts. If Pac-Man is captured by a ghost, the ghosts will return to their base and a Pac-Man will start a new life. In addition, the ghosts have 3 states which can be described below.

Chase - In this state the ghost will chase Pac-Man following him wherever he travels in the maze. In addition each ghost performs a unique behaviour depending on it's colour. The red ghost is aggressive and will follow closely behind Pac-Man. The pink ghost tries to trap Pac-Man by blocking his path. The light blue ghost behaviour is unpredictable and the orange ghost tends to his own thing.

Scatter - In this state the ghosts will traverse to a corner of the grid for a few moments before returning to chase Pac-Man.

Frightened - In this state all the ghosts colour will change to dark blue to represent they can be destroyed, and then will randomly move around the grid. After a few seconds the ghosts will flash white before transitioning to one of the other states.**(Pittman, 2009)**[**16**]

This state system for the ghosts is very similar for what I plan to implement in this project using the FSM device. In my system the IA will feature several states that will affect how they interact with their surroundings similarly to how the ghosts in Pac-Man work. In addition to this Pac-Man the player himself also has states, states being a normal state where he traverses the level eating dots and a dead state when the player is eventually eaten by a a ghost, this will also be implemented in the project.

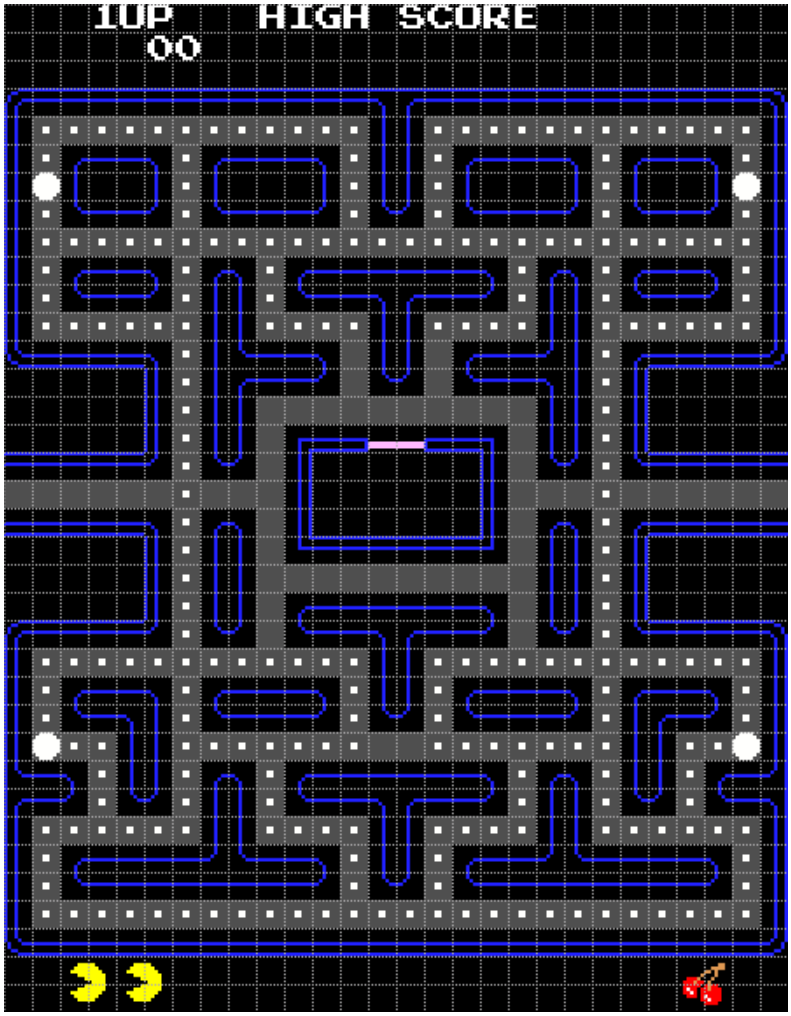

Figure 2.4: A visual representation how the grid in Pac-Man functions.[16]

In addition Pac-Man also utilizes a grid like structure something that I plan to use in the project. The gameplay takes place in a grid of tiles in which each tile is 8 pixel squared. The entire grid of the game consists of a 28 x 36 and each tile contains a dot which occupy the center of the tile which means each adjacent dot has a distance of 8 pixels from each other.**(Pittman, 2009)[17]**

2.3.2 Academic:Endless Web

In terms of academically produced projects Gillian Smith, Alexei Othenin-Girard, Jim Whitehead, Noah Wardrip-Fruin at the University of California, Santa Cruz have developed a PCG based game another ‘ Endless Web, a 2D platforming game in which the player’s actions determine the ongoing creation of the world she is

exploring. ‘ ’The goal of the game is for the player to rescue six dreamers who are trapped in their nightmares’[**18**]. This game uses a PCG system called launch pad which is a rhythm-based level generator for 2D platforming games, I plan to adopt a psuedorandom generator for the PCG that will be produced in the game. Endless Web also implemented ’Power ups’ which ’eases level exploration this could also be implemented a simmilar way to how I plan to utilize FSM in order to alter states of the NPC and player on screen. In addition the random generation also generates normal platforms and hazardous platforms that can potentially kill the player, this is something that will be implemented in the game, in a sense that their will be hazardous tiles that will be generated that can alter the state of the player to dead.

One of the problems they presented was during the PCG portals which were used for level goals where generated some players complained they were positioned in awkward places on the level. Therefore, the solution to this was to intelligently place them probabilistically on the map using a probability table. This is something I may need to adopt for this project during the evaluation phase if feedback states the levels are generated awkwardly.

3

System Requirements

Contents

3.1 Introduction

In this section I am going to discuss the criteria surrounding the functionality and usability objectives of this project in order to consider it a success. In addition I will also list the criteria in which I will consider the project complete and extension goals I intend to implement.

3.2 Functionality Requirements

	Requirements
i	Users can control and navigate a playable character in a first person perspective.
ii	System is able to procedurally generate unique levels.
iii	The enemies have various states they can transition between.
iv	Enemy IA can chase the player and if caught player will be destroyed
v	The player is able to pickup power-ups which can dictate the state of the enemy IA

Table 3.1: List of my functionality requirements of the system.

3.2.1 Basic Level: [i,ii,iii]

If the functional requirements corresponding to [i,ii,iii] are to be accomplished by this system then I can consider the system to be functional at the most basic level for it to succeed. Since the game will enable the player to control the character in a procedurally generated area and complete the objective avoiding the IA

3.2.2 Complete: [i,ii,iii,iv,v]

If all the functional requirements are met [i,ii,iii,iv,v] then I can fully consider the game to be a success. Since the addition of the iv requirement enables that the IA in the game to be able to chase the target which will be the player similar to what was detailed in Pac-Man. In regards to the v functionality, I am going to need to implement a FSM that will change states depending on if the player picked up a power-up. This will help determine whether the IA should be in a fleeing state in which it will avoid the player.

3.2.3 Extension Goals

As I listed above if this project is able to meet the functionality criteria list then I will consider it complete. However I have an extension goal of implementing VR compatibility into the project which I plan to implement if I have there are no unforeseen time constraints.

3.3 Usability Requirements

	Requirements
i	Users are able to input their own frequency and threshold values to generate specific level they want.
ii	Users are able to input the number of IA they want to spawn in on the level.
iii	Users are able to opt in or out of VR computability if the extension goal is achieved

Table 3.2: List of the usability requirements of the system.

4 Design

Contents

4.1 Introduction

In this section I am going to discuss how I am going to apply the context of the research from the literature review to some of the core components of the final project.

4.2 Design/Methodology

4.2.1 Existing Development / Third-party Implementation

In terms of existing development and third-party implementation in this project, I will be using implementation from a texture creation tutorial.In particular the Noise class, which is based on Ken Perlin's Improved gradient noise algorithm [**19**].

Next I will be heavily modifying a maze tutorial implementation [**20**], in which I will use the implementation from the GameManager and coordinates cs script and the technique used to generate tiles in intervals using a coroutine.

I will also be using some of my own existing implementation from a previous Game AI project, I will use the full implementation from the PathAgent, Pathfinder, Graph, AdjacencyListGraph, AstarIA, RandomIA cs scripts. Furthermore I will be using the WayPointGraph cs script and slightly modifying it changing the initial code to ensure only nodes with a approximate cost of 1.5 are connected by an edge, and a list of every tile that was procedurally generated are added to a navigation graph as waypoints.

4.2.2 Development

To develop this project I will be using the Unity3D engine and Microsofts Visual Studio IDE in order to implement the game. In addition I will be using the C Sharp programming language in conjunction with Visual studio to write the scripts for the project. I decided to use the Unity3D engine because I already had little experience with it, it is easy to use and it will enable me to produce the project I want. The Unity3D engine will enable me to render simple objects that will represent the player and the IA in this project, and cameras to allow the game to run in the correct perspective. I also plan on using free to use low poly models from the Unity asset store for the objects and tiles to improve the aesthetics of the project.

The Unity engine also features VR support which will allow me to easily implement one of my extension goals of VR capabilities for the game.

4.2.3 Intelligent Agents

Using Unity I am going to create individual prefabs for each NPC that will feature in this project. These prefabs will represent IAs in the game and will also feature an Enemy script component which will control the current state and behaviour of the IA. The FSM will consist of a switch statement that dictates the state of the enemy. The first state I will mention 'Chase' will consist of a path finding

algorithm such as A* that will be used to enable the agent to chase the player following the nodes the player has passed in an attempt to catch the player causing the 'Game Over' scenario, under the condition that the agent sees the player or is in close distance to the player. The next state will be 'Flee' which will be used to avoid the player once the player has acquired a Power-Up similar to how the ghosts flee from Pac-Man. The final important state is the 'Roam' state in which the IA will aimlessly roam the level until they see or come close to the player. There will also be a 'Dead' state which will lead to the game object of the IA being destroyed if caught while in it's flee state.

In addition the player will also have three states, in which the first two 'Alive' and 'Dead' dictating if the player should be spawned in or not. 'Powered-Up' which will be a state that the player will remain in for x amount of time. The systems and functionality of the IA and player present will be similar to how I described they act in Pac-Man, which can be found in the existing market research in Chapter 2.

In addition I also plan to implement separate behaviours mechanisms for each IA on screen similar to Pac-Man, the IA will only have subtle differences and will be handled using a behaviour tree mechanism.

Below is a state transition diagram illustrating the structure of the FSM for the IA in game, and the player that was described above.

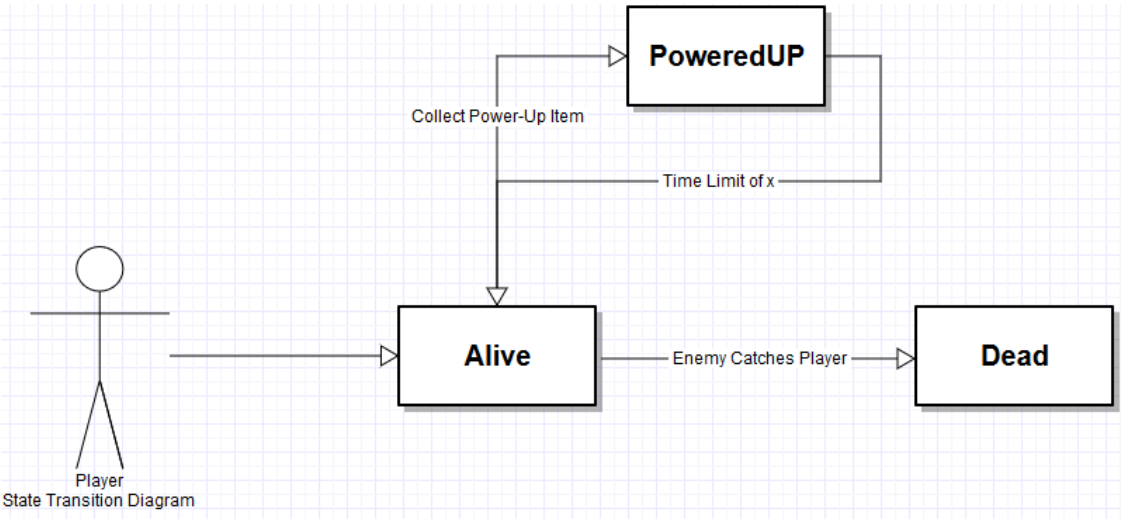

Figure 4.1: A state transition diagram representing the states the player can have.

(4.1)

The above diagram describes the structure of the player FSM. The main state being the 'Alive' state which transitions to the 'PoweredUp' state if the player collects the power-up item, which will then transition back to the alive state after x amount of time. The 'Alive' state can also transition to the 'Dead' state if the enemy catches the player.

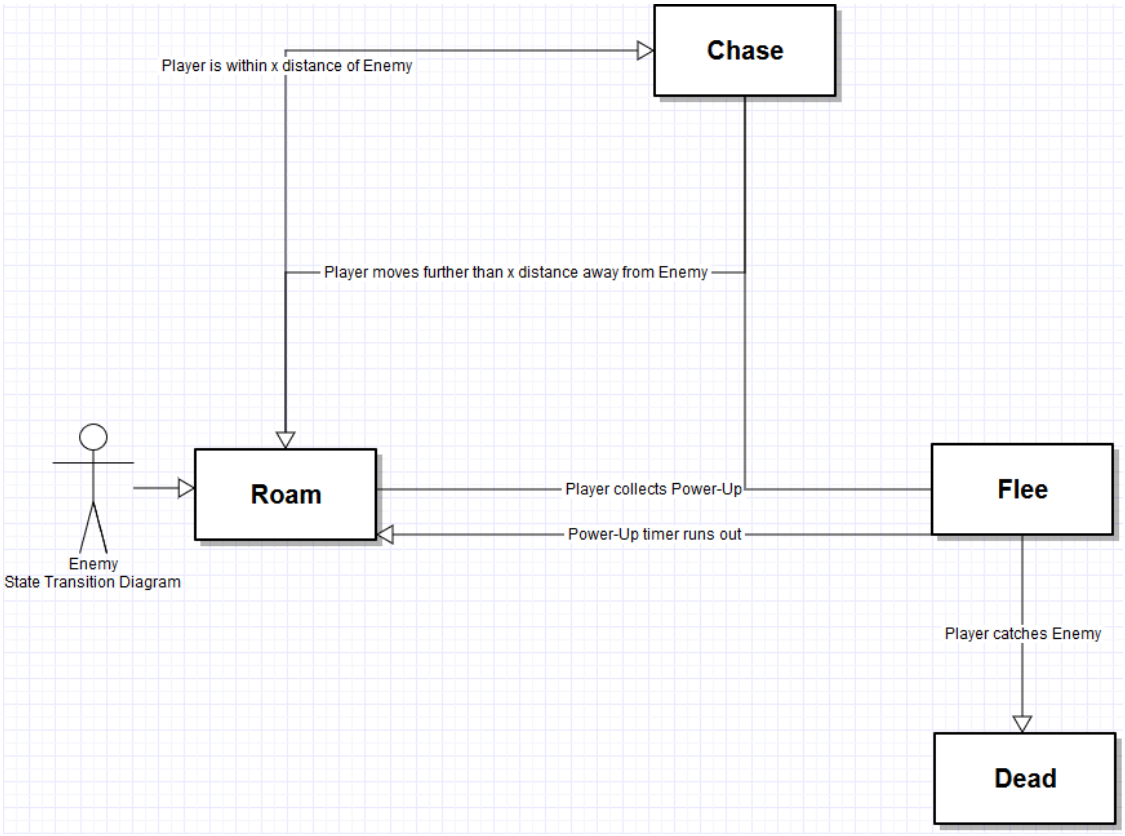

Figure 4.2: A state transition diagram representing the states the enemy can have.

(4.2)

The above diagram describes the structure of the enemy FSM. The initial state the enemy will be in will be the 'Roam' state in which the enemy IA will aimless roam the level using a random path finding algorithm component. If the player is close the the enemy then the enemy will transition to the 'Chase' state. In the 'Chase' state the enemy will chase the player using the A star path finding algorithm, if the player moves a further x distance from the enemy then it will transition back to the 'Roam' state, if the player is in the 'Chase' or 'Roam' state and the player collects a power-up then the enemy will transition to the 'Flee' state. The 'Flee' state involves the agent travelling to the furthest possible location away from the player and will transition back to 'Roam' if the power up timer expires. If the player

catches the enemy while enemy is in the 'Flee' state then the enemy will transition to the 'Dead' state. In which the the enemy game object will be destroyed.

As I learned in Chapter 2, according to Ruseel + Norvig[1] consider there are five classes of agents. The type of agent I plan to implement into this project will be the goal-based agent, the procedurally generated world will be the search space. The initial state will be when the agent is spawned in the world and it will be initialised in the 'Roam' state. The goal state will be the node/ waypoint the player is currently occupying when the agent is transitioning between the 'Roam' and 'Chase' states. Below is a diagram of the structure of a goal based agent.

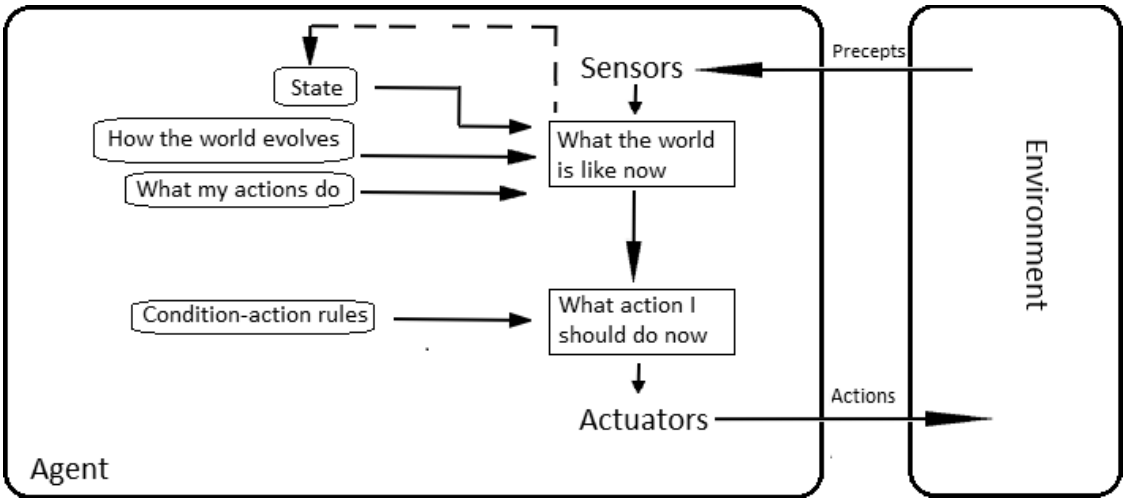

Figure 4.3: The structure of a goal-based agent.

(4.3)

4.2.4 Procedural Content Generation

With regards to the PCG (Procedural Content Generation) aspect of this project the area that will be generated will consist of a user determined X * Z grid that will be algorithmically generated with a tile prefab. In addition this grid will also act as a navigation graph with each tile acting as a node, and each node acting as a waypoint to be moved to by the IA and the player. The world will also be

populated by obstacles, these obstacles will consist of various prefabs, and the prefab to instantiate will be randomly selected and their location will be determined using noise. As stated in the research section in Chapter 2, under Perlin noise it is stated that an example of a use in PCG is using noise values as threshold function. Furthermore it also mentioned 'using 3D Perlin noise to generate caves. You can say that anything above a certain density threshold is solid earth and anything below that threshold is open air (cave).'[12] Similarly in this project the noise values will be calculated using the input vector of the location of every tile in the generated level, in conjunction with a user defined frequency In which the output noise value will be used and compared to the threshold to decide if an obstacle should be placed on that area of the level. This will create a psuedorandom solution in generating unique levels dependent on a frequency and threshold value.

4.2.5 Navigation Graph

In the Navigation Graphs subsection I learned about some of the constraints surrounding implementation of navigation graphs. A constraint that was mentioned is they can be time consuming to implement if the graph contains a lot of nodes or the geometry of the environment changes. In this project that constraint will be solved by assigning a waypoint to the 3D Tile primitives rendered in Unity. These tiles will be procedurally generated and will act as the nodes in our navigation graph. By doing this the game will not need to adjust to a variety of environments.

I decided to work with a navigation graph instead of a navigation mesh because the environments that I am going to be working with are going to be relatively simple and grid based such that I can have a node for every square in grid. However this will also mean I will not be able to have really big environments as it may lead to increased loading times. This is because when assigning waypoints to all the tiles it will then need assign the appropriate adjacent edges in the navigation graph. Furthermore when agents need to calculate their path, having too many nodes will cause increased loading times.

5 Implementation

Contents

5.1 Introduction

In this section I am going to be discussing the implementation component of this project, I will also be highlighting any existing implementation and third-party implementation that I may have used.

5.2 PCG

For the Perlin noise algorithm, like i mentioned in chapter 4 I used the code from the noise tutorial[20]. Which is an implementation based on Ken Perlin's Improved gradient noise algorithm. The tutorial involved trying to create a texture of a chaotic or natural looking surface, I used that implementation and I am applying it

to this project in a way that uses the noise functions input values of a Vector and frequency to get a return value between -1 and 1 below is the algorithm.

```
public class Noise {

    //hash values
    private static int[] hash =
    {
       151,160,137, 91, 90, 15,131, 13,201, 95, 96, 53,194,233, 7,225,
        140, 36,103, 30, 69,142, 8, 99, 37,240, 21, 10, 23,190, 6,148,
        247,120,234, 75, 0, 26,197, 62, 94,252,219,203,117, 35, 11, 32,
         57,177, 33, 88,237,149, 56, 87,174, 20,125,136,171,168, 68,175,
         74,165, 71,134,139, 48, 27,166, 77,146,158,231, 83,111,229,122,
         60,211,133,230,220,105, 92, 41, 55, 46,245, 40,244,102,143, 54,
         65, 25, 63,161, 1,216, 80, 73,209, 76,132,187,208, 89, 18,169,
        200,196,135,130,116,188,159, 86,164,100,109,198,173,186, 3, 64,
         52,217,226,250,124,123, 5,202, 38,147,118,126,255, 82, 85,212,
        207,206, 59,227, 47, 16, 58, 17,182,189, 28, 42,223,183,170,213,
        119,248,152, 2, 44,154,163, 70,221,153,101,155,167, 43,172, 9,
        129, 22, 39,253, 19, 98,108,110, 79,113,224,232,178,185,112,104,
        218,246, 97,228,251, 34,242,193,238,210,144, 12,191,179,162,241,
         81, 51,145,235,249, 14,239,107, 49,192,214, 31,181,199,106,157,
        184, 84,204,176,115,121, 50, 45,127, 4,150,254,138,236,205, 93,
        222,114, 67, 29, 24, 72,243,141,128,195, 78, 66,215, 61,156,180,

        151,160,137, 91, 90, 15,131, 13,201, 95, 96, 53,194,233, 7,225,
        140, 36,103, 30, 69,142, 8, 99, 37,240, 21, 10, 23,190, 6,148,
        247,120,234, 75, 0, 26,197, 62, 94,252,219,203,117, 35, 11, 32,
         57,177, 33, 88,237,149, 56, 87,174, 20,125,136,171,168, 68,175,
         74,165, 71,134,139, 48, 27,166, 77,146,158,231, 83,111,229,122,
         60,211,133,230,220,105, 92, 41, 55, 46,245, 40,244,102,143, 54,
         65, 25, 63,161, 1,216, 80, 73,209, 76,132,187,208, 89, 18,169,
        200,196,135,130,116,188,159, 86,164,100,109,198,173,186, 3, 64,
         52,217,226,250,124,123, 5,202, 38,147,118,126,255, 82, 85,212,
        207,206, 59,227, 47, 16, 58, 17,182,189, 28, 42,223,183,170,213,
        119,248,152, 2, 44,154,163, 70,221,153,101,155,167, 43,172, 9,
        129, 22, 39,253, 19, 98,108,110, 79,113,224,232,178,185,112,104,
        218,246, 97,228,251, 34,242,193,238,210,144, 12,191,179,162,241,
         81, 51,145,235,249, 14,239,107, 49,192,214, 31,181,199,106,157,
        184, 84,204,176,115,121, 50, 45,127, 4,150,254,138,236,205, 93,
        222,114, 67, 29, 24, 72,243,141,128,195, 78, 66,215, 61,156,180

    };

        private const int hashMask = 255;

    //Compute squrare root of 2
    private static float sqr2 = Mathf.Sqrt(2f);
```

```
//2D Gradient coefficients
private static Vector2[] gradients2D = {
    new Vector2( 1f, 0f),
    new Vector2(-1f, 0f),
    new Vector2( 0f, 1f),
    new Vector2( 0f,-1f),
    new Vector2( 1f, 1f).normalized,
    new Vector2(-1f, 1f).normalized,
    new Vector2( 1f,-1f).normalized,
    new Vector2(-1f,-1f).normalized
};

//Dot function to compute and return a value for g(x, y) = ax + by
    where a and b   are either -1, 0 or 1
private static float Dot(Vector3 g, float x, float z)
{
    return g.x * x + g.z * z;
}


private const int gradientsMask2D = 7;

//smooth function to smooth transitions in the interpolated values.
//based on Ken Perlins 2002 revision
private static float Smooth(float t)
{
    //smooth curve function that has a function that has a second and
       first     derrivative of 0 at both ends so the rate of change
       is always 0 at gradient    boundaries.
    //6t5 - 15t4 + 10t3. function
    return t * t * t * (t * (t * 6f - 15f) + 10f);
}

public static float Perlin2D(Vector3 V1, float f)
{
    //Convert the input vector to floating values
    V1 *= f;
    //get the floor value of the floating vector values
    int ix0 = Mathf.FloorToInt(V1.x);
    //(Use Z value instead of x because using a 3D engine and y
       represents the     hight.
    int iy0 = Mathf.FloorToInt(V1.z);
    //Compute gradient values
    //calculate relative distance between the grid point vector and
       floating point   vector
    float tx0 = V1.x - ix0;
    float ty0 = V1.z - iy0;
```

```
        //
        float tx1 = tx0 - 1f;
        float ty1 = ty0 - 1f;
        ix0 &= hashMask;
        iy0 &= hashMask;
        int ix1 = ix0 + 1;
        int iy1 = iy0 + 1;

        int h0 = hash[ix0];
        int h1 = hash[ix1];
        //assign associated random gradient vectors.
        Vector2 g00 = gradients2D[hash[h0 + iy0] & gradientsMask2D];
        Vector2 g10 = gradients2D[hash[h1 + iy0] & gradientsMask2D];
        Vector2 g01 = gradients2D[hash[h0 + iy1] & gradientsMask2D];
        Vector2 g11 = gradients2D[hash[h1 + iy1] & gradientsMask2D];

        //using the dot function where g(x, y) = ax + by where a and b
        float v00 = Dot(g00, tx0, ty0);
        float v10 = Dot(g10, tx1, ty0);
        float v01 = Dot(g01, tx0, ty1);
        float v11 = Dot(g11, tx1, ty1);

        float tx = Smooth(tx0);
        float ty = Smooth(ty0);
        //return linear interpolation of the pairs of the top two points
            and bottom two   points, and the smoothed values of tx and ty
        return Mathf.Lerp(
            Mathf.Lerp(v00, v10, tx),
            Mathf.Lerp(v01, v11, tx),
            ty) * sqr2;
        //  since the maximum value is still reached at the center off a
            cell with four   diagonal gradients pointing at its center.
    }

}
```

As you can see from the functions above the Perlin2D method takes a given frequency, and a 3D vector (since we are using a 3D engine in a grid based game the Z component of the 3DVector will be used instead of the Y Component) which will be the location of a tile to generate a psuedo-random number. In order to do this the method does several steps. First it starts by converting the input vector to a 2D vector with floating values with a decimal component, this 2D vector is now between 4 grid points which all have gradient vectors. Second it calculates

the relative distance vector between one grid point and the previously calculated floating 2D vector point. It then calculates the dot products between the four gradient vectors and the relative distance vector. The 2D Perlin noise function then uses the smooth function which is to smooth the relative distance vector using a smooth curve function. The function

$$6t^5 - 15t^4 + 10t^3 \tag{5.1}$$

, has a second and first derivative of 0 at both t=0 and t=1 so the rate of change is always 0 at gradient boundaries. This is based of Ken Perlins improved Perlin noise, since the initial algorithm featured the smooth function

$$3t^2 - 2t^3 \tag{5.2}$$

but was then improved since this function had two deficiencies.Finally after the distance vectors have been smoothed the 4 dot products are now interpolated in pairs such that, the top grid point dot product values are interpolated with the x dimension and the bottom two points are interpolated by the X dimension and the value of both are interpolated with the Y dimension which is then returned as a value between -1 and 1.

```
//Function to render tile at coordinates of input c
private Tile CreateTile(Coordinates c)
{
    Tile t = Instantiate(tilePrefab) as Tile;
    t.xz = c;
   //assign a name to the tile
    t.name = "Tile:[" + c.x + "," + c.z + "]";
    t.transform.parent = transform;
    //set the new tile at the corresponding position
    Vector3 point = new Vector3(c.x - Vector.x * 0.5f + 0.5f, 0f, c.z
        - Vector.z * 0.5f + 0.5f);
    //render prefab at the location of the point Vecor
    t.transform.localPosition = point;
    //Count tiles
    tileCount++;
    //Add cell coordinates and tile to dictionary
    FullDung.Add(c, t);
    //Add current tile
```

```
        //freeTiles.Add(t.gameObject);
        //Call The Noise function
        Nfunction method = Noise.noiseDimensions[1];
        //The noise method takes the declared point and frequency as an
            input then returns a value between -1 and 1 if that value is
            greater than the user
        //determined threshold then build a wall
        if (method(point, (frequency)) > threshold)
          {
            //build wall at location c
              buildWall(c);
          }
        //return as Tile component
        return t;
    }


    //Construct new obstacle
     private Wall buildWall(Coordinates c)
     {
        //Randomly generate a number to determine which prefab to render
            to have variety in what obstacles are spawned
        int randomObstacleNo = Random.Range(0, 4);
        Wall w = null;
        if (randomObstacleNo == 0)
        {
            w = Instantiate(wPrefab) as Wall;
        }
        if (randomObstacleNo == 1)
        {
            w = Instantiate(wPrefab2) as Wall;
        }
        if (randomObstacleNo == 2)
        {
            w = Instantiate(wPrefab3) as Wall;
        }
        if (randomObstacleNo == 3)
        {
            w = Instantiate(wPrefab4) as Wall;
        }

        w.xz = c;
        //assign name of obstacle and the coordinates location
        w.name = "Obstacle:[" + c.x + "," + c.z + "]";
        w.transform.parent = transform;

        //set obstacle position
        w.transform.localPosition = new Vector3(c.x - Vector.x * 0.5f +
```

```
            0.5f, 0f, c.z - Vector.z * 0.5f + 0.5f);
        //some objects are not perfecly alligned with tiles so adjust
            certain prefabs when rendered to centere them in the middle
            of a tile.
        if(randomObstacleNo==0)
        {
            w.transform.Translate(1f, 0, 0.75f);
        }

        if(randomObstacleNo==1)
        {
            w.transform.Translate(0.7f, 0, 0.8f);
        }
        // remove the tile that occupies the cell the obstacle was placed
            on from the free tiles list.
        freeTiles.Remove(Gettile(c).gameObject);
        wallLit.Add(w);
        //return as wall component
        return w;
    }
```

Listed above is the methods used to render the tiles and obstacles to construct a level. As can be seen in the CreateTile method, the 2D Perlin noise function is used here in an if clause when building obstacles. The input of the method is 'point' being the vector position of the generated tile and user determined frequency .If the 2D Perlin noise value returns a value greater than the user determined threshold then the buildWall method is called. The buildwall method is then called if the noise threshold clause condition returns true. The method renders a random prefab by generating a number to select a prefab and an obstacle is rendered on top of the tile with the input vector that triggered the if clause on the 2D Perlin noise function.

When it came to rendering the tiles and obstacles to the screen I created a list of all possible coordinates within the search space, and then rendered them column by column removing coordinates from the list as the tiles were rendered.I used a coroutine to render each column in intervals to show the level generation. Below is an example of the implementation used to procedurally generate a level, the Generate method implementation uses the generation delay and coroutine technique used in the maze tutorial[20] .

```
public IEnumerator Generate()
```

```
{
    //instantitate a generation delay which is a delay for each tile
        generated.
    WaitForSeconds generationDelay = new WaitForSeconds(stepDelay);
    //call cleangame function to clear all gameobjects
    cleanGame();
    //Room area = area of the search space
    RoomArea = Vector.x * Vector.z ;
    //plot the map with the coordinates for each tile.
    for(int i=CD.x;i<Vector.x+CD.x;i++)
    {
        for(int j=CD.z;j<Vector.z+CD.z;j++)
        {
            Coordinates newXZ = new Coordinates(i, j);
            plotMap.Add(newXZ);
        }
    }
    //add new tile to the frontier
    freeTiles.Add(CreateTile(CD).gameObject);
    FullDung.Remove(new Coordinates(CD.x, CD.z));
    //while coordinates still exist in the plotMap list call
        GenerateLevel function to render tiles per interval
    while (plotMap.Count>0)
    {
        yield return generationDelay;
        GenerateLevel(freeTiles);
        //when all tiles are generated
        if (freeTiles.Count-1 == RoomArea)
        {
              //If level generated is too populated then return to
                 Menu
              if(wallLit.Count<15 ||
                 wallLit.Count>(freeTiles.Count/2))
               {
                  SceneManager.LoadScene("Menu");
               }
              //Spawn Player in
               Coordinates playerSpawn = new Coordinates(CD.x+5,
                   CD.z+5);
               SpawnPlayerIn(playerSpawn);
               //If VR is False Destroy VR Object
               if (VR == false)
               {
                   Destroy(GameObject.Find("Player/VRSimulatorCameraRig"));
               }
               //Else Destroy the players head object consisting of
                   the player arms, head and camera
               else
```

```
                {
                    Destroy(GameObject.Find("Player/Head"));
                }
                //Spawn enemyNo number of enemies at random locations
                    in the level
                for (int enemyNo = 1; enemyNo <= numberOfEnemies;
                    enemyNo++)
                {
                    Coordinates enemySpawn = new
                        Coordinates(Random.Range(CD.x, Vector.x),
                        Random.Range(CD.z, Vector.z));
                    SpawnEnemyIn(enemySpawn, enemyNo);
                }
                //Spawn a goal to return to
                Coordinates goalSpawn = new Coordinates((Vector.x +
                    CD.x) - playerSpawn.x, (Vector.z + CD.z) -
                    playerSpawn.z);
                SpawnGoalIn(goalSpawn);
                break;

        }
    }
}
```

```
private void GenerateLevel(List<GameObject> tiles)
{
        //select tile at end of list
        int Index = tiles.Count - 1;
        //current tile is assigned to tile at end of list
        GameObject current = tiles[Index];

    //Create a Column of tiles
    for (int i = 0; i < Vector.z; i++)
    {
        //if plotMap contains coordinates
        if (plotMap.Count != 0)
        {
            //assign c to coordinates at end of plot map list
            Coordinates c = plotMap[plotMap.Count - 1];
            //remove last element in plot map list
            plotMap.RemoveAt(plotMap.Count - 1);

            //if coordinates are within the search space (Vector.x
                and Vector.z max height and width of the search space)
            if (coordinatesCheck(c))
            {
                    //assign next tile to a new game obejct and create
```

```
                    tile at current coordinates
                GameObject nextTile = CreateTile(c).gameObject;
                //add nextTile to list of gameobject tiles to be
                    used as Waypoint later.
                tiles.Add(nextTile);
            }
        }
    }
}
```

5.3 States

In terms of handling the transitions between states involving the player and the IA (Enemy game objects), I used Enumerations for the states to enable me to use a collection of related constants to refer to. I implemented these states in a switch statement to handle the transitions between these states. Below is the implementation on how I handled the state transitions.

```
public class Player : MonoBehaviour {

    public Coordinates xz;
    public GameObject sight;
    public GameObject head;
    public float horizontalMovement = 2;
    public float verticalMovement = 2;
    //Enumerations for States of Enemy
    enum State { Alive,Dead,PoweredUp}
    State currentState = State.Alive;
    static bool[,] visited = new bool[500, 500];
    public static List<int> playerPath = new List<int>();
    public static Coordinates current;
    public static bool Powered;
    Direction currentDirection;
    void Update()
    {

        //Move Up
        if(Input.GetKeyDown(KeyCode.UpArrow) ||
            Input.GetKeyDown(KeyCode.W))
        {
            Coordinates Up = new Coordinates(0, 1);
            xz += Up;
           // print("X= "+ xz.x+ "Z= "+ xz.z);
           // player.transform.Translate(0, 0, Up.z);
```

```
    currentDirection = Direction.North;
}
//Move Right
if (Input.GetKeyDown(KeyCode.RightArrow) ||
    Input.GetKeyDown(KeyCode.D))
{
    Coordinates Right = new Coordinates(1, 0);
    xz += Right;
 //   print("X= " + xz.x + "Z= " + xz.z);
  //  player.transform.Translate(Right.x, 0, 0);
    currentDirection = Direction.East;
}
//Move Down
if (Input.GetKeyDown(KeyCode.DownArrow) ||
    Input.GetKeyDown(KeyCode.S))
{
    Coordinates Down = new Coordinates(0, -1);
    xz += Down;
//    print("X= " + xz.x + "Z= " + xz.z);
   // player.transform.Translate(0, 0, Down.z);
    currentDirection = Direction.South;
}
//Move Left
if (Input.GetKeyDown(KeyCode.LeftArrow) ||
    Input.GetKeyDown(KeyCode.A))
{
    Coordinates Left = new Coordinates(-1, 0);
    xz += Left;
  //  print("X= " + xz.x + "Z= " + xz.z);
   // player.transform.Translate(Left.x, 0, 0);
    currentDirection = Direction.West;
}
//print("Function Test: " + "X= " + Vector3ToCoordinates().x +
    "Z= " + Vector3ToCoordinates().z);
// direction enumerators dictating what direction the player
    moves in
switch (currentDirection)
{
    case Direction.North:
        this.gameObject.transform.Translate(0, 0, 0.1f);
        break;
    case Direction.East:
        this.gameObject.transform.Translate(0.1f, 0, 0);
        break;
    case Direction.South:
        this.gameObject.transform.Translate(0, 0, -0.1f);
        break;
```

```
            case Direction.West:
                this.gameObject.transform.Translate(-0.1f, 0, 0);
                break;
        }


        playerFSM();
        //print(Dungeon.Score);
       //Assign mouse speed
        float h= horizontalMovement * Input.GetAxis("Mouse X");
        float v = verticalMovement * Input.GetAxis("Mouse Y");
        head = GameObject.Find("Player/Head");
        //Set the sight rotation to match mouse movement
        if (head != null)
        {
            head.transform.Rotate(0, Input.GetAxis("Mouse X") * 8, 0);
        }

        visitingNodes();
    }

    public void playerFSM()
    {
        foundPowerUp();
        switch (currentState)
        {   //Dead state destroy the player object and reset the game
            back to level 1
            case State.Dead:
                Destroy(this.gameObject);
                Dungeon.complete = true;
                break;
            //Powered up state set powered to true with is used as a
                static variable in the Enemy.cs script
            case State.PoweredUp:
                Powered = true;
                //Invoke the setAliveState() method to trigger in x
                    seconds which is used as a time limit for how long
                Invoke("setAliveState", 10);
               // print("Power");
                break;
            //Alive state which sets powered to false which controls
                certain behaviours of the Enemy gameobjects
            case State.Alive:
                Powered = false;
                break;

        }
```

```
}


//Keeps a log of most up to date
public void visitingNodes()
{
    playerPath.Insert(0,Dungeon.freeTiles.IndexOf(Dungeon.Gettile(Vector3ToCoordina
    if(playerPath.Count>30)
    {
        playerPath.RemoveAt(playerPath.Count - 1);
    }

}
//Convet vector3 vector position to Coordinates
public Coordinates Vector3ToCoordinates()
{

    Coordinates output = new
        Coordinates((int)this.gameObject.transform.position.x+11,
        (int)this.gameObject.transform.position.z+16);
    current = output;
    return output;
}

public void cleanWaypoints()
{
    Dungeon.freeTiles.Clear();
}


public bool foundPowerUp()
{
    GameObject u;
    //If power-up has been destroyed (collected by player)
    if (GameObject.Find("Power-Up") != null)
    {
        u = GameObject.Find("Power-Up");
        if (Vector3.Distance(this.gameObject.transform.position,
            u.transform.position) <= 2.8)
        {
            //change state of player to powered up and destroy the
                power-up gameobject
            currentState = State.PoweredUp;
            Destroy(u);
            return true;
        }
    }
```

```
        return false;

    }


    //set state to alive.
    public void setAliveState()
    {
        currentState = State.Alive;
    }

}
```

```
public class Enemy : MonoBehaviour {
    public Coordinates xz;
    public GameObject p;
    public float speed;
    public static Coordinates current;
    //Enumerations for States of Enemy
    enum State { Roam, Dead, Flee, Chase }
    RandomIA randomPath;
    AstarIA astarPath;
    FleeIA fleePath;
    State currentState = State.Roam;
    Color[] colAr = new Color[4];
    int colorIndex;

    private void Awake()
    {
        //Assign colours to be used by the material of gameobjects
        colAr[0] = Color.blue;
        colAr[1] = Color.red;
        colAr[2] = Color.yellow;
        colAr[3] = Color.magenta;
        colorIndex = Random.Range(0, 4);
        randomPath = this.gameObject.GetComponent<RandomIA>();
        astarPath = this.gameObject.GetComponent<AstarIA>();
        fleePath = this.gameObject.GetComponent<FleeIA>();
        PathAgent.speed = speed;
    }


    void Update()
    {
        FSM();
    }

    public void FSM()
```

```
{
    //assign player gameobject.
    p = GameObject.Find("Player");




    switch (currentState)
    {
        // Dead state will destroy the gameobject this script
            (Enemy.cs) is attatched to.
        case State.Dead:
            Dungeon.Multiplier++;
            Dungeon.Score += 10*Dungeon.Multiplier;
            Destroy(this.gameObject);
            break;
        // Flee state will calculate a path to get away from the
            player.
        case State.Flee:
            //Assign all fleeing Enemys gameobjects's material to the
                white colour to represent the change in state.
            this.gameObject.GetComponentInChildren<Renderer>().material.color
                = Color.white;
            //If the gameobject is fleeing and is caught by the
                player then the state will change to dead.
            if (Vector3.Distance(p.transform.position,
                this.gameObject.transform.position) <= 2.8)
            {
                currentState = State.Dead;

            }
            //If player is not powered up anymore change state to
                Roam.
            if (Player.Powered == false)
            {
                currentState = State.Roam;
            }
            //If the game obejct was previous using the RandomIA
                script then remove it because Enemy is no longer
                chasing player.
            if (randomPath.enabled == true)
            {
                randomPath.enabled = false;
            }
            //If the game obejct was previous using the AstarIA
                script then remove it because Enemy is no longer
```

```
            chasing player.
        if (astarPath.enabled == true)
        {
            astarPath.enabled = false;
        }
        if (fleePath.enabled==false)
        {
            fleePath.enabled = true;
        }
        break;
    //Roam state will calculate random paths around the level
        until it sees the player (comes within distance of the
        player).
    case State.Roam:
        //Assign an appropriate colour to the game object this
            script is attatched to (Enemy.cs).
        if
            (this.gameObject.GetComponentInChildren<Renderer>().material.color
            != colAr[colorIndex])
        {
            this.gameObject.GetComponentInChildren<Renderer>().material.color
                = colAr[colorIndex];
        }
        //If the game obejct was previous using the AstarIA
            script then remove it because Enemy is no longer
            chasing player.
        if (astarPath.enabled == true)
        {
            astarPath.enabled = false;
        }
        //If the game object is not currently using the RandomIA
            script as a random path finding method then apply it
            as a component to this gameobject.
        if (randomPath.enabled==false)
        {
            randomPath.enabled = true;
        }
        //If gameobject sees the player then change state to
            chase.
        if (Vector3.Distance(p.transform.position,
            this.gameObject.transform.position) <= 15)
        {
            currentState = State.Chase;
        }
        //If the player is touching the enemy then destroy the
            player.
        if (Vector3.Distance(p.transform.position,
            this.gameObject.transform.position) <= 2.8)
```

```
            {
                Dungeon.Score = 0;
                Dungeon.Multiplier = 0;
                print("Final Score" + Dungeon.Score);
                Destroy(p);
                Dungeon.complete = true;
            }
            if (fleePath.enabled==true)
            {
                fleePath.enabled = false;
            }
            //If the player is powered up then change state to Flee.
            if (Player.Powered == true)
            {
                currentState = State.Flee;
            }

            break;
        case State.Chase:

            //If the game obejct was previous using the RandomIA
                script then remove it because Enemy is no longer
                chasing player.
            if (randomPath.enabled==true)
            {
                randomPath.enabled = false;
            }
            //If the game object does not have the component of the
                AstarIA script then apply it to calculate path to the
                player.
            if (astarPath.enabled==false)
            {
                astarPath.enabled =true;
            }
            //Destroy player if the Enemy (this.gameobject) catches
                player.
            if (Vector3.Distance(p.transform.position,
                this.gameObject.transform.position) <= 2.8)
            {
                Dungeon.Score = 0;
                Dungeon.Multiplier = 0;
                Destroy(p);
                Dungeon.complete = true;
            }

            //change state back to roam if the player is no longer in
                sight.
            if (Vector3.Distance(p.transform.position,
```

```
                this.gameObject.transform.position) >= 25)
            {
                currentState = State.Roam;
            }
            //change state to flee if player is powered-up.
            if (Player.Powered == true)
            {
                currentState = State.Flee;
            }
            if (fleePath.enabled == true)
            {
                fleePath.enabled = false;
            }
            break;

        }
        Vector3ToCoordinates();

    }

    //Convet vector3 vector position to Coordinates
    public Coordinates Vector3ToCoordinates()
    {
        Coordinates output = new
            Coordinates((int)this.gameObject.transform.position.x + 11,
            (int)this.gameObject.transform.position.z + 16);
        current = output;
        return output;
    }

}
```

Listed above is the implementation for the Player and Enemy classes which each handle the state transitions for their respective object. The player class has the simplest state machine device such that it only has 3 states depicted in the Enumerations. The three states are Alive, Dead and Poweredup. The Alive state is normal state the player will be in most of the time, in this state IA will chase the player if they come close to the player and will 'destroy' the player resulting in the player losing. The state will just result in the player game object being destroyed and is a result of the IA chasing and catching the player. The final state is the Poweredup state which triggers a state change in the Enemy game object and invokes the setAliveState() method to trigger in x amount of seconds, this is to ensure

the player is only in the Poweredup state for x amount of seconds. The Enemy class state machine is more complex in the way it handles the transitions between states.It still uses Enumerations for the collection of states only the conditions in which it changes states are more complex. There are four states in the Enemy class which consist of Dead, Roam, Flee and Chase. The Dead state similarly to the player class such that if the Enemy game object is in this state then it is destroyed. The Roam state assigns its natural colour and removes any path finding scripts that is not Random search and will begin to roam until the player is within x Distance of the Enemy then it will switch to the chase state. The chase state will remove any already assigned scripts that is not using A star search and will begin chasing the player navigating finding the shortest path to the player, using the A star path finding algorithm. If the player moves a great distance from the Enemy then it changes back to the Roam state in which it will begin to roam again. If the agent catches the player while it is in chase state then it is the player object that will be destroyed. The final state is the flee state in which no matter what state the Enemy is in (aside from the dead state) if the player picks up the Power up item then all Enemy game object will switch to the flee state. The flee state involves all Enemy game objects changing colour to white to represent the state change, and all enemy objects fleeing from the player moving to the furthest node from the players current position. When in the flee state Enemy game objects can be destroyed by the player if the player catches the Enemy.

5.4 Path finding

To handle the path finding aspect of the project I used the implementation of a way point system from a previous game AI project and modified it to work with this project. I changed the findwaypoints function to add all available tile game objects to the waypoint list. Furthermore in the buildgraph function I changed the code to enable all nodes to only have edges to other nodes with a 1.5 cost ensuring all adjacent nodes are connected. Lastly I changed the returnNearest function to return the node the current agents coordinates currently occupy.

```
public class WaypointGraph
{

    public Graph navGraph;
    public static List<GameObject> waypoints;
    public static int[,] arrayPoints= new
        int[Dungeon.freeTiles.Count,Dungeon.freeTiles.Count];
    public static Dictionary<GameObject, int> TileNode = new
        Dictionary<GameObject, int>();
    public GameObject this[int i]
    {
        get { return waypoints[i]; }
        set { waypoints[i] = value; }
    }

    public WaypointGraph(GameObject waypointSet)
    {
        //assign waypoints to new list
        waypoints = new List<GameObject>();
        //assign navGraph to new AdjacencyL
        navGraph = new AdjacencyListGraph();

        findWaypoints(waypointSet);
        buildGraph();
    }

    private void findWaypoints(GameObject waypointSett)
    {
        GameObject player = GameObject.Find("Player");
       // waypoints.Add(player);
       //use Dungeon.freeTiles a list of all game object tiles and add
           them as waypoints to the waypoint list
        if (Dungeon.freeTiles.Count != null)
        {
            for(int i=0;i < Dungeon.freeTiles.Count;i++)
            {

                waypoints.Add(Dungeon.freeTiles[i]);


            }
            Debug.Log(Dungeon.freeTiles.Count+Dungeon.wallLit.Count);
            // Debug.Log("Found " + waypoints.Count + " waypoints.");

        }
        else
        {
            //Debug.Log("No waypoints found.");
```

```
    }
}

private void buildGraph()
{

    int n = waypoints.Count;

    navGraph = new AdjacencyListGraph();
    //add all nodes to to the navgraph
    for (int i = 0; i < n; i++)
    {
        navGraph.addNode(i);

    }


    for (int i = 0; i < n; i++)
    {
        for (int j = 1; j < n; j++)
        {
            //calculate distance cost between nodes
            Vector3 wPos1 = waypoints[i].transform.position;
            Vector3 wPos2 = waypoints[j].transform.position;
            float cost = Vector3.Distance(wPos1, wPos2);
            if (cost <= 1.5)
            {
                navGraph.addEdge(i, j, cost);
                //Debug.Log(cost);
            }

           // waypoints.Find(arrayPoints[i, j]);

        }
    }
}

//find nearest node from current vector position
public int? returnNearest(Vector3 here)
{
    int Start =
        Dungeon.freeTiles.IndexOf(Dungeon.Gettile(Enemy.current).gameObject);
    Debug.Log("Node:" + Start);
    return Start;

}
```

```
}
```

The WaypointGraph.cs script uses the findWaypoints method to populate the list of waypoints by adding each object in the Dungeon.freeTiles list which is a static list containing the game object of each tile present in the current level. The buildGraph method then adds the number of nodes counted in the way points and then calculates the cost between any two nodes, if the cost is less than or equal to 1 then it adds an edge between the pair of nodes. This creates an edge between adjacent nodes.

5.4.1 Random

As described in section 5.3, the 'Roam' state adopts the RandomIA.cs script which randomly selects nodes and builds a path that the enemy IA will traverse to and will then continue to build paths until it changes states. I used this implementation from a previous game AI project. Below is the RandomIA script.

```
public class RandomIA : PathAgent
{
    public override Pathfinder createPathfinder()
    {
        return new RandomPathfinder();
    }

}

public class RandomPathfinder : Pathfinder
{
    public override List<int> findPath(int start, int goal)
    {
        //path of nodes
        List<int> path = new List<int>();
        //add starting node
        path.Add(start);
        //initialise random object rnd
        System.Random rnd = new System.Random();
        while (path.Count < 2)
        {
            //assign next node to a random number from 0 to number of
                nodes in the search space
            int nextNode = rnd.Next(0, Dungeon.freeTiles.Count);
```

```
            //add next node
            path.Add(nextNode);
        }


        return path;
    }
}
```

5.4.2 A*

As described in section 5.3, the 'Chase' state adopts the AstarIA.cs script which utilizes the A* search algorithm to enable the agent to traverse nodes to catch the player. To implement this I used BlueRaja's simple priority queue implementation which is licensed under the MIT license. I use the simple priority queue to enable me to enqueue nodes to a the queue using the corresponding cost between the current and previous visited nodes. I used this implementation from a previous game AI project. Below is the AstarIA script

```
public class AstarIA : PathAgent
{
    public static float heuristicCost(int a, int b)
    {
        //set gameobjects assigned to waypoints in the waypoints array.
        GameObject wayPoint1 = waypoints[a];
        GameObject wayPoint2 = waypoints[b];
        //create vector3 for two positions of the 2 waypoints
        Vector3 pos1 = wayPoint1.transform.position;
        Vector3 pos2 = wayPoint2.transform.position;
        // calculate the distance to be used as a heuristic
        float dx = (float)Math.Abs(pos1.x - pos2.x);
        float dy = (float)Math.Abs(pos1.y - pos2.y);
        return (float)Math.Sqrt(dx * dx + dy * dy);
    }

    public override Pathfinder createPathfinder()
    {
        //create a heuristic that will be calculated from two positions
        Heuristic cost = new Heuristic(heuristicCost);
        //return new AstarPathfinder with the heuristic 'cost'
        return new AStarPathfinder(cost);
    }
```

```
}

public delegate float Heuristic(int a, int b);

public class AStarPathfinder : Pathfinder
{
    protected Heuristic guessCost;

    public AStarPathfinder(Heuristic h)
    {
        guessCost = h;
    }

    public override List<int> findPath(int start, int goal)
    {
        //create Simple priority queue and Dictionary
        SimplePriorityQueue<int> frontier = new
            SimplePriorityQueue<int>();
        Dictionary<int, int> visitedFrom = new Dictionary<int, int>();
        Dictionary<float, float> costSoFar = new Dictionary<float,
            float>();
        //Create a stack to pop nodes back in to the list.
        Stack<int> nextDestination = new Stack<int>();
        List<int> l1 = new List<int>();
        List<int> l2 = new List<int>();
        List<int> path = new List<int>();
        //add starting node with cost of 0 to frontier
        frontier.Enqueue(start, 0);
        Debug.Log(start);
        //set visited from [start] to -1
        visitedFrom[start] = -1;
        //cost so far = 0;
        costSoFar[start] = 0;

        while (frontier.Count > 0)
        {
            //get the node at the head of the queue / lowest priority
            int current = frontier.Dequeue();
            //if head of queue = goal then simplePriority queue complete.
            if (current == goal)
            {
                break;
            }
            //create list neighbours of current node
            List<int> neighbours = navGraph.neighbours(current);
            foreach (int next in neighbours)
```

```
            {
                //calculate the cost accumalted so far.
                float nextCost = costSoFar[current] +
                    navGraph.cost(current, next);
                //If no neighbours cost have been compared or check if
                    new lower cost path has been found
                if (!costSoFar.ContainsKey(next) || nextCost <
                    costSoFar[next])
                {
                    //   Debug.Log("Next " + next + " Current: " +
                        current);
                    //add next node to priority queue with a cost of next
                        cost + the heuristic guess cost
                    frontier.Enqueue(next, nextCost + guessCost(next,
                        goal));
                    visitedFrom[next] = current;
                    //cost so far is added to the dictionary
                    costSoFar[next] = nextCost;
                    l1.Add(next);
                    l2.Add(visitedFrom[next]);
                }
            }
        }
        //push target goal to stack
        nextDestination.Push(goal);
        for (int destination = goal; destination != start; destination =
            visitedFrom[destination])

        {
            //keep pushing all the nodes visited in a dictionary
            nextDestination.Push(visitedFrom[destination]);
        }
        for (int i = nextDestination.Count; i > 0; i--)
        {
            //while stack is greater than 0 keep popping stack

            path.Add(nextDestination.Pop());
        }
        //return the path
        return path;
    }

}
```

5.4.3 Flee

As described in section 5.3, the 'Flee' state adopts the FleeIA.cs script which works similarly to the RandomIA.cs script, it sets the 1st waypoint of the path to be at the corners on the level causing the agents to traverse to a corner of the level and then begin to travel to random nodes that have not been visited by the player.

```
public class FleeIA : PathAgent {


        public override Pathfinder createPathfinder()
        {
            return new FleePathfinder();
        }



    public class FleePathfinder : Pathfinder
    {
        public override List<int> findPath(int start, int goal)
        {
            //path of nodes
            List<int> path = new List<int>();
            int [] corner = new int[4];
            corner[0] = 0;
            corner[1] = 49;
            corner[2] = Dungeon.RoomArea - 1;
            corner[3] = corner[2] - 50;
            start = Random.Range(0,corner.Length);
            //add starting node
            path.Add(start);

            //initialise random object rnd
            System.Random rnd = new System.Random();
            while (path.Count < 25)
            {
                //assign next node to a random number from 0 to number of
                    nodes in the search space
                int nextNode = rnd.Next(0, Dungeon.freeTiles.Count);
                if (!Player.playerPath.Contains(nextNode))
                {
                    //add next node
                    path.Add(nextNode);
                }
            }
```

```
            return path;
        }
    }
}
```

6 Testing

Contents

6.1 Introduction

In this section I will discuss and detail the tests that were carried out in order to ensure the game is fully functional, and detail the errors that were found to ensure the game meets its functionality and utility requirements.

6.2 White Box Testing

6.2.1 Function Testing

Table below reflects some of the white box testing that was conducted by myself in order to ensure the functionality of the project.

Function	Test Description	Input	Expected Output	Actual Output	Test Result
CreateTile(Coordinates c)	Testing function to render tile at Coordinates c	new Coordinates(0,0).	Renders a new tile at (0,0)	Renders a new tile at (0,0)	Pass
DestroyTile(Coordinates c)	Testing function to destroy tile at Coordinates c from previous test.	new Coordinates(0,0).	Destroy a new tile at (0,0)	Renders a new tile at (0,0)	Pass
BuildWall(Coordinates c)	Testing function to render a random obstacle at Coordinates c from previous test.	new Coordinates(0,0)	Render obstacle at new Coordinates(0,0).	Render obstacle at new Coordinates(0,0).	Pass
Gettile(Coordinates c)	Testing function to return Tile object from dictionary	new Coordinates(0,0) .	Return Tile object in Dictionary at Coordinates(0,0).	Return Tile object in Dictionary at Coordinates(0,0).	Pass
coordinatesCheck(Coordinates c)	Testing function to return a bool value if coordinates are within the level area	new Coordinates(0,0) .	Return true.	Return true.	Pass

Table 6.1: White box function testing.

6.3 Black Box Testing

6.3.1 System Testing

Below is the table documented of all the system tests that were done and the results.

During this testing phase I sought out an external tester who had no user experience with the tech demo, and as I monitored the tester conducted the following tests.

Test Number	Test Description	Input	Expected Output	Actual Output	Test Result (Pass/Fail)
1	Testing player movement, Up Direction	'Up Arrow Key' And 'W' Key	The player moves up one space, by coordinates (0,1).	The player moves up one space, by coordinates (0,1).	Pass
2	Testing player movement, Down Direction	'Down Arrow Key' And 'S' Key	The player moves down one space, by coordinates (0,-1).	The player moves down one space, by coordinates (0,-1).	Pass
3	Testing player movement, Right Direction	'Right Arrow Key' And 'D' Key	The player moves right one space, by coordinates (1,0).	The player moves right one space, by coordinates (1,0).	Pass
4	Testing player movement, Left Direction	'Left Arrow Key' And 'A' Key	The player moves left one space, by coordinates (-1,0).	The player moves left one space, by coordinates (-1,0).	Pass
5	Testing level generation using 2D Perlin noise.	Threshold = 0.3 And Frequency = 0.1	A level with psuedorandom placement of obstacles.	See fig 6.1 below.	Pass
6	Testing level generation using 2D Perlin noise.	Threshold = 0.4 And Frequency = 0.8	A level with psuedorandom placement of obstacles.	See fig 6.2 below.	Pass
7	Testing level generation using 2D Perlin noise.	Threshold = 0.5 And Frequency = 0.9	A level with psuedorandom placement of obstacles.	See fig 6.3 below.	Pass
8	Testing level generation using 2D Perlin noise.	Threshold = 0.1 And Frequency = 0.1	A level with psuedorandom placement of obstacles.	See fig 6.4 below.	Pass
9	Testing level generation using 2D Perlin noise.	Threshold = 0.3 And Frequency = 0.3	A level with psuedorandom placement of obstacles.	See fig 6.5 below.	Pass

Table 6.2: List of tests carried out for system testing.

Test Number	Test Description	Input	Expected Output	Actual Output	Test Result (Pass/Fail)
10	Testing level generation using 2D Perlin noise.	Threshold = -0.3 And Frequency = 0.2	A level with psuedorandom placement of obstacles.	See fig 6.6 below.	Fail
11	Testing level generation using 2D Perlin noise.	Threshold = 0.8 And Frequency = 0.5	A level with psuedorandom placement of obstacles.	See fig 6.7 below.	Fail
12	Testing Power up colour of player power-up state transition	Collect power-up	Agents Colour change to white	See fig 6.8 below.	Pass
13	Testing 'Chase' state	Move player within 15 Vector3 distance of Enemy IA	EnemyIA chases the player using A* search algorithm	EnemyIA chases the player using A* search algorithm.	Pass
14	Testing 'Roam' State	N/A	Enemy IA random traverse between nodes.	Enemy IA random traverse between nodes.	Pass
15	Testing the mouse X inputs for rotating the camera	Moving mouse X left to right.	Camera movement corresponds with mouse x movement.	Camera movement corresponds with mouse x movement.	Pass
16	Testing the reset button	Space bar	Level and score resets.	Level and score resets.	Pass
17	Testing input of number of enemies	Number of Enemies = 1.	Spawn 1 IA when level is finished generated	Spawn 1 IA when level is finished generated	Pass
18	Testing input of number of enemies	Number of Enemies = 4.	Spawn 4 IA when level is finished generated	Spawn 4 IA when level is finished generated	Pass

Table 6.3: Black box testing for the unity project.

6.3.2 Results

Test 5, 6, 7, 8, 9

Test 5:

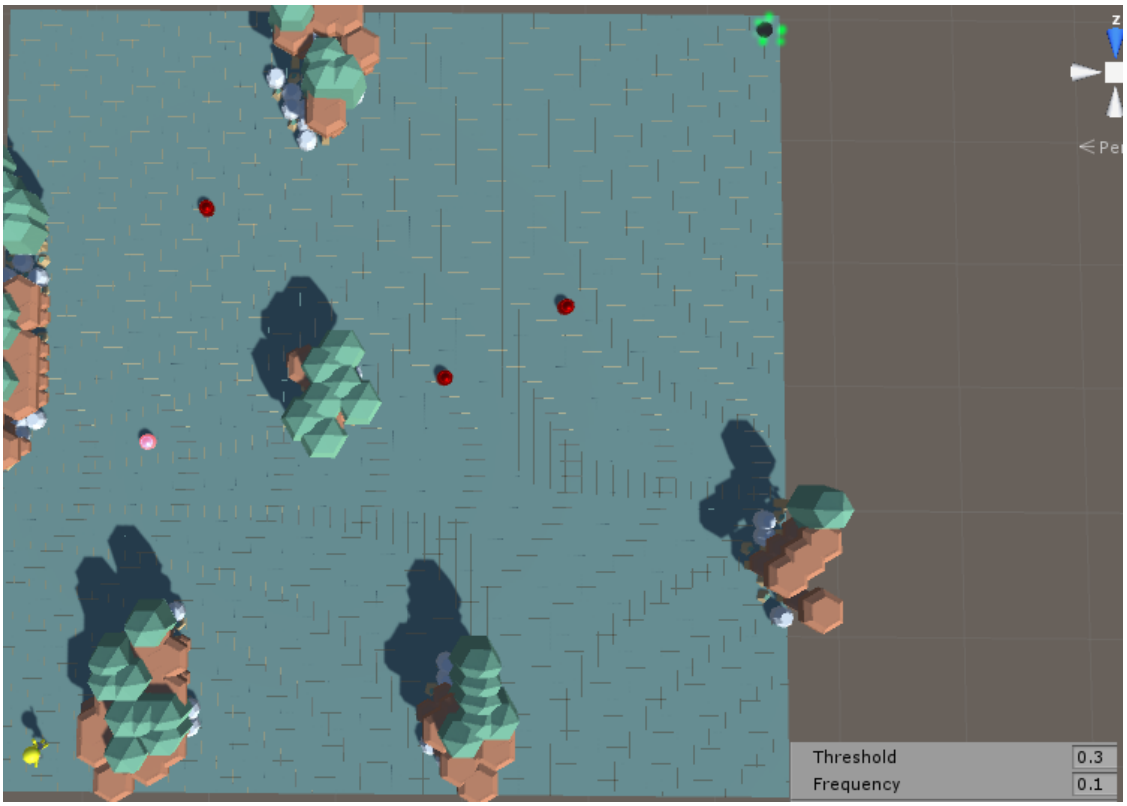

Figure 6.1: Above shows the inputs (Threshold and frequency) and the corresponding output (the generated level) for this test.

Test 6:

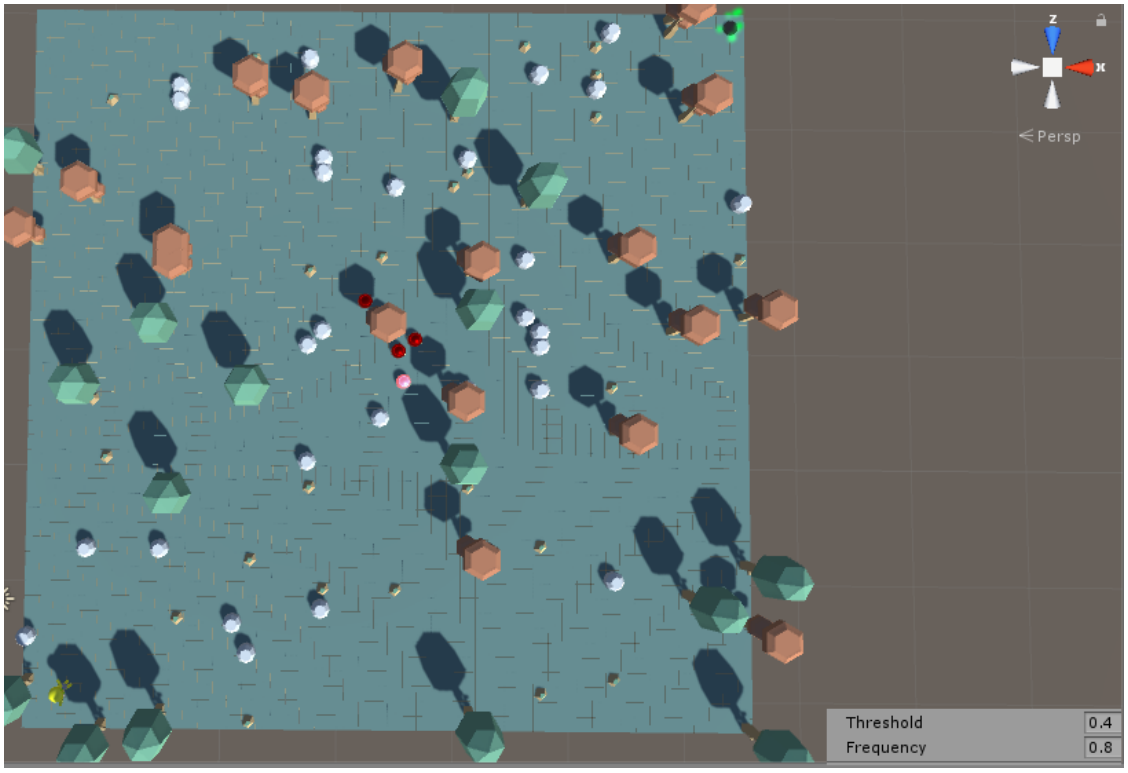

Figure 6.2: Above shows the inputs (Threshold and frequency) and the corresponding output (the generated level) for this test.

Test 7:

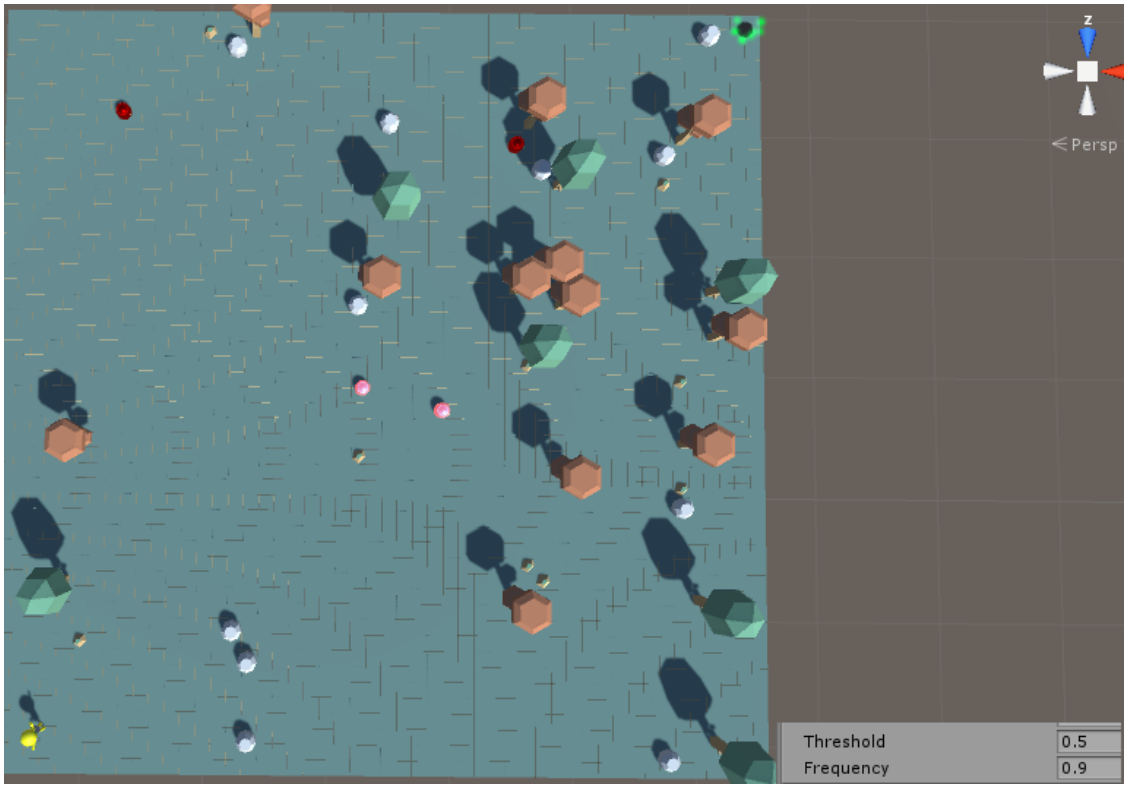

Figure 6.3: Above shows the inputs (Threshold and frequency) and the corresponding output (the generated level) for this test.

Test 8:

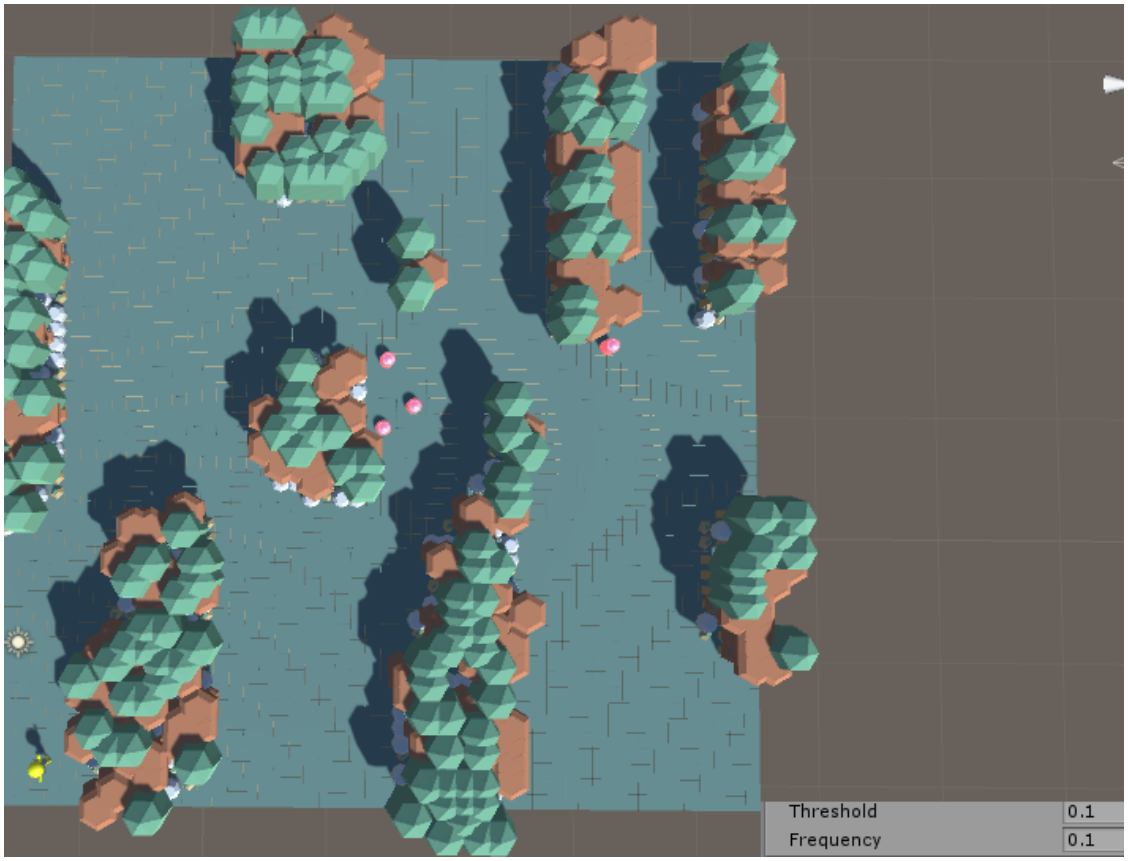

Figure 6.4: Above shows the inputs (Threshold and frequency) and the corresponding output (the generated level) for this test.

Test 9:

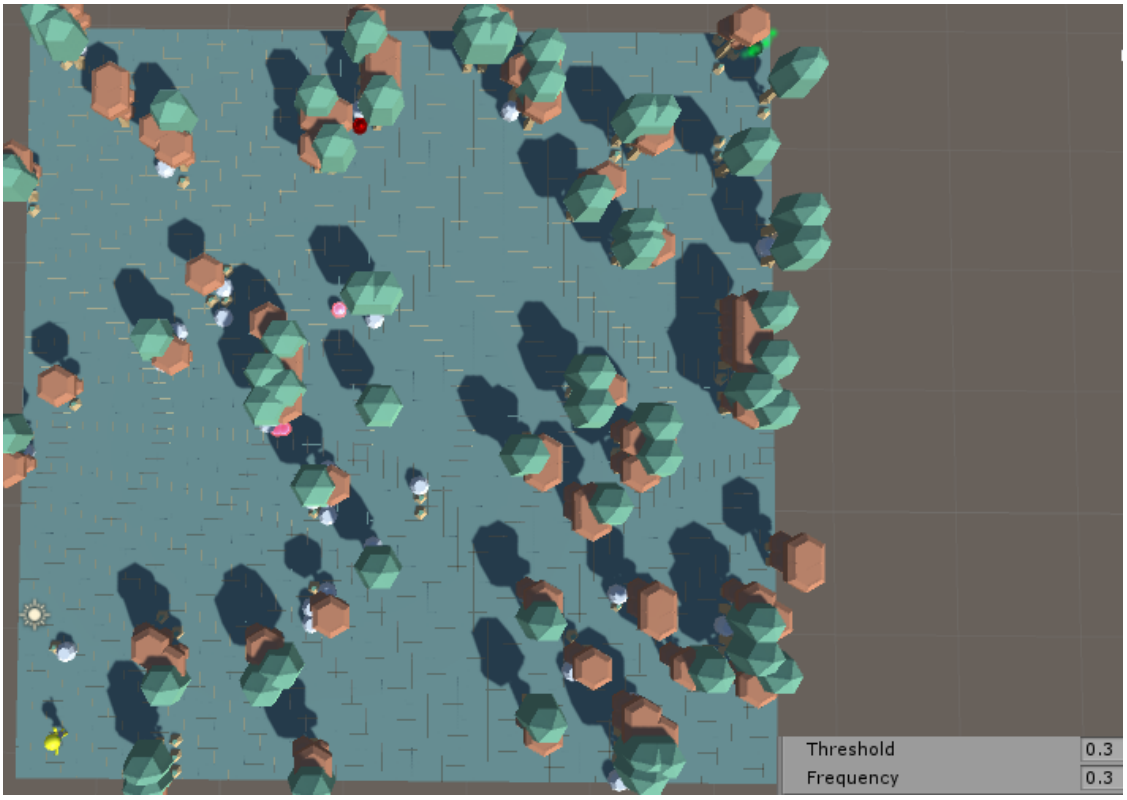

Figure 6.5: Above shows the inputs (Threshold and frequency) and the corresponding output (the generated level) for this test.

These are the results from the tests 5, 6, 7, 8, 9 It indicates that using the result of 2D Perlin noise based on the vector positions of each cell in the level to generate obstacles to produce unique levels proved to be a success among these tests.

Test 10, 11

Test 10:

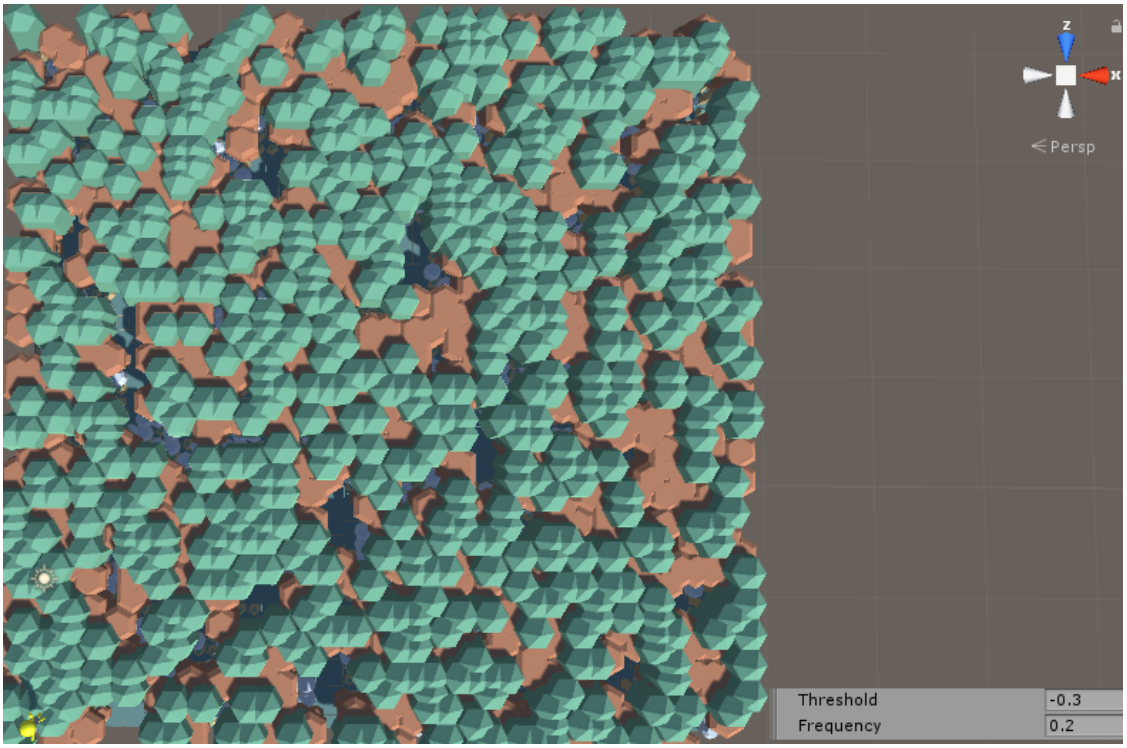

Figure 6.6: Above shows the inputs (Threshold and frequency) and the corresponding output (the generated level), which produces a level which is not suitable to be used.

Test 11:

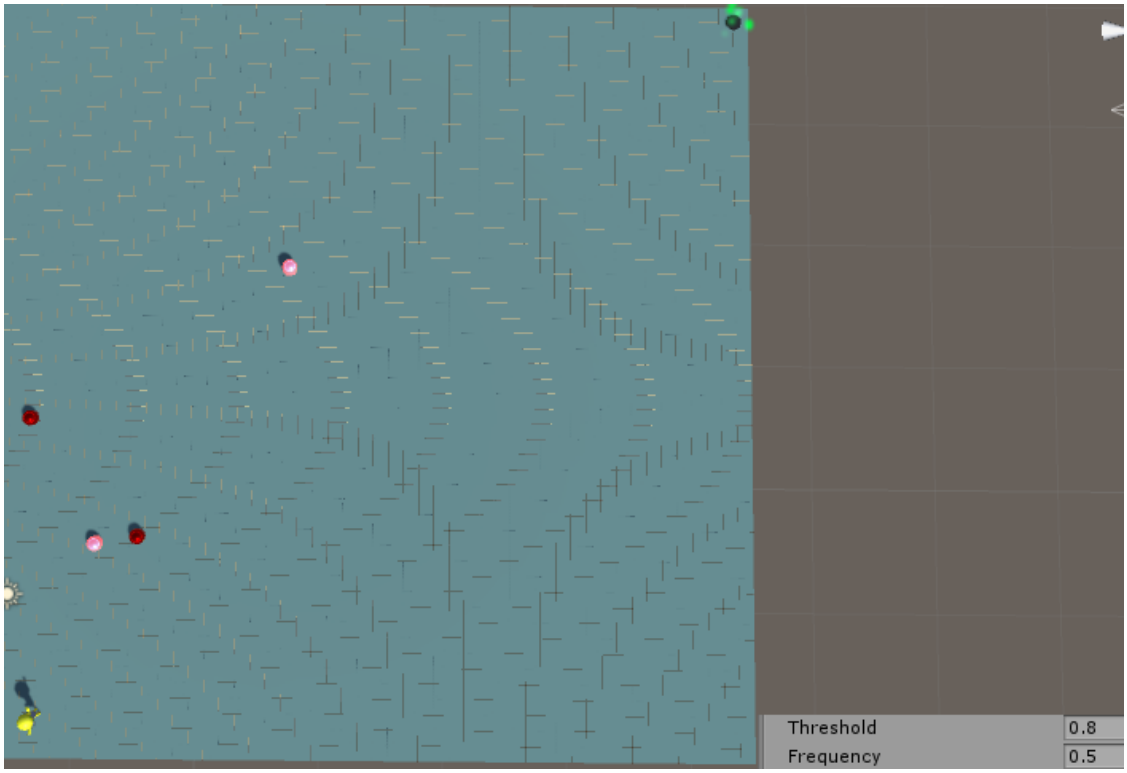

Figure 6.7: Above shows the inputs (Threshold and frequency) and the corresponding output (the generated level), which produces a blank level with no generated obstacles.

(6.1)

These are the results for the tests 10, 11 it shows that despite previous tests passing, using the return value of the 2D Perlin noise as a threshold can sometimes produce levels with an undesirable output. Test 10 shows a level that generates obstacles at every cell which is due to the low threshold. Test 11 shows a level in which the 2D Perlin noise value threshold was never met which generated a blank level which isn't a uniquely designed level. I fix this by simply adding an if clause when the level is generated. If the number of obstacles is less than 15 or greater than half the number of available free tiles then it will return to the main menu.

```
if(wallLit.Count<15 || wallLit.Count>(freeTiles.Count/2))
                {
                    Application.LoadLevel("Menu");
                }
```

Test 12

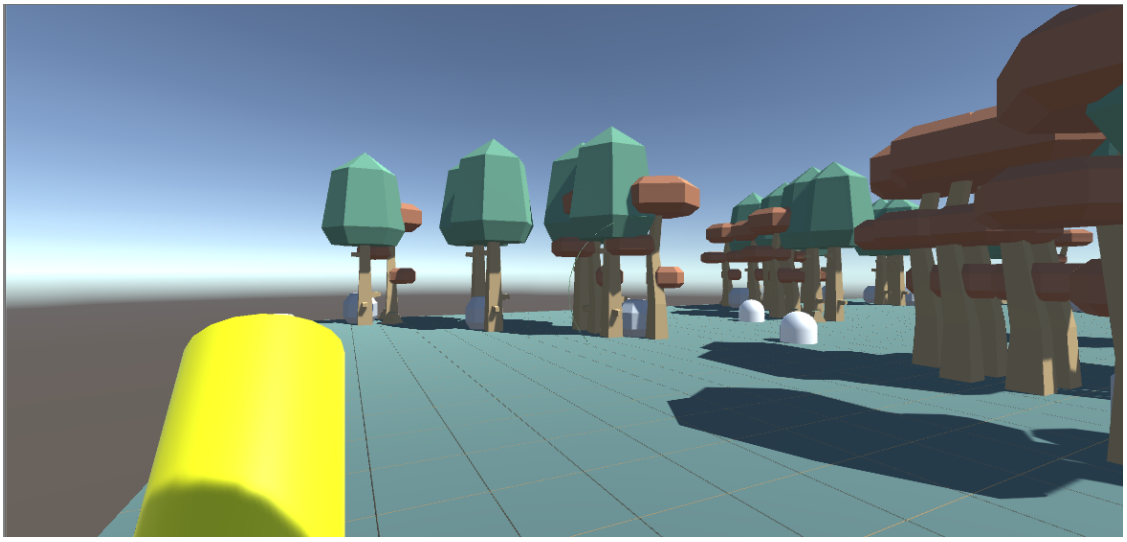

Figure 6.8: Colour change representing change in state.

(6.2)

Test 12 shows the Enemy game objects will change colour to white when they transition to the flee state. The test was a success.

6.3.3 Final Thoughts

After conducting all the testing I am able to conclude that my project is fully functional and meets all the functionality and usability requirements stated in chapter 4.

7

Evaluation of the System

Contents

7.1 Introduction

In this section I will discuss the criteria in which this project can be deemed a success. I will list my evaluation criteria and conduct surveys in order to assess the success of the project.

7.2 Evaluation Criteria

	Evaluation Criteria	
i	Users can control and navigate a playable character in a first person perspective.	**Complete**
ii	The level is procedurally generated at the start and no two levels visually look exactly the same	**Complete**
iii	IA have fully integrated path finding agent capabilities and are able to track the player using the level as a navigation graph.	**Complete**
iv	Each IA has several states that can change depending on the tiles they have explored and the state of the player.	**Complete**

Table 7.1: Evaluation Criteria.

7.3 Self Evaluation

Below is a discussion of my own evaluation of the system and how successful I think the final project is. In the appendix I have included links to videos of trials I carried out for my own evaluation of the project.

7.3.1 Criteria:[i]

After carrying out a series of runs of the project each time I was able to traverse the graph using a rendered character model. As can be identified in the videos provided in the appendix I am able to navigate in a first person perspective in each of them and flee from any agents chasing the character therefore I can consider this criteria met.

7.3.2 Criteria:[ii]

I carried out a series of runs of showcasing levels being procedurally generated with separate inputs for the frequency and threshold. Each level was successfully generated and I was able to prove that I can successfully generate pseudorandom levels each with a unique layout. Since I am able to do this I can consider this criteria met.

7.3.3 Criteria:[iii]

In the series of runs I did I was able to demonstrate that the IA are capable of moving from one node to another. I was able to demonstrate as the player got close to the IA, the IA was able to chase the player creating multiple paths with the goal state constantly changing to the node the player is located at. Since I was able to observe this I can consider this criteria to be met

7.3.4 Criteria:[iv]

In the series of runs I did I was able to showcase the states that the IA feature in this project. I was able to show the instances when the player is far away from the agent they are in the 'Roam' state in which they are using a random search. I was also able to show that when the player gets close to the agent they will begin to chase the player in the 'Chase' state using the A star algorithm. Lastly I showed when the player picks up a power-up the agents change colour and begin to flee from the player in the 'Flee' state. Since I was able to demonstrate that all of the state transitions featured in this project work I can consider this criteria to be met.

7.3.5 What went right

I was able to successfully build a project in unity which utilizes Ken Perlin's Perlin noise algorithm as a method of populating the level with obstacles. In addition I was also able to render several IA which use state machines to dictate the IA current goal. These states of the IA also use path finding algorithms commonly used in AI and video games.Combining these techniques I was able to successfully fulfil the original objective of this project which was to build a game demonstrating some techniques that can be used in video games. Finally I can say that in my opinion the game meets each condition of the evaluation criteria and can officially be deemed a success, and complete.

7.3.6 What went wrong

Despite much going right for this project, I did face some problems due to various reasons such as time constraints. For example when the game is running, occasionally when the IA have to change states from 'Roam' to 'Chase' it can cause the game to suffer and stop and take pauses momentarily for 1-2 seconds. This is due to the IA changing algorithms from random search algorithm to A* search and recalculating and finding the shortest path from the IA current position to the player from a navigation graph with around 2500 nodes. The 1-2 seconds can be considerably longer or shorter depending on the specifications of the station you are using to run the game. I also used low poly assets for the objects on screen to keep the load times shorter. I would have also liked for the game to be able to dynamically expand the search space or make the render a bigger search space from the beginning but this had to be limited by time and performance constraints as a bigger search space would cause the game to pause for longer.

7.4 User Evaluation

The user evaluation consisted of me sending my packaged unity project and an online survey to a sample group of people and getting them to assess the quality of the project. Before filling in the survey, users were instructed to play the game for a minimum of 5 minutes and were advised optimal values for frequency and threshold from my own experience. Below is example of the responses I received.

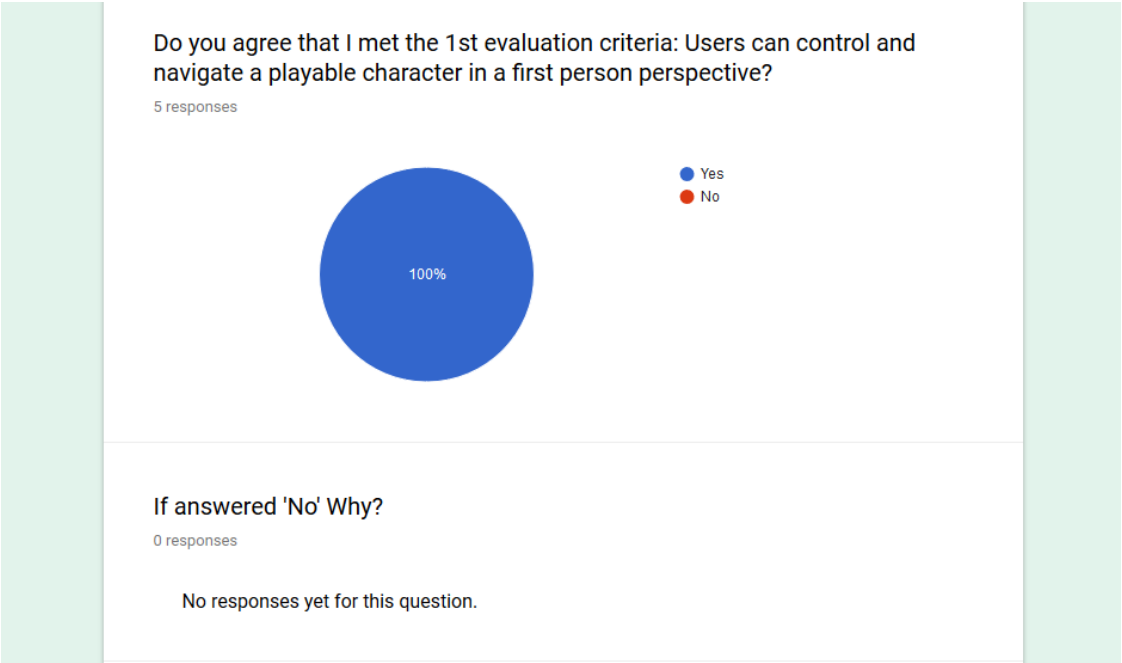

Figure 7.1: I conducted an online survey these are the results surrounding criteria i

The above diagram represents the responses I received regarding the success of the evaluation criteria [i]. This was the most basic criteria and in order to consider it a success. Users would have to agree they were able to control and navigate a playable character in first person perspective. As you can see from the above results all 5 responses agreed the project is atleast capable of meeting this criteria. Therefore at the very least I can assume this part of the project is fully functional and I can consider the criteria to be a success.

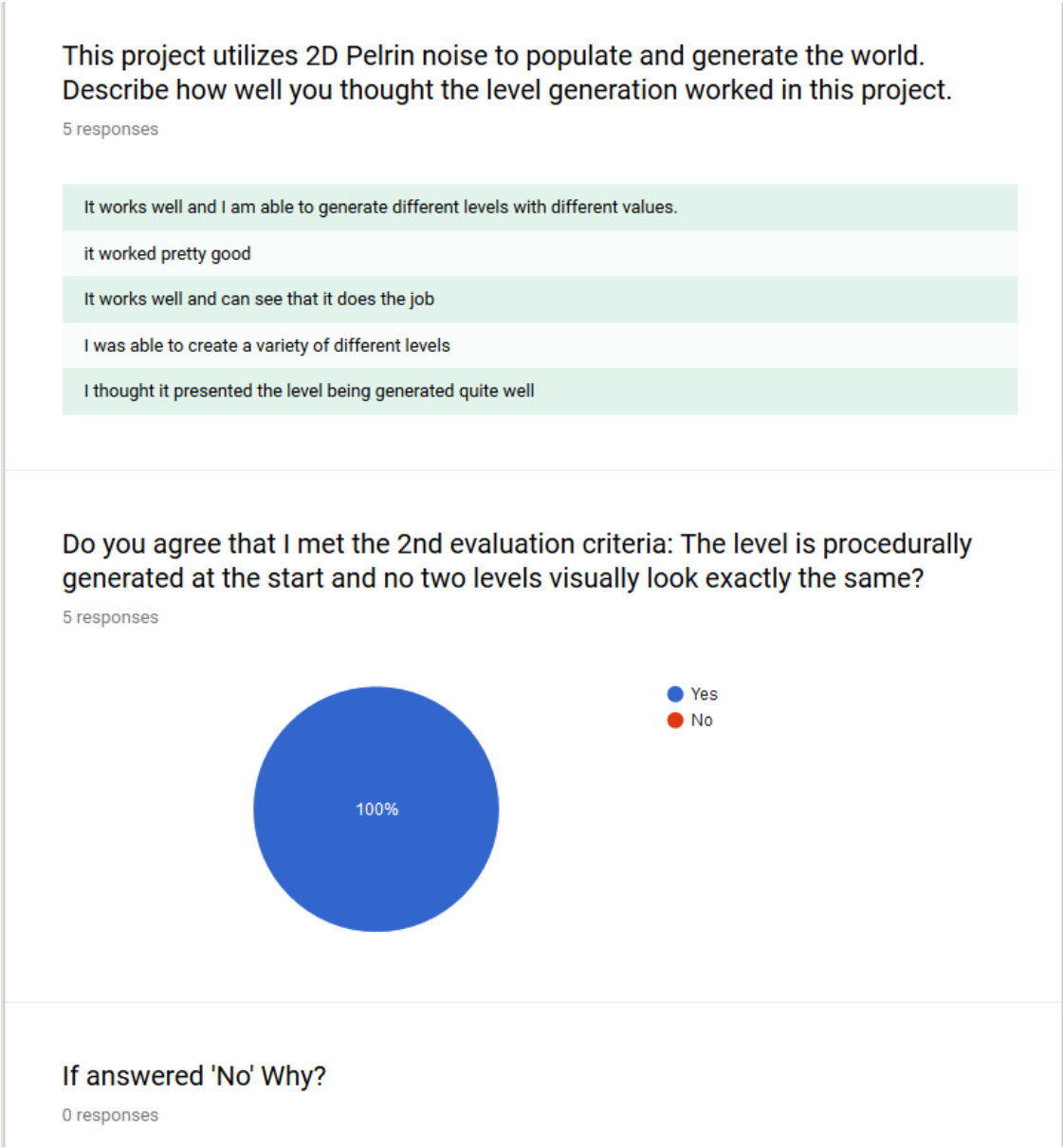

Figure 7.2: I conducted an online survey these are the results surrounding criteria ii

The above diagram represents the responses that I received in regards to success of the evaluation criteria [ii]. In which after playing the game for atleast 5 minutes all of of the respondents agreed that I met the criteria, which involves procedurally generating levels and no two levels looking the same. Some users also commented on their success stating "It worked well and I am able to generate different levels with different values", "I was able to create a variety of different levels" and "I thought it presented the level being generated quite well". From the positive

responses I received regarding this criteria I am able to at least say that the level generation in this project functions as intended, and I am successfully able to produce psuedorandom levels which look different depending on input values from the threshold and frequency. Therefore I can consider this criteria to be a success.

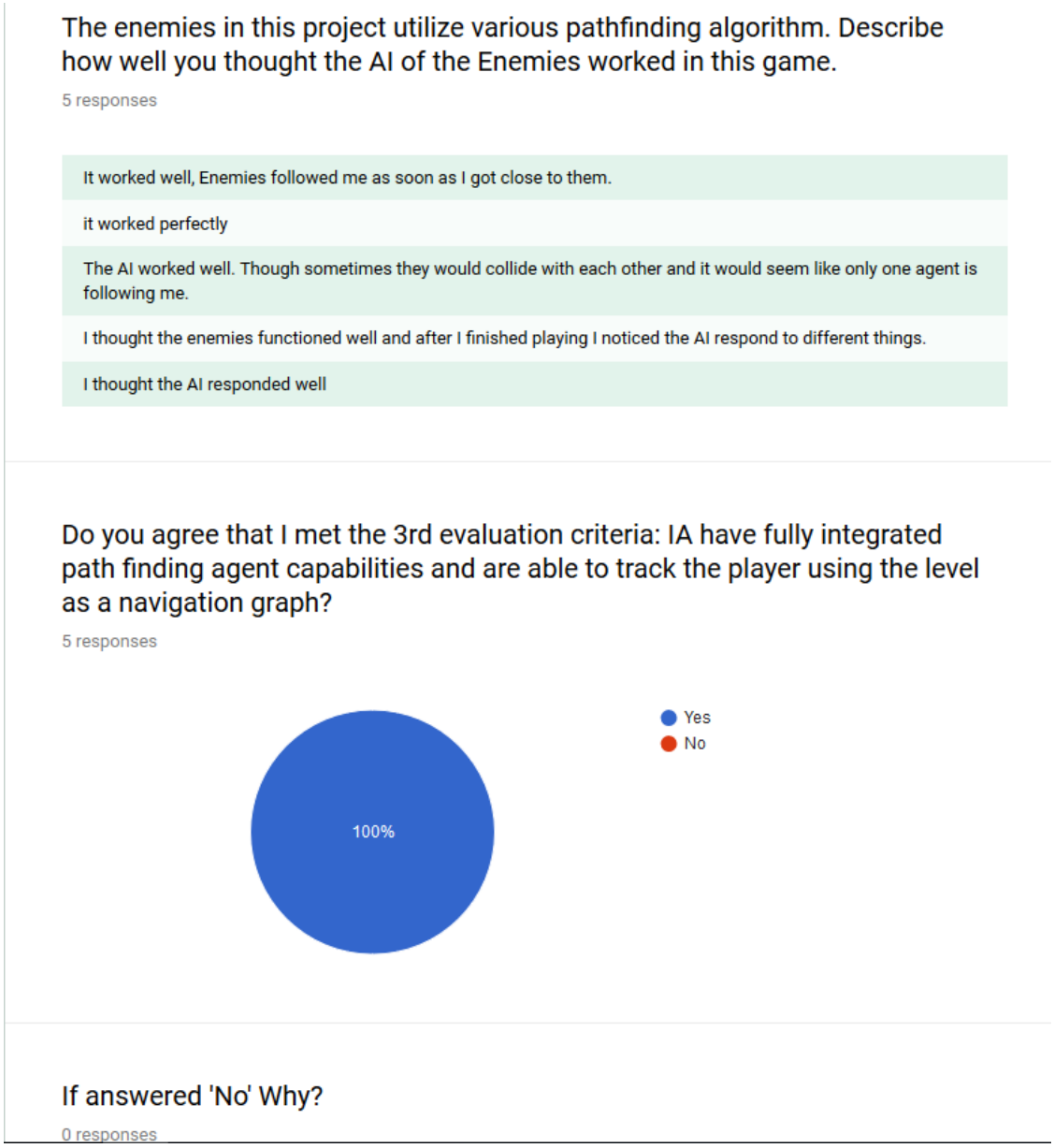

Figure 7.3: I conducted an online survey these are the results surrounding criteria iii

The above diagram represents the responses that I received in regards to the success of the evaluation criteria [iii]. This criteria involved the AI of the agents

and their path finding capabilities. Again all 5 respondents believed I had met the criteria and users also commented on how well it was implemented stating "It worked well, enemies follow me as soon as I get close to them" and "The AI worked well,Though sometimes they would collide with each other and it would seem like only one agent is following me". The results from the respondents clearly indicate that they believe I met the criteria for the objective so I can consider the AI of these agents to be functional, however I can still improve this functionality by implementing a multi agent algorithm and optimizing the A* algorithm to be aware of other agents. Overall I can still consider this criteria to be a success though since I the respondents felt the project accomplished this.

Figure 7.4: I conducted an online survey these are the results surrounding criteria iv

The above diagram represents the responses that I received in regards to the

success of the evaluation criteria [iv]. This criteria success was dependant on the state transitions in the IA. All respondents agreed that I met the evaluation criteria and some commented on the success of this criteria stating "I think it worked well I noticed changes in states when I got close to the IA it would chase me and when I picked up a green object the enemies would turn white and move in the opposite direction." and "This worked well, I noticed several changes in the states of enemies". From these respondents I can tell that the state changes are well implemented and functional. Therefore I can say that this criteria is a success.

7.5 Final Thoughts

In summary from my own evaluation and user evaluation I can say that I have successfully met all the evaluation criteria and can say the project functions how I initially intended. Furthermore from my results I can say I successfully managed to implement all the AI techniques I initially planned to implement.

Despite me successfully meeting all the evaluation criteria and the project functioning as intended, I can still further improve the project beyond the conclusion of this report. For example as I mentioned in 7.3.6, the project can face significant load times due to agents changing search algorithms and calculating new paths based on a large navigation graph. Furthermore one of the statements in regards to user evaluation of criteria [iii] mentioned that sometimes agents "collide with each other" and would move as one unit, I could improve this by improving the path finding algorithms.

8 Conclusion and Future Work

Contents

8.1 Introduction

In this section I am going to discuss the conclusion to this report and my plans for further work and development on the software.

8.2 Conclusion

The original goal when starting this project was to produce a software implementation in the form of a Unity project which presents certain AI techniques commonly used in video games. I was able to implement several techniques such as producing a PCG algorithm which is able to procedurally generate a search space with obstacles using 2D perlin noise values. This technique allowed me to provide levels to users which are unique for each user. Furthermore I was able to integrate a FSM device into each enemy featured in the game which utilize different search algorithms

with the goal node changing depending on the current state. Prior to starting this project I had minimal knowledge of Unity and PCG techniques, through developing and conducting the literature review I was able to gain an understanding on AI techniques and the modern usage of AI in video games. I was able to produce feasible solution to procedurally generating content in this project, and gained a better understanding on using Unity.

8.3 Future Work

While the original purpose of developing the software project was to present certain AI techniques commonly used in video games. I still have plans to further improve the quality and performance of this software and implement more AI techniques to expand the goal of the original project.

- Optimize the code to reduce the pauses mentioned in the evaluation when enemies are transitioning between states. Also improve the overall aesthetics of the project by creating my own 3D models or purchasing a more pleasant looking high poly 3D models which will be suitable for this project from the Unity asset store.

- Continue work on one of the extension goals for the project, which was to make it VR compatible. I plan to make the game compatible with more VR devices and enable users to interact with objects in the search space.

- Improve the complexity of the AI of the enemies featured in the game. I plan to do this by implementing behaviour trees to manipulate how each individual enemy behave and react to their environment. Furthermore by optimizing the A* algorithm of the enemy IA, I can implement multi-agent pathfinding to enhance the ways the IA find their target (the player).

- Further down the line I also want to change the navigation system from a navigation graph to a navigation mesh and add a height component to this

project alongside a 3D perlin noise implementation which will improve the variety in levels.

Bibliography

1 . . . . . Stuart J. Russell and Peter Norvig,1995, Artificial Intelligence A Modern Approach, Page 31 & Chapter 2 & Page 526.

2 . . . . . Siyuan Xu, History of AI design in video games and its development in RTS games, Department of Interactive Media I&' Game development, Worcester Polytechnic Institute, USA, Retrieved February 2017.

3 . . . . . Ben Coppin,2004 , Artificial Intelligence Illuminated, Page 548.

4 . . . . . Matt Buckland,2005, Programming Game Ai by Example, Page 44.

5 . . . . . David M. Bourg, Glenn Seeman, July 2004,AI for Game Developers,Chapter 6.2.

6 . . . . . Amit Patel,Introduction to A*,Retrieved March 2017, Source.

7 . . . . . Xiao Cui and Hao Sh, January 2011 , A*-based Pathfinding in Modern Computer Games, School of Engineering and Science, Victoria University, Melbourne, Australia.

8 . . . . . Alex Schearer, February 2009, Pathfinding with A*, Retrieved March 2017, Source.

9 . . . . . Ian Millington, John Funge, 2009, Artificial Intelligence for Games, Page 823.

10 . . . . . Sid Meier's Civilization V, Firaxis Games, Retrieved March 2017, Source.

11 . . . . . Herman Tulleken, 2008, Dev.Mag, Issue 20, Source.

12 . . . . . Red Blob Games, 2013, Noise Functions and Map Generation, Source.

13 . . . . . Federico Mazzini, 2016, Early attempts of Writing a Procedurally Generated 2D Terrain Using Perlin Noise in Swift, Source.

14 . . . . . Ken Perlin, 2002, Improving Noise, Source.

15 . . . . . Igor Borovikov, 2011 ,Navigation Graph Generation, Retrieved March 2017, Source.

16 . . . . . Jamey Pittman, 2009, The Pac-Man Dossier:Page 3, Retrieved March 2017, Source.

17 . . . . . Jamey Pittman, 2009, The Pac-Man Dossier: Page 5, Retrieved March 2017, Source.

18 . . . . . Gillian Smith, Alexei Othenin-Girard, Jim Whitehead, Noah Wardrip-Fruin, PCG-Based Game Design: Creating Endless Web, University of California, Santa Cruz, Retrieved February 2017, Source.

19 . . . . . Implementation of Perlin Noise algorithm, Source.

20 . . . . . Maze Tutorial, Source.

Appendices

Contents

.1 Appendix A-Project Proposal

Final Project Proposal: Project Title: AI Tech Demo / Navigation Game Project Timeframe Dec 2016-May 2017 (5 months) Project Developer: THOMPSON-HALL, Kane | 510102AY3 Supervisor: Devlin, Kate Overview / Design: I am planning on developing an AI Tech demo game in which I will be procedurally generating a 3d tile based world using either perlin or simplex noise to place the tiles. This 3d world will also build and represent a regular navigational graph which each tile generated will represent a node. I will have the player (user) navigate their way towards the goal, and once the player reaches the goal the world will again be procedurally generated with a different 3d environment. There will be several types of tiles generated that will affect the player, and other AI agents state, such as tiles that if moved to will slow the player/AI agent down. In addition, in this 3d world I will implement multiple agents that will utilise a finite-state machine to be able to be able to react to their environment and the player's actions. I also intend on implementing various path finding techniques in each of these agents to navigate their way around the world, some agents will wonder aimlessly by a form of random search, and other agents will try to obstruct/ attack the player by searching for the player using algorithms such as A, Dijkstra's, and D* algorithms which I will optimize to improve performance. If I have time I also plan on implementing VR to this tech demo using google cardboard to enhance the experience.

Technologies: To develop this project, I will be using Unity game engine in which I will use for certain assets for the tiles, player and agents. I will be using C#' to program this tech demo and I will be using the visual studio ide to develop it in. For the unity scene, I may initially use simple objects for the player agents but, If I have the time I may also use Maya (3D animation and modelling software) to create assets for my unity scene. If I implement a VR experience I will also be using google cardboard or other VR device and an android phone to implement and test the experience. I will have access to unity with a personal licence with my home computer and laptop also at computing labs at the university of goldsmiths. In addition, if I implement VR I will purchase my own device and android phone and I will have access to VR Devices at the university of goldsmiths.

Techniques: I will be using the techniques I have learned in my Games Ai module by implementing and procedurally generating a 3D world. I will also be implementing Finite state machines in the agents from techniques I have learned in games AI. I will also implement algorithms learned from AI such as path finding algorithms. User Requirements: The target audiences for this game, are Computer science enthusiasts / potential enthusiasts whom are interested in the implementation and how the game works. Second are game enthusiasts people that play games regularly and are over the age of 13.

Ethical Audit: All software and technologies used in the development process of the project will either be open sourced and free for me to use or I will have a student / free licence to use the software for the purposes I intend to use and develop it for. In addition will not be working with any minors or vulnerable adults during and after the development of the projects.

Requirements: Procedurally generated 3d world using a form of noise which enables the user to complete levels and experience a completely new level each time and that no two user experiences will be the same. AI agents will be implemented which feature Finite state machines and those states are affected by its environment and the player's actions. In addition, AI agents, will all also feature path finding algorithms from aimless movement to utilizing more complex algorithm to track

and attack the player. Extra requirements: If I have time I will also implement VR features using google cardboard or other VR Equipment.

Evaluation and Development Plan: I will be approaching this using a waterfall development approach in which I will tackle each stage of the development process sequentially, with a huge focus on planning. When the project is finished, I will create an Evaluation criteria in which I will include my requirements and survey a sample group to comment if they believed my project matched these requirements, and then once I have received the feedback I will then assess their comments to determine if my requirements have been met and I can consider this project a success. I will also use feature analysis in which I will match the requirements to features that I have implemented and describe why I believe this requirement has been met.

Project Plan: I plan to use a gantz chart to keep track and monitor the projects development process.

Requirements: January- I plan to identify the stakeholders of the project outline all the requirements for the Create use case diagrams. Research watch tutorials on how to implement certain features I plan to implement Design: Create initial UML diagrams Outline the assets I am going to need for this system Initial designs on how I want the game to look Implementation: Begin implementation and integration of the software If time is available implement planned features / extend scope of project. Verification: Begin testing evaluate the project. Maintenance: Package the project for potential distribution. Evaluate project on potential feedback from google play store etc.

.2 Appendix B-Gantt chart

Figure 1: Gantt chart indicating the project plan and timeline I had when doing this project.

(1)

Google Sheets Source.

.3 Appendix C-Evaluation Videos

Video demonstrating the transition from the 'Chase' state to 'Flee': Source.

Video demonstrating the transition from the 'Roam' state to 'Chase':Source.

Video demonstrating the levels being procedurally generated with different frequency and threshold inputs:Source.

.4 Appendix D-Evaluation Survey

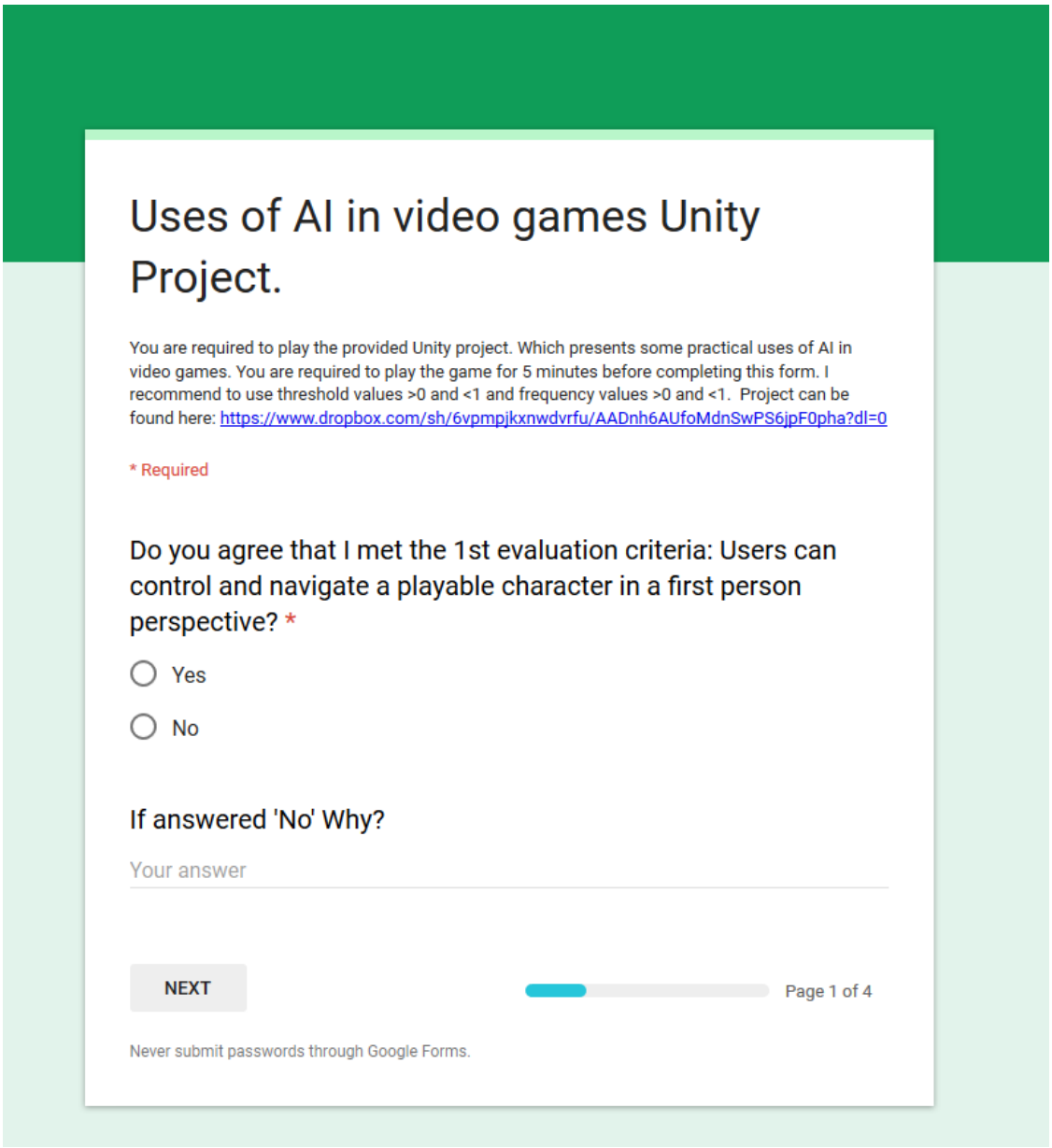

Figure 2: The evaluation survey I created using google forms when evaluating my product

(2)

Survey:Source.

.5 Appendix E-Weekly Logs

Kane Thompson-Hall Progress Log Week of Jan 23rd: Supervisor: Kate Devlin

I did not meet with my supervisor this week due to special circumstances. However I have started the planning stages of the project and have begun working on the preliminary report by starting the introduction section.

I designed a preliminary Gannt chart using Google Sheets which will be my project plan throughout the year. The Gantt chart is subject to change but I outlined steps I will be taking to complete this project. *Began some implementation in Unity adding prefabs and creating tiles on the floor Research: I researched the use of noise in procedurally generated games and I got some ideas in how I am going to implement noise into my project when it comes to generating content. By implementing simplex noise and having the values dictate the tile that is generated.

Kane Thompson-Hall Progress Log Week End of Feb 3rd 2017 Supervisor: Kate Devlin

Meeting 1 Minutes I met with my supervisor this week and we discussed my project proposal and the future of the project and what I should be doing for the next week. Firstly, we discussed about the target audience, in my proposal I mentioned that one of my target audiences is 'children'. However, after discussing this with my supervisor we concluded that I if my target audience was children I would need to make special allowances when it comes to testing and working with children can cause ethical complications. We also discussed the development cycle, in my proposal I mentioned I will be following a waterfall development model but I will reconsider this since the development model of a game is not necessarily sequential like waterfall model and more iterative like the spiral model. I was also advised to look over the project guide to get a better understanding of what I am going to include in my final and preliminary report, and to look at examples of past project reports. Finally, the conclusion of the meeting was regarding the literature review. We discussed what I should include in the literature review and to check google scholars for existing papers on what experts are saying about the game I want to make / algorithms I want to use and discuss them. In addition, I need

to do existing market research regarding the game I want to make and observe what's in the market both commercially developed and academically developed. I also need to research algorithms that I am going include in the project such as Perlin noise and FSM. In summary some key points of our discussion and what needs to be done for next week are: • Do not aim at children due to ethical reasons. • Rethink about waterfall development cycle. • Go over project guide and existing project reports. • Begin literature review use google scholars, library etc • Research FSM, Perlin Noise, Pathfinding algorithms.

Progress Log: This week I started my preliminary report I completed my Introduction, and Motivation sections and I also made a start to the aims and objectives section. I done a small amount of implementation and implemented an algorithm that generates tiles in unity.

Kane Thompson-Hall Progress Log Week End of Feb 10th 2017 Supervisor: Kate Devlin

My supervisor was unavailable this week. However, I have acted on the feedback from my supervisor last week. In that I have changed my target audience, to not be targeted at young children as it could cause ethical problems working with children when it comes to testing my game. I've also gone over the project guide and I have a better understanding on what to include in my preliminary report. This week I have finished a substantial amount of my preliminary report. I have acted on the feedback last week and done a large amount of my literature review. I have completed research on existing market on both commercial and academic projects. In addition, I have completed research on Procedural content generation, perlin noise, and finite state machines and their use in the current industry. I am currently in the process of completing my project report and I am documenting what I have done this term and what I have planned for next term. In conjunction, I am working on the Gantt chart for the project plan for my final project. For next week, I plan to do more implementation of my project, such as completing the perlin noise algorithm and generate empty rooms, to be able to display it for an upcoming presentation. I will also be working on completing my report and hopefully can

hand it in early. In summary I will be working on the following for next week: • Completing my preliminary report • Decide on project development cycle plan • Do small amount of implementation to at least can display concept idea

Kane Thompson-Hall Progress Log Week End of Feb 17th 2017 Supervisor: Kate Devlin

This week was reading week. This week I finalized my Preliminary Project Report in which I discussed the current progress of the project and what is planned next. I also finalized my project plan which was something my supervisor discussed with me a couple weeks ago, as you can see it follows a waterfall development like structure.

This sheet is hosted on google sheets which can be found here. Source Referring to the Gannt chart this is currently the week of 13/02/2017 and I am track with my project plan. This week I finalized my preliminary project report and I continued my project implementation, and next week following the project plan I intend to focus on a substantial amount of implementation to prepare for an upcoming presentation. This week I faced some challenges with implementing connecting rooms but I managed to overcome that by creating a passage from the edges of each room. More over due to these challenges the extra implementation time next week will also allow me to catch up on my progress regarding implementation. I also got some ideas as to how I am going to implement certain Ai and PCG techniques from some of the research I conducted from my literature review in my preliminary project report. In summary next week I plan to focus on my implementation in order to prepare for an informal presentation in one of the labs and to catch up with my progress on the implementation which may have fell behind this week due to some unforeseen challenges.

Kane Thompson-Hall Progress Log Week End of Feb 24th 2017 Supervisor: Kate Devlin My supervisor was unavailable this week due to disability leave however I am continuing the development beyond the initial feedback. This week was the week following reading week.in which I primarily focused on the implementation. Following the challenges I faced last week this week I began working on the noise generation to have a Pseudo random generator when it came to generating levels, to

have some control over how the levels look. In addition, I have started development on the player, and the IA on how they navigate the room. I have decided to implement a navigation graph in which I create waypoints for each tile on the grid to be a moveable space. In addition, next week following my development plan, I plan to further develop the IA by implementing a path finding algorithm in addition to a functional navigational graph system in which each tile represents a node in the graph.

Also, following my development plan I am going to begin and hopefully complete my introduction next week to my final project report in addition to continuing the implantation on the system. So, in summary next week I plan to: • Continue Implementation on the system working on navigational graph and the IA pathfinding methods. • Attempt to finish my introduction to my final project report.

Kane Thompson-Hall Progress Log Week End of March 3rd 2017 Supervisor: Kate Devlin

I met with my supervisor this week and we discussed my preliminary report that I handed in. We discussed the feedback and what I should do and include in my final project report. One of the first points that was pointed out is in my introduction for my final report I should give more of an Overview of what my game is about and how it will play. Separately, I was also advised to avoid using Wikipedia as sources in my Preliminary report and to follow up the references to where Wikipedia got their sources from or expand beyond Wikipedia and reference that instead. Moreover, I was informed that for my Literature review I should include more background research and explaining certain terminology what it was like when they first entered Computer Science and what they are like now, E.g. how IA where first used in games compared to how they are used now. In addition, I also need to expand and do background research for more features of my project such as path finding algorithms. I was advised to look at other past project guides to get an idea on how to structure my Literature review for my final project report. To summarise the minutes of the meeting: • Make amendments to Introduction and aims and objectives. • Don't use Wikipedia for sources and follow information

up to where Wikipedia got their information from. • Expand on my literature review, include research on how certain aspects of my project was used when they were first introduced compare them to now. • Look at other project guides to get a better understanding. • Use google scholars and REMEMBER I can use college login to gain access to some papers that charge fees.

As you can see from the following plan this week I worked on the introduction to my final project report and the implementation. In terms of the introduction I have downloaded Texmaker Latex and have formed a structure for my report in addition to doing the introduction sections. In terms of implementation I have fully implemented a user controlled character whose objective is to navigate to the goal. I've also almost completed a navigational graph which is populated after the room is procedurally generated. Next week I plan to finish off my navigational graph system and begin work on implementing FSM of the IA.

Kane Thompson-Hall Progress Log Week End of March 10th 2017 Supervisor: Kate Devlin

I met with my supervisor this week and I presented the current state of my project and what I am currently working on. I was advised on what I should focus on and made an agreement with my supervisor on what I will deliver next week. I plan on finishing my draft literature review to hand to my supervisor for feedback. In summary • I presented the current state of my project to my supervisor. • Discussed what I plan on implementing next. • What I am struggling with and how I am going to overcome them issues. • Agreed on me finishing my Literature review to hand in next week. The remaining of this week and early next week before my next meeting with my supervisor I plan to finish my literature review and complete as much of my draft computing project report as possible. This week I managed to complete some of my background research in my literature review mainly focusing on Intelligent Agents Finite-State Machines, Before the week is over and leaning in to early next week I plan to complete my Literature review and go deeper in my draft report. In addition to working on my draft report I have been continuing my implementation for my project. This week I have successfully

implemented A star algorithm in the IA which is using a the procedurally generated search space as a navigational graph to navigate the area. I have also implemented minor state changes such as the player 'dying' if the IA catches the player. This week I also encountered a few problems in the implementation, I am currently trying to procedurally generate a continued search space rather than the current static space. I am also currently trying to ensure all the nodes in the navigation graph have the appropriate edges as currently every node has an edge connected to all other nodes. Next week I plan on continuing my implementation fixing some of the errors that I have encountered this week in addition to making my project for presentable by getting assets from unity asset store, and making it run faster. I will also focus more on the report to hand in to my supervisor.

Next week is the week of the 13/03/2017 I plan on completing my literature review and start work on the methods section. My project isn't in a good state to test yet so I may postpone that until next week or complete as much testing as I can for the current progress of the report.

Kane Thompson-Hall Progress Log Week End of March 17th 2017 Supervisor: Kate Devlin

This week I have spoken to my supervisor via email and she will be away for a couple of weeks. However last meeting I agreed with her that I will send in my draft final project report of what I have completed so far, this week so I can receive feedback on it. I will continue to work on my draft final project report up until the draft report deadline at the end of march. This week I have done a substantial amount of work surrounding my project report I have been completing my literature review in my final project report and have begun the design/method section, and will soon begin testing once I have implemented the complete project in a basic form. In addition, regarding implementation I have been working on setting the edges of each node that is generated the appropriate adjacent edges.

In terms of my project plan I should be working on testing now but I still have some implementation to build to have my project in a testing state. Therefore, I am going to need to delay testing and evaluation and complete it as soon as possible

preferably before 31st of March and keep on with the implementation and the rest of the final report. For next week and the remainder of this week I will continue to work on the design section of my final report and implementation of the node edges and improve the map generation to get my project ready for a state of testing and evaluation. I hope to begin my testing and evaluation soon.

Kane Thompson-Hall Progress Log Week End of March 24th 2017 Supervisor: Kate Devlin

Last week I successfully handed in some of my draft final project report consisting of my almost complete literature review to my supervisor for feedback. In which I made the appropriate changes following the initial feedback I received regarding my preliminary report. This week I have been preparing my final project report draft for the deadline of the 31st of March. I have been working on additional sections I never planned to implement such as a system requirements section. I've also begun working on my testing section, I have tested several basic features that match my functionality requirements. In regards to the implementation I have finally solved the problem with connecting the tiles to their appropriate edges. I did this by only applying edges at a certain distance with regards to each tiles vector position.

In terms of my project plan I am a little bit behind and I am supposed to be working on my evaluation and testing for the finalisation of the final project report draft. However, I am still working on testing and hope to begin some evaluation of certain sections next week ready in time to conclude the draft. In addition, I will also continue working on implementation to meet all the functionality requirements criteria.

Kane Thompson-Hall Progress Log Week End of March 31st 2017 Supervisor: Kate Devlin

This week was the final week before submission of the draft final project report on learn . gold and I have been working primarily on the report. To get it ready for submission I still need to work on the Testing and evaluation sections which I have fallen behind schedule. There have been sections that I never included in the original project plan that I decided to add to the final project report, such as System

requirements and implementation. This has slowed my progress and the reason why I have been slightly behind schedule. However, this week I begun working on doing minor black box testing for the project. I In terms of implementation this week I have been working on optimizing the generation of obstacles in the environment using noise values. To produce a procedurally generated environment with obstacles such as trees in a pseudorandom way. I have been working on a formula using the noise method to produce various patterns of obstacles in the environment with a user defined frequency.

Following the plan I should be working on the final introduction for the next two weeks however my introduction is mostly finished therefore I will be working on my testing and evaluation and continuing my implementation of the project.

.6 Appendix F-Full Implementation

Game Manager Class

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour {

    public Dungeon DPrefab;
    //private to hold the instance (we do this to instantiate a maze in
        begin game and destroy it in restart game before we begin a new
        game)
    private Dungeon DInstance;

    // Use this for initialization
    void Start () {
        //Begingame method called at the start.
        print("Test");
        BeginGame();
    }

   // Update is called once per frame
   void Update () {
        //If space key is pressed restart game/ regenerate maze
        if (Input.GetKeyDown(KeyCode.Space) || Dungeon.complete==true)
        {
            Dungeon.complete = false;
```

```
            SceneManager.LoadScene("Menu");
            RestartGame();
        }



    }

    private void BeginGame()
    {
        //Instantiate maze when we begin game
        DInstance = Instantiate(DPrefab) as Dungeon;
        StartCoroutine(DInstance.Generate());
    }

    private void RestartGame()
    {
        StopAllCoroutines();
        //destroy maze and begin game again.
        Destroy(DInstance.gameObject);
        BeginGame();
    }


}
```

Dungeon Class

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

public class Dungeon : MonoBehaviour
{

    //a variable to create a delay and will be used for generating the
        grid.
    public float stepDelay;
    public Tile tilePrefab;
    private Tile[,] tiles;
    public Coordinates Vector;
    public static Dictionary<Coordinates,Tile> FullDung = new
        Dictionary<Coordinates,Tile>();
    public int NumberOfRooms;
    public static Coordinates CD = new Coordinates(0, 0);
    public static int tileCount = 0;
    public static int RoomNo=0;
```

```
public static int RoomArea;
//reference wall instance
public Wall wPrefab;
public Wall wPrefab2;
public Wall wPrefab3;
public Wall wPrefab4;
private List<Tile> allTiles = new List<Tile>();
private List<Tile> newList = new List<Tile>();
public static int interval=0;
public Player playerPrefab;
public Enemy enemy1Prefab;
public Enemy enemy2Prefab;
public Enemy enemy3Prefab;
public Enemy enemy4Prefab;
public static Coordinates playerSpawn = new Coordinates(2, 2);
public static List<GameObject> freeTiles = new List<GameObject>();
public static List<Wall> wallLit = new List<Wall>();
public static Wall[,] trees;
public Goal goalPrefab;
public static int Level = 1;
public static bool complete = false;
public static int numberOfEnemies;
private List<Coordinates> plotMap = new List<Coordinates>();
public static float threshold;
public static float frequency;
public PowerUp powerPrefab;
public static bool VR;
public static int Score=0;
public static int Multiplier=0;
public int spawnG=0;

public IEnumerator Generate()
{
    //instantitate a generation delay which is a delay for each tile
        generated.
    WaitForSeconds generationDelay = new WaitForSeconds(stepDelay);
    //call cleangame function to clear all gameobjects
    cleanGame();
    //Room area = area of the search space
    RoomArea = Vector.x * Vector.z ;
    //plot the map with the coordinates for each tile.
    for(int i=CD.x;i<Vector.x+CD.x;i++)
    {
        for(int j=CD.z;j<Vector.z+CD.z;j++)
        {
            Coordinates newXZ = new Coordinates(i, j);
            plotMap.Add(newXZ);
        }
```

```
}
//add new tile to the frontier
freeTiles.Add(CreateTile(CD).gameObject);
FullDung.Remove(new Coordinates(CD.x, CD.z));
//while coordinates still exist in the plotMap list call
    GenerateLevel function to render tiles per interval
while (plotMap.Count>0)
{
    yield return generationDelay;
    GenerateLevel(freeTiles);
    //when all tiles are generated
    if (freeTiles.Count-1 == RoomArea)
    {
          //If level generated is too populated then return to
              Menu
          if(wallLit.Count<15 ||
              wallLit.Count>(freeTiles.Count/2))
           {
              SceneManager.LoadScene("Menu");
           }
          //Spawn Player in
           Coordinates playerSpawn = new Coordinates(CD.x+5,
               CD.z+5);
           SpawnPlayerIn(playerSpawn);
           //If VR is False Destroy VR Object
           if (VR == false)
           {
               Destroy(GameObject.Find("Player/VRSimulatorCameraRig"));
           }
           //Else Destroy the players head object consisting of
               the player arms, head and camera
           else
           {
               Destroy(GameObject.Find("Player/Head"));
           }
           //Spawn enemyNo number of enemies at random locations
               in the level
           for (int enemyNo = 1; enemyNo <= numberOfEnemies;
               enemyNo++)
           {
               Coordinates enemySpawn = new
                   Coordinates(Random.Range(CD.x, Vector.x),
                   Random.Range(CD.z, Vector.z));
               SpawnEnemyIn(enemySpawn, enemyNo);
           }
           //Spawn a goal to return to
           Coordinates goalSpawn = new Coordinates((Vector.x +
               CD.x) - playerSpawn.x, (Vector.z + CD.z) -
```

```
                playerSpawn.z);
            SpawnGoalIn(goalSpawn);
            break;

        }
    }
}


private void Update()
{
    //spawn a new power-up object if Power-up object has been
        collected / cannot be found.
    if(GameObject.Find("Power-Up")==null)
    {
        Coordinates PowerSpawn = new Coordinates(Random.Range(0,
            Vector.x), Random.Range(0, Vector.z));
        SpawnPowerUps(PowerSpawn);
    }
}



private void GenerateLevel(List<GameObject> tiles)
{
        //select tile at end of list
        int Index = tiles.Count - 1;
        //current tile is assigned to tile at end of list
        GameObject current = tiles[Index];

    //Create a Column of tiles
    for (int i = 0; i < Vector.z; i++)
    {
        //if plotMap contains coordinates
        if (plotMap.Count != 0)
        {
            //assign c to coordinates at end of plot map list
            Coordinates c = plotMap[plotMap.Count - 1];
            //remove last element in plot map list
            plotMap.RemoveAt(plotMap.Count - 1);

            //if coordinates are within the search space (Vector.x
                and Vector.z max height and width of the search space)
            if (coordinatesCheck(c))
            {
                    //assign next tile to a new game obejct and create
                        tile at current coordinates
                    GameObject nextTile = CreateTile(c).gameObject;
```

```
                    //add nextTile to list of gameobject tiles to be
                        used as Waypoint later.
                    tiles.Add(nextTile);
                }
            }
        }
    }

    //Function to render tile at coordinates of input c
    private Tile CreateTile(Coordinates c)
    {
        Tile t = Instantiate(tilePrefab) as Tile;
        t.xz = c;
       //assign a name to the tile
        t.name = "Tile:[" + c.x + "," + c.z + "]";
        t.transform.parent = transform;
        //set the new tile at the corresponding position
        Vector3 point = new Vector3(c.x - Vector.x * 0.5f + 0.5f, 0f, c.z
            - Vector.z * 0.5f + 0.5f);
        //render prefab at the location of the point Vecor
        t.transform.localPosition = point;
        //Count tiles
        tileCount++;
        //Add cell coordinates and tile to dictionary
        FullDung.Add(c, t);
        //Add current tile
        //freeTiles.Add(t.gameObject);
        //Call The Noise function
        Nfunction method = Noise.noiseDimensions[1];
        //The noise method takes the declared point and frequency as an
            input then returns a value between -1 and 1 if that value is
            greater than the user
        //determined threshold then build a wall
        if (method(point, (frequency)) > threshold)
          {
            //build wall at location c
              buildWall(c);
          }
        //return as Tile component
        return t;
    }

    //Destory tile object at coordinates c
    private void DestroyTile(Coordinates c)
    {
        //Destroy tile gameobject
        Destroy(Gettile(c).gameObject);
        //add the tile to all tiles list
```

```
    allTiles.Add(Gettile(c));
    //remove it from available tiles.
    freeTiles.Remove(Gettile(c).gameObject);

}
//function to spawn the player in at input coordinates c
private Player SpawnPlayerIn(Coordinates c)
{
    //instantiate the player model prefab
    Player player = Instantiate(playerPrefab) as Player;
    //assign player coordinates to input coordinates
    player.xz = c;
    //Assign the name of the player game object
    player.name = "Player";
    player.transform.parent = transform;
    //set the vector 3 position of the player.
    player.transform.localPosition = new Vector3(c.x - Vector.x *
        0.5f + 0.5f, 10f, c.z - Vector.z * 0.5f + 0.5f);
    return player;
}

//SpawnENemy function this function will create an IA at coordinates
    c
private Enemy SpawnEnemyIn(Coordinates c,int enemyNo)
{
    //generate number to determine which type of IA to spawn
    int spawnEnemy = Random.Range(0, 2);
    Enemy enemy=null;
    if (spawnEnemy == 0)
    {
        enemy = Instantiate(enemy1Prefab) as Enemy;
    }
    if (spawnEnemy == 1)
    {
        enemy = Instantiate(enemy2Prefab) as Enemy;
    }
    //assign agent coordinates to the input cooordinates
    enemy.xz = c;
    //Assign name to agent
    enemy.name = "Enemy"+enemyNo;
    enemy.transform.parent = transform;
    //set the location of the IA to spawn
    enemy.transform.localPosition = new Vector3(c.x - Vector.x * 0.5f
        + 0.5f, 1f, c.z - Vector.z * 0.5f + 0.5f);
    return enemy;
}
//Spawn goal in at input coordinates c
private Goal SpawnGoalIn(Coordinates c)
```

```
{
    Goal goal = Instantiate(goalPrefab) as Goal;
    //increment level +1 each time a new goal is spawned.
    Level++;
    goal.xz = c;
    //Assign name to goal
    goal.name = "Goal";
    goal.transform.parent = transform;
    //set the new goal at the corresponding position
    goal.transform.localPosition = new Vector3(c.x - Vector.x * 0.5f
        + 0.5f, 1f, c.z - Vector.z * 0.5f + 0.5f);
    return goal;
}

//function that spawns a power-up at coordinates c
private PowerUp SpawnPowerUps(Coordinates c)
{
    PowerUp pow = Instantiate(powerPrefab) as PowerUp;
    pow.xz = c;
    //assign name to power-up
    pow.name = "Power-Up";
    pow.transform.parent = transform;
    //set the vector 3 position of the power-up
    pow.transform.localPosition = new Vector3(c.x - Vector.x * 0.5f +
        0.5f, 1f, c.z - Vector.z * 0.5f + 0.5f);
    return pow;
}

//Construct new obstacle
private Wall buildWall(Coordinates c)
{
    //Randomly generate a number to determine which prefab to render
        to have variety in what obstacles are spawned
    int randomObstacleNo = Random.Range(0, 4);
    Wall w = null;
    if (randomObstacleNo == 0)
    {
        w = Instantiate(wPrefab) as Wall;
    }
    if (randomObstacleNo == 1)
    {
        w = Instantiate(wPrefab2) as Wall;
    }
    if (randomObstacleNo == 2)
    {
        w = Instantiate(wPrefab3) as Wall;
    }
    if (randomObstacleNo == 3)
```

```
    {
        w = Instantiate(wPrefab4) as Wall;
    }

    w.xz = c;
    //assign name of obstacle and the coordinates location
    w.name = "Obstacle:[" + c.x + "," + c.z + "]";
    w.transform.parent = transform;

    //set obstacle position
    w.transform.localPosition = new Vector3(c.x - Vector.x * 0.5f +
        0.5f, 0f, c.z - Vector.z * 0.5f + 0.5f);
    //some objects are not perfecly alligned with tiles so adjust
        certain prefabs when rendered to centere them in the middle
        of a tile.
    if(randomObstacleNo==0)
    {
        w.transform.Translate(1f, 0, 0.75f);
    }

    if(randomObstacleNo==1)
    {
        w.transform.Translate(0.7f, 0, 0.8f);
    }
    // remove the tile that occupies the cell the obstacle was placed
        on from the free tiles list.
    freeTiles.Remove(Gettile(c).gameObject);
    wallLit.Add(w);
    //return as wall component
    return w;
}

//method to check if coordinates are in the generated search space;
public bool coordinatesCheck(Coordinates c)
{
    return c.x >= CD.x && c.x < Vector.x+CD.x && c.z >= CD.z && c.z <
        Vector.z+CD.z;
}

//return the tile located at coordinates c
public static Tile Gettile(Coordinates c)
{
    if (FullDung.ContainsKey(c))
    {
        return FullDung[c];
    }
    else
    {
```

```
            return null;
        }
    }

    //return obstacle at coordinates location c
    public static Wall GetWall(Coordinates c)
    {
        return trees[c.x, c.z];
    }


    //function to clean game and wipe all gameobjects when a new level
        needs creating or player dies.
    public void cleanGame()
    {
        freeTiles.Clear();
        allTiles.Clear();
        newList.Clear();
        FullDung.Clear();
        wallLit.Clear();
        GameObject e1 = GameObject.Find("Enemy1");
        GameObject e2 = GameObject.Find("Enemy2");
        GameObject e3 = GameObject.Find("Enemy3");
        GameObject e4 = GameObject.Find("Enemy4");
        GameObject p = GameObject.Find("Player");
        GameObject g = GameObject.Find("Goal");
        Destroy(e1);
        Destroy(p);
        Destroy(g);
        if(e2!=null)
        {
            Destroy(e2);
        }
        if (e3 != null)
        {
            Destroy(e3);
        }
        if (e4 != null)
        {
            Destroy(e4);
        }
        Tile.interval = -1;
        Tile.changeDirection = 1;
        Tile.roomSize = 1000;

    }
```

```
}
```

Enemy Class

```
using UnityEngine;
using System.Collections;

public class Enemy : MonoBehaviour {
    public Coordinates xz;
    public GameObject p;
    public float speed;
    public static Coordinates current;
    //Enumerations for States of Enemy
    enum State { Roam, Dead, Flee, Chase }
    RandomIA randomPath;
    AstarIA astarPath;
    FleeIA fleePath;
    State currentState = State.Roam;
    Color[] colAr = new Color[4];
    int colorIndex;

    private void Awake()
    {
        //Assign colours to be used by the material of gameobjects
        colAr[0] = Color.blue;
        colAr[1] = Color.red;
        colAr[2] = Color.yellow;
        colAr[3] = Color.magenta;
        colorIndex = Random.Range(0, 4);
        randomPath = this.gameObject.GetComponent<RandomIA>();
        astarPath = this.gameObject.GetComponent<AstarIA>();
        fleePath = this.gameObject.GetComponent<FleeIA>();
        PathAgent.speed = speed;
    }


    void Update()
    {
        FSM();
    }

    public void FSM()
    {
        //assign player gameobject.
        p = GameObject.Find("Player");
```

```
switch (currentState)
{
    // Dead state will destroy the gameobject this script
       (Enemy.cs) is attatched to.
    case State.Dead:
        Dungeon.Multiplier++;
        Dungeon.Score += 10*Dungeon.Multiplier;
        Destroy(this.gameObject);
        break;
    // Flee state will calculate a path to get away from the
       player.
    case State.Flee:
        //Assign all fleeing Enemys gameobjects's material to the
           white colour to represent the change in state.
        this.gameObject.GetComponentInChildren<Renderer>().material.color
           = Color.white;
        //If the gameobject is fleeing and is caught by the
           player then the state will change to dead.
        if (Vector3.Distance(p.transform.position,
           this.gameObject.transform.position) <= 2.8)
        {
            currentState = State.Dead;

        }
        //If player is not powered up anymore change state to
           Roam.
        if (Player.Powered == false)
        {
            currentState = State.Roam;
        }
        //If the game obejct was previous using the RandomIA
           script then remove it because Enemy is no longer
           chasing player.
        if (randomPath.enabled == true)
        {
            randomPath.enabled = false;
        }
        //If the game obejct was previous using the AstarIA
           script then remove it because Enemy is no longer
           chasing player.
        if (astarPath.enabled == true)
        {
            astarPath.enabled = false;
        }
```

```
        if (fleePath.enabled==false)
        {
            fleePath.enabled = true;
        }
        break;
    //Roam state will calculate random paths around the level
        until it sees the player (comes within distance of the
        player).
    case State.Roam:
        //Assign an appropriate colour to the game object this
            script is attatched to (Enemy.cs).
        if
            (this.gameObject.GetComponentInChildren<Renderer>().material.color
            != colAr[colorIndex])
        {
            this.gameObject.GetComponentInChildren<Renderer>().material.color
                = colAr[colorIndex];
        }
        //If the game obejct was previous using the AstarIA
            script then remove it because Enemy is no longer
            chasing player.
        if (astarPath.enabled == true)
        {
            astarPath.enabled = false;
        }
        //If the game object is not currently using the RandomIA
            script as a random path finding method then apply it
            as a component to this gameobject.
        if (randomPath.enabled==false)
        {
            randomPath.enabled = true;
        }
        //If gameobject sees the player then change state to
            chase.
        if (Vector3.Distance(p.transform.position,
            this.gameObject.transform.position) <= 15)
        {
            currentState = State.Chase;
        }
        //If the player is touching the enemy then destroy the
            player.
        if (Vector3.Distance(p.transform.position,
            this.gameObject.transform.position) <= 2.8)
        {
            Dungeon.Score = 0;
            Dungeon.Multiplier = 0;
            print("Final Score" + Dungeon.Score);
            Destroy(p);
```

```
            Dungeon.complete = true;
        }
        if (fleePath.enabled==true)
        {
            fleePath.enabled = false;
        }
        //If the player is powered up then change state to Flee.
        if (Player.Powered == true)
        {
            currentState = State.Flee;
        }

        break;
    case State.Chase:

        //If the game obejct was previous using the RandomIA
            script then remove it because Enemy is no longer
            chasing player.
        if (randomPath.enabled==true)
        {
            randomPath.enabled = false;
        }
        //If the game object does not have the component of the
            AstarIA script then apply it to calculate path to the
            player.
        if (astarPath.enabled==false)
        {
            astarPath.enabled =true;
        }
        //Destroy player if the Enemy (this.gameobject) catches
            player.
        if (Vector3.Distance(p.transform.position,
            this.gameObject.transform.position) <= 2.8)
        {
            Dungeon.Score = 0;
            Dungeon.Multiplier = 0;
            Destroy(p);
            Dungeon.complete = true;
        }

        //change state back to roam if the player is no longer in
            sight.
        if (Vector3.Distance(p.transform.position,
            this.gameObject.transform.position) >= 25)
        {
            currentState = State.Roam;
        }
        //change state to flee if player is powered-up.
```

```
                if (Player.Powered == true)
                {
                    currentState = State.Flee;
                }
                if (fleePath.enabled == true)
                {
                    fleePath.enabled = false;
                }
                break;

        }
        Vector3ToCoordinates();

    }



    //Convet vector3 vector position to Coordinates
    public Coordinates Vector3ToCoordinates()
    {
        Coordinates output = new
            Coordinates((int)this.gameObject.transform.position.x + 11,
            (int)this.gameObject.transform.position.z + 16);
        current = output;
        return output;
    }

}
```

Player Class

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
public class Player : MonoBehaviour {

    public Coordinates xz;
    public GameObject sight;
    public GameObject head;
    public float horizontalMovement = 2;
    public float verticalMovement = 2;
    //Enumerations for States of Enemy
    enum State { Alive,Dead,PoweredUp}
    State currentState = State.Alive;
    static bool[,] visited = new bool[500, 500];
    public static List<int> playerPath = new List<int>();
    public static Coordinates current;
    public static bool Powered;
```

```
Direction currentDirection;
void Update()
{

    //Move Up
    if(Input.GetKeyDown(KeyCode.UpArrow) ||
        Input.GetKeyDown(KeyCode.W))
    {
        Coordinates Up = new Coordinates(0, 1);
        xz += Up;
       // print("X= "+ xz.x+ "Z= "+ xz.z);
       // player.transform.Translate(0, 0, Up.z);

        currentDirection = Direction.North;
    }
    //Move Right
    if (Input.GetKeyDown(KeyCode.RightArrow) ||
        Input.GetKeyDown(KeyCode.D))
    {
        Coordinates Right = new Coordinates(1, 0);
        xz += Right;
     //   print("X= " + xz.x + "Z= " + xz.z);
      // player.transform.Translate(Right.x, 0, 0);
        currentDirection = Direction.East;
    }
    //Move Down
    if (Input.GetKeyDown(KeyCode.DownArrow) ||
        Input.GetKeyDown(KeyCode.S))
    {
        Coordinates Down = new Coordinates(0, -1);
        xz += Down;
    //    print("X= " + xz.x + "Z= " + xz.z);
       // player.transform.Translate(0, 0, Down.z);
        currentDirection = Direction.South;
    }
    //Move Left
    if (Input.GetKeyDown(KeyCode.LeftArrow) ||
        Input.GetKeyDown(KeyCode.A))
    {
        Coordinates Left = new Coordinates(-1, 0);
        xz += Left;
      //  print("X= " + xz.x + "Z= " + xz.z);
       // player.transform.Translate(Left.x, 0, 0);
        currentDirection = Direction.West;
    }
    //print("Function Test: " + "X= " + Vector3ToCoordinates().x +
        "Z= " + Vector3ToCoordinates().z);
    // direction enumerators dictating what direction the player
```

```
            moves in
        switch (currentDirection)
        {
            case Direction.North:
                this.gameObject.transform.Translate(0, 0, 0.1f);
                break;
            case Direction.East:
                this.gameObject.transform.Translate(0.1f, 0, 0);
                break;
            case Direction.South:
                this.gameObject.transform.Translate(0, 0, -0.1f);
                break;
            case Direction.West:
                this.gameObject.transform.Translate(-0.1f, 0, 0);
                break;
        }


        playerFSM();
        //print(Dungeon.Score);
       //Assign mouse speed
        float h= horizontalMovement * Input.GetAxis("Mouse X");
        float v = verticalMovement * Input.GetAxis("Mouse Y");
        head = GameObject.Find("Player/Head");
        //Set the sight rotation to match mouse movement
        if (head != null)
        {
            head.transform.Rotate(0, Input.GetAxis("Mouse X") * 8, 0);
        }

        visitingNodes();
    }

    public void playerFSM()
    {
        foundPowerUp();
        switch (currentState)
        {   //Dead state destroy the player object and reset the game
            back to level 1
            case State.Dead:
                Destroy(this.gameObject);
                Dungeon.complete = true;
                break;
            //Powered up state set powered to true with is used as a
                static variable in the Enemy.cs script
            case State.PoweredUp:
                Powered = true;
                //Invoke the setAliveState() method to trigger in x
```

```
                seconds which is used as a time limit for how long
            Invoke("setAliveState", 10);
           // print("Power");
            break;
        //Alive state which sets powered to false which controls
            certain behaviours of the Enemy gameobjects
        case State.Alive:
            Powered = false;
            break;

    }


}


//Keeps a log of most up to date
public void visitingNodes()
{
    playerPath.Insert(0,Dungeon.freeTiles.IndexOf(Dungeon.Gettile(Vector3ToCoordinates()
    if(playerPath.Count>30)
    {
        playerPath.RemoveAt(playerPath.Count - 1);
    }

}
//Convet vector3 vector position to Coordinates
public Coordinates Vector3ToCoordinates()
{

    Coordinates output = new
        Coordinates((int)this.gameObject.transform.position.x+11,
        (int)this.gameObject.transform.position.z+16);
    current = output;
    return output;
}

public void cleanWaypoints()
{
    Dungeon.freeTiles.Clear();
}


public bool foundPowerUp()
{
    GameObject u;
    //If power-up has been destroyed (collected by player)
    if (GameObject.Find("Power-Up") != null)
```

```
        {
            u = GameObject.Find("Power-Up");
            if (Vector3.Distance(this.gameObject.transform.position,
                u.transform.position) <= 2.8)
            {
                //change state of player to powered up and destroy the
                    power-up gameobject
                currentState = State.PoweredUp;
                Destroy(u);
                return true;
            }
        }
        return false;

    }


    //set state to alive.
    public void setAliveState()
    {
        currentState = State.Alive;
    }

}
```

Power-Up Class

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

public class PowerUp : MonoBehaviour {
    public Coordinates xz;




}
```

Goal Class

```
using UnityEngine;
using System.Collections;

public class Goal : MonoBehaviour {

    public Coordinates xz;
```

```
    void Update()
    {
        foundGoal();

    }

    //check if player is near goal
    public static bool foundGoal()
    {
        GameObject p = GameObject.Find("Player");
        GameObject g = GameObject.Find("Goal");
        if (Vector3.Distance(p.transform.position, g.transform.position)
            <= 2.8)
        {
            Destroy(g);
            Dungeon.complete = true;
            return true;
        }
        else
        {
            return false;
        }

    }




}
```

Coordinates Class

```
using UnityEngine;
using System.Collections;
[System.Serializable]
//Struct to manipulate two integers as one
public struct Coordinates
{
    //define int variables for x and z coordinates.
    public int x;
    public int z;

    public Coordinates(int x, int z)
    {
        this.x = x;
        this.z = z;
```

```
    }

    //operator to add coordinates
    public static Coordinates operator +(Coordinates V1, Coordinates V2)
    {
        V1.x += V2.x;
        V1.z += V2.z;
        return V1;
    }
}
```

Direction Class

```
using UnityEngine;
using System.Collections;

public enum Direction
{
    //4 Directions
    North, East, South, West



}
```

DirectionProperties Class

```
using UnityEngine;
using System.Collections;

public static class DirectionProperties
{
    public const int numOfDirections = 4;

    //array of directions
    private static Direction[] opposite =
    {
        Direction.South,
        Direction.West,
        Direction.North,
        Direction.East
    };
    //array of rotations
    private static Quaternion[] rotate =
    {
        Quaternion.identity,
        Quaternion.Euler(0f, 90f, 0f),
        Quaternion.Euler(0f, 180f, 0f),
```

```
        Quaternion.Euler(0f, 270f, 0f)
    };


    //get opposite direction to input
    public static Direction getOpposite(this Direction d)
    {
        return opposite[(int)d];
    }
    //get rotation
    public static Quaternion getRotation(this Direction d)
    {
        return rotate[(int)d];
    }

    //the numerical direction of each direction
    private static Coordinates[] directionalVectors =
    {
        new Coordinates(0,1),
        new Coordinates(1,0),
        new Coordinates(0,-1),
        new Coordinates(-1,0)
    };
    //return random direction from:North,East,South,West
    public static Direction RandomDirection
    {
        get
        {
            return (Direction)Random.Range(0, numOfDirections);
        }
    }



    //method to convert directional vector to its coordinates
    public static Coordinates toCoordinates(this Direction d)
    {
        return directionalVectors[(int)d];
    }



}
```

Adjacency List Graph Class

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
public class AdjacencyListGraph : Graph {

    Dictionary<int, Dictionary<int, float>> d = new Dictionary<int,
        Dictionary<int, float>>();
    public AdjacencyListGraph()
    {

    }

    bool Graph.addNode(int a)
    {
        if (!d.ContainsKey(a))
        {
            Dictionary<int, float> neighCost = new Dictionary<int,
                float>();
            d.Add(a, neighCost);
            return true;
        }
        else
        {
            return false;
        }
    }

    bool Graph.addEdge(int a, int b, float cost)
    {
        if (d.ContainsKey(a) & d.ContainsKey(b))
        {

            d[a].Add(b, cost);
            return true;
        }
        else
        {
            return false;
        }
    }

    List<int> Graph.nodes()
    {

        List<int> nodes = new List<int>(this.d.Keys);

        return nodes;
    }
```

```
    List<int> Graph.neighbours(int a)
    {
        if (d.ContainsKey(a))
        {
            List<int> neighbours = new List<int>(d[a].Keys);
            return neighbours;
        }
        else
        {
            return null;
        }
    }


    float Graph.cost(int a, int b)
    {
        if (d.ContainsKey(a))
        {
            float cost = d[a][b];
            return cost;
        }
        else
        {
            return 0;
        }

    }
}
```

Waypoint Graph Class

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//Implementation from Games AI Coursework
public class WaypointGraph
{

    public Graph navGraph;
    public static List<GameObject> waypoints;
    public static int[,] arrayPoints= new
        int[Dungeon.freeTiles.Count,Dungeon.freeTiles.Count];
    public static Dictionary<GameObject, int> TileNode = new
        Dictionary<GameObject, int>();
    public GameObject this[int i]
    {
        get { return waypoints[i]; }
```

```
    set { waypoints[i] = value; }
}

public WaypointGraph(GameObject waypointSet)
{
    //assign waypoints to new list
    waypoints = new List<GameObject>();
    //assign navGraph to new AdjacencyL
    navGraph = new AdjacencyListGraph();

    findWaypoints(waypointSet);
    buildGraph();
}

private void findWaypoints(GameObject waypointSett)
{
    GameObject player = GameObject.Find("Player");
   // waypoints.Add(player);
   //use Dungeon.freeTiles a list of all game object tiles and add
       them as waypoints to the waypoint list
    if (Dungeon.freeTiles.Count != null)
    {
        for(int i=0;i < Dungeon.freeTiles.Count;i++)
        {

            waypoints.Add(Dungeon.freeTiles[i]);


        }
        Debug.Log(Dungeon.freeTiles.Count+Dungeon.wallLit.Count);
        // Debug.Log("Found " + waypoints.Count + " waypoints.");

    }
    else
    {
        //Debug.Log("No waypoints found.");

    }
}

private void buildGraph()
{

    int n = waypoints.Count;

    navGraph = new AdjacencyListGraph();
    //add all nodes to to the navgraph
    for (int i = 0; i < n; i++)
```

```
        {
            navGraph.addNode(i);

        }


        for (int i = 0; i < n; i++)
        {
            for (int j = 1; j < n; j++)
            {
                //calculate distance cost between nodes
                Vector3 wPos1 = waypoints[i].transform.position;
                Vector3 wPos2 = waypoints[j].transform.position;
                float cost = Vector3.Distance(wPos1, wPos2);
                if (cost <= 1.5)
                {
                    navGraph.addEdge(i, j, cost);
                    //Debug.Log(cost);
                }

               // waypoints.Find(arrayPoints[i, j]);

            }
        }
    }

    //find nearest node from current vector position
    public int? returnNearest(Vector3 here)
    {
        int Start =
            Dungeon.freeTiles.IndexOf(Dungeon.Gettile(Enemy.current).gameObject);
        Debug.Log("Node:" + Start);
        return Start;

    }


}
```

PathAgent Class

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Text;
//Implementation from Games AI Coursework
public abstract class PathAgent : MonoBehaviour
{
```

```
// Set from inspector
public GameObject waypointSet;
public GameObject IA;
// Waypoints
public static WaypointGraph waypoints;
protected int? current;

protected List<int> path;
protected Pathfinder pathfinder;

public static float speed;
protected static float NEARBY = 0.2f;
protected static System.Random rnd = new System.Random();
public abstract Pathfinder createPathfinder();
```

```
void Start()
{
    IA = GameObject.Find("Enemy1");
    waypoints = new WaypointGraph(waypointSet);
    waypointSet = GameObject.Find("Dungeon(Clone)");
    path = new List<int>();
    pathfinder = createPathfinder();
    pathfinder.navGraph = waypoints.navGraph;
}

void Update()
{

    if (path.Count == 0)
    {
        // We don't know where to go next
        generateNewPath();

    }
    else
    {
        // Get the next waypoint position
        GameObject next = waypoints[path[0]];
        Vector3 there = next.transform.position;
        Vector3 here = transform.position;

        // Are we there yet?
        float distance = Vector3.Distance(here, there);
```

```
            if (distance < NEARBY)
            {
                // We're here
                current = path[0];
                path.RemoveAt(0);
              //  Debug.Log("Arrived at waypoint " + current);
            }
        }
    }


    //state that adds a new node to path
    IEnumerator newNode()
    {
        Debug.Log("check");
        Random r = new Random();
        int newr = Random.Range(0, 5);
        path.Add(newr);
        yield return 0;
    }



    void FixedUpdate()
    {
        if (path.Count > 0)
        {
            GameObject next = waypoints[path[0]];
            Vector3 position = next.transform.position;
            transform.position = Vector3.MoveTowards(transform.position,
                position, speed);
        }
    }

    protected void generateNewPath()
    {

        if (current != null)
        {
            // We know where are
            List<int> nodes = waypoints.navGraph.nodes();

            if (nodes.Count > 0)
            {
                // Pick a random node to aim for
                int target = nodes[playerTarget()];
              //  Debug.Log("New target: " + target);
                // Find a path from here to there
```

```
            path = pathfinder.findPath(current.Value, target);
            // Debug.Log( playerTarget());
            // Debug.Log();

        }
        else
        {
            // There are zero nodes
           Debug.Log("No waypoints - can't select new target");
        }

    }
    else
    {
        // We don't know where we are

        // Find the nearest waypoint
        int? target = waypoints.returnNearest(transform.position);

        if (target != null)
        {
            // Go to target

            path.Clear();
            path.Add(target.Value);
          //  Debug.Log("Heading for nearest waypoint: " + target);
        }
        else
        {
            ///Couldn't find a waypoint
         //  Debug.Log("Can't find nearby waypoint to target" +
             "target is: " + target);
        }

    }
}

public static string writePath(List<int> path)
{
    var s = new StringBuilder();
    bool first = true;
    foreach (int t in path)
    {
        if (first)
        {
            first = false;
        }
        else
```

```
            {
                s.Append(", ");
            }
            s.Append(t);
        }
        return s.ToString();
    }

    //assign target to the current node the player is occupying
    public static int playerTarget()
    {
        int Target =
            Dungeon.freeTiles.IndexOf(Dungeon.Gettile(Player.current).gameObject);

        return Target;
    }


    public static int playerStart()
    {
        int Start =
            Dungeon.freeTiles.IndexOf(Dungeon.Gettile(Enemy.current).gameObject);

        return Start;
    }
}
```

Path Finder Class

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//Implementation from Games AI Coursework
public abstract class Pathfinder
{
    public Graph navGraph;
    public abstract List<int> findPath(int a, int b);
}
```

AStar IA Class

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

```
public class AstarIA : PathAgent
{
    public static float heuristicCost(int a, int b)
    {
        //set gameobjects assigned to waypoints in the waypoints array.
        GameObject wayPoint1 = waypoints[a];
        GameObject wayPoint2 = waypoints[b];
        //create vector3 for two positions of the 2 waypoints
        Vector3 pos1 = wayPoint1.transform.position;
        Vector3 pos2 = wayPoint2.transform.position;
        // calculate the distance to be used as a heuristic
        float dx = (float)Math.Abs(pos1.x - pos2.x);
        float dy = (float)Math.Abs(pos1.y - pos2.y);
        return (float)Math.Sqrt(dx * dx + dy * dy);
    }

    public override Pathfinder createPathfinder()
    {
        //create a heuristic that will be calculated from two positions
        Heuristic cost = new Heuristic(heuristicCost);
        //return new AstarPathfinder with the heuristic 'cost'
        return new AStarPathfinder(cost);
    }


}

public delegate float Heuristic(int a, int b);

public class AStarPathfinder : Pathfinder
{
    protected Heuristic guessCost;

    public AStarPathfinder(Heuristic h)
    {
        guessCost = h;
    }

    public override List<int> findPath(int start, int goal)
    {
        //create Simple priority queue and Dictionary
        SimplePriorityQueue<int> frontier = new
            SimplePriorityQueue<int>();
        Dictionary<int, int> visitedFrom = new Dictionary<int, int>();
        Dictionary<float, float> costSoFar = new Dictionary<float,
            float>();
```

```
//Create a stack to pop nodes back in to the list.
Stack<int> nextDestination = new Stack<int>();
List<int> l1 = new List<int>();
List<int> l2 = new List<int>();
List<int> path = new List<int>();
//add starting node with cost of 0 to frontier
frontier.Enqueue(start, 0);
Debug.Log(start);
//set visited from [start] to -1
visitedFrom[start] = -1;
//cost so far = 0;
costSoFar[start] = 0;

while (frontier.Count > 0)
{
    //get the node at the head of the queue / lowest priority
    int current = frontier.Dequeue();
    //if head of queue = goal then simplePriority queue complete.
    if (current == goal)
    {
        break;
    }
    //create list neighbours of current node
    List<int> neighbours = navGraph.neighbours(current);
    foreach (int next in neighbours)
    {
        //calculate the cost accumalted so far.
        float nextCost = costSoFar[current] +
            navGraph.cost(current, next);
        //If no neighbours cost have been compared or check if
            new lower cost path has been found
        if (!costSoFar.ContainsKey(next) || nextCost <
            costSoFar[next])
        {
            //   Debug.Log("Next " + next + " Current: " +
                current);
            //add next node to priority queue with a cost of next
                cost + the heuristic guess cost
            frontier.Enqueue(next, nextCost + guessCost(next,
                goal));
            visitedFrom[next] = current;
            //cost so far is added to the dictionary
            costSoFar[next] = nextCost;
            l1.Add(next);
            l2.Add(visitedFrom[next]);
        }
    }
}
```

```
        //push target goal to stack
        nextDestination.Push(goal);
        for (int destination = goal; destination != start; destination =
            visitedFrom[destination])

        {
            //keep pushing all the nodes visited in a dictionary
            nextDestination.Push(visitedFrom[destination]);
        }
        for (int i = nextDestination.Count; i > 0; i--)
        {
            //while stack is greater than 0 keep popping stack

            path.Add(nextDestination.Pop());
        }
        //return the path
        return path;
    }

}
```

Random IA Class

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//Implementation from Games AI Coursework
public class RandomIA : PathAgent
{
    public override Pathfinder createPathfinder()
    {
        return new RandomPathfinder();
    }

}

public class RandomPathfinder : Pathfinder
{
    public override List<int> findPath(int start, int goal)
    {
        //path of nodes
        List<int> path = new List<int>();
        //add starting node
        path.Add(start);
        //initialise random object rnd
        System.Random rnd = new System.Random();
        while (path.Count < 2)
        {
```

```
            //assign next node to a random number from 0 to number of
                nodes in the search space
            int nextNode = rnd.Next(0, Dungeon.freeTiles.Count);
            //add next node
            path.Add(nextNode);
        }


        return path;
    }
}
```

Flee IA Class

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

public class FleeIA : PathAgent {


        public override Pathfinder createPathfinder()
        {
            return new FleePathfinder();
        }



    public class FleePathfinder : Pathfinder
    {
        public override List<int> findPath(int start, int goal)
        {
            //path of nodes
            List<int> path = new List<int>();
            int [] corner = new int[4];
            corner[0] = 0;
            corner[1] = 49;
            corner[2] = Dungeon.RoomArea - 1;
            corner[3] = corner[2] - 50;
            start = Random.Range(0,corner.Length);
            //add starting node
            path.Add(start);

            //initialise random object rnd
            System.Random rnd = new System.Random();
            while (path.Count < 25)
            {
```

```
                //assign next node to a random number from 0 to number of
                    nodes in the search space
                int nextNode = rnd.Next(0, Dungeon.freeTiles.Count);
                if (!Player.playerPath.Contains(nextNode))
                {
                    //add next node
                    path.Add(nextNode);
                }
            }


            return path;
        }
    }
}
```

Tile Class

```
using UnityEngine;
using System.Collections;

public class Tile : MonoBehaviour
{
    public Coordinates xz;
    public static int interval = -1;
    public static int changeDirection = 1;
    public static int roomSize = 1000;
    //return edge of cell


    public Direction nextDirection
    {
        get
        {
            //interval assigned to number of edges


            interval++;
            if(interval%(roomSize-1)==0)
            {
                changeDirection++;
            }
            if(changeDirection==4)
            {
                changeDirection = 0;
            }
            return (Direction)changeDirection;
```

```
            throw new System.InvalidOperationException("Tiles has no
                uninitialized directions left.");
        }
    }

}
```

Wall Class

```
using UnityEngine;
using System.Collections;

public class Wall : MonoBehaviour {
    public Coordinates xz;
}
```

Play Button Class

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System;
using UnityEngine.SceneManagement;
//script for play button
public class PlayButton : MonoBehaviour {

    public Button btnPlay;
    public Text freq;
    public Text threshold;
    public Toggle VR;
    public Text score;
    public Text noEnemies;
    void Start()
    {
        Button btn = btnPlay.GetComponent<Button>();
        btn.onClick.AddListener(TaskOnClick);
        //update score
        score.text += Dungeon.Score;
    }

    //on play button click assign field values to static variables
    void TaskOnClick()
    {
        Dungeon.frequency = float.Parse(freq.text);
        Dungeon.threshold = float.Parse(threshold.text);
        Dungeon.VR = VR.isOn;
        Dungeon.numberOfEnemies = int.Parse(noEnemies.text);
```

```
        SceneManager.LoadScene("FinalProject");
    }


}
```

Graph Class

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
public interface Graph
{

    bool addNode(int a);        // true if node added
    bool addEdge(int a, int b, float cost); // true if edge added

    List<int> nodes();
    float cost(int a, int b);
    List<int> neighbours(int a);

}
```

Enemy Class

```
using UnityEngine;
using System.Collections;

public class Enemy : MonoBehaviour {
    public Coordinates xz;
    public GameObject p;
    public float speed;
    public static Coordinates current;
    //Enumerations for States of Enemy
    enum State { Roam, Dead, Flee, Chase }
    RandomIA randomPath;
    AstarIA astarPath;
    FleeIA fleePath;
    State currentState = State.Roam;
    Color[] colAr = new Color[4];
    int colorIndex;

    private void Awake()
    {
        //Assign colours to be used by the material of gameobjects
        colAr[0] = Color.blue;
        colAr[1] = Color.red;
```

```
        colAr[2] = Color.yellow;
        colAr[3] = Color.magenta;
        colorIndex = Random.Range(0, 4);
        randomPath = this.gameObject.GetComponent<RandomIA>();
        astarPath = this.gameObject.GetComponent<AstarIA>();
        fleePath = this.gameObject.GetComponent<FleeIA>();
        PathAgent.speed = speed;
    }


    void Update()
    {
        FSM();
    }

    public void FSM()
    {
        //assign player gameobject.
        p = GameObject.Find("Player");




        switch (currentState)
        {
            // Dead state will destroy the gameobject this script
                (Enemy.cs) is attatched to.
            case State.Dead:
                Dungeon.Multiplier++;
                Dungeon.Score += 10*Dungeon.Multiplier;
                Destroy(this.gameObject);
                break;
            // Flee state will calculate a path to get away from the
                player.
            case State.Flee:
                //Assign all fleeing Enemys gameobjects's material to the
                    white colour to represent the change in state.
                this.gameObject.GetComponentInChildren<Renderer>().material.color
                    = Color.white;
                //If the gameobject is fleeing and is caught by the
                    player then the state will change to dead.
                if (Vector3.Distance(p.transform.position,
                    this.gameObject.transform.position) <= 2.8)
                {
                    currentState = State.Dead;
```

```
        }
        //If player is not powered up anymore change state to
            Roam.
        if (Player.Powered == false)
        {
            currentState = State.Roam;
        }
        //If the game obejct was previous using the RandomIA
            script then remove it because Enemy is no longer
            chasing player.
        if (randomPath.enabled == true)
        {
            randomPath.enabled = false;
        }
        //If the game obejct was previous using the AstarIA
            script then remove it because Enemy is no longer
            chasing player.
        if (astarPath.enabled == true)
        {
            astarPath.enabled = false;
        }
        if (fleePath.enabled==false)
        {
            fleePath.enabled = true;
        }
        break;
    //Roam state will calculate random paths around the level
        until it sees the player (comes within distance of the
        player).
    case State.Roam:
        //Assign an appropriate colour to the game object this
            script is attatched to (Enemy.cs).
        if
            (this.gameObject.GetComponentInChildren<Renderer>().material.color
            != colAr[colorIndex])
        {
            this.gameObject.GetComponentInChildren<Renderer>().material.color
                = colAr[colorIndex];
        }
        //If the game obejct was previous using the AstarIA
            script then remove it because Enemy is no longer
            chasing player.
        if (astarPath.enabled == true)
        {
            astarPath.enabled = false;
        }
        //If the game object is not currently using the RandomIA
            script as a random path finding method then apply it
```

```
            as a component to this gameobject.
        if (randomPath.enabled==false)
        {
            randomPath.enabled = true;
        }
        //If gameobject sees the player then change state to
            chase.
        if (Vector3.Distance(p.transform.position,
            this.gameObject.transform.position) <= 15)
        {
            currentState = State.Chase;
        }
        //If the player is touching the enemy then destroy the
            player.
        if (Vector3.Distance(p.transform.position,
            this.gameObject.transform.position) <= 2.8)
        {
            Dungeon.Score = 0;
            Dungeon.Multiplier = 0;
            print("Final Score" + Dungeon.Score);
            Destroy(p);
            Dungeon.complete = true;
        }
        if (fleePath.enabled==true)
        {
            fleePath.enabled = false;
        }
        //If the player is powered up then change state to Flee.
        if (Player.Powered == true)
        {
            currentState = State.Flee;
        }

        break;
    case State.Chase:

        //If the game obejct was previous using the RandomIA
            script then remove it because Enemy is no longer
            chasing player.
        if (randomPath.enabled==true)
        {
            randomPath.enabled = false;
        }
        //If the game object does not have the component of the
            AstarIA script then apply it to calculate path to the
            player.
        if (astarPath.enabled==false)
        {
```

```
                astarPath.enabled =true;
            }
            //Destroy player if the Enemy (this.gameobject) catches
                player.
            if (Vector3.Distance(p.transform.position,
                this.gameObject.transform.position) <= 2.8)
            {
                Dungeon.Score = 0;
                Dungeon.Multiplier = 0;
                Destroy(p);
                Dungeon.complete = true;
            }

            //change state back to roam if the player is no longer in
                sight.
            if (Vector3.Distance(p.transform.position,
                this.gameObject.transform.position) >= 25)
            {
                currentState = State.Roam;
            }
            //change state to flee if player is powered-up.
            if (Player.Powered == true)
            {
                currentState = State.Flee;
            }
            if (fleePath.enabled == true)
            {
                fleePath.enabled = false;
            }
            break;

    }
    Vector3ToCoordinates();

}



//Convet vector3 vector position to Coordinates
public Coordinates Vector3ToCoordinates()
{
    Coordinates output = new
        Coordinates((int)this.gameObject.transform.position.x + 11,
        (int)this.gameObject.transform.position.z + 16);
    current = output;
    return output;
}
```

```
}
```