

# ANALYSIS OF A RANGE OF OPTIMISATION PATHFINDING ALGORITHMS

Full name: Gagandeep Malhotra

Candidate number: 5380

Centre name: Langley Grammar School

Centre number: 51411

Qualification code: H446

Date: 2020

NEA AQA A-Level Computer Science

## Table of Contents

<b>Analysis</b> .....	3
What is the problem area being investigated? .....	3
How am I solving this problem? .....	3
Background .....	4
Dijkstra’s Algorithm.....	4
A* (A Star) Search.....	4
Breadth-First Search .....	5
Greedy Best First Search.....	6
Who is my end-user? .....	6
How was this problem researched?.....	6
Survey .....	7
Interview.....	10
Objectives .....	12
Modelling of pathfinding algorithms .....	18
Dijkstra’s Algorithm.....	18
A* (A Star) Search.....	19
<b>Documented Design</b> .....	20
How does the system work? .....	20
Pseudocode: .....	20
Dijkstra’s Algorithm.....	20
A* (A Star) .....	20
Greedy Best First Search.....	21
Breadth-First Search .....	22
Data Structures.....	22
Lists .....	22
Graphs.....	22
Queues .....	22
Multi-dimensional Array .....	22
Dictionary.....	23
Variables .....	23
File Structure .....	23
Structure diagram .....	24
HCI (Human Computer Interaction) .....	25
Main Menu.....	25
Instructions Page.....	25

Settings Page .....	26
Maze Type Selection.....	27
Preset/Random Maze Menu.....	28
Custom Maze Menu .....	30
Pathfinder Choosing Menu .....	30
Pathfinders are run .....	31
Unity Engine Interface .....	35
<b>Technical Solution:</b> .....	36
Background Colour Code: .....	36
Background Rotation Code: .....	36
Camera Control Code: .....	37
Change Camera Code:.....	41
Demo Controller Code: .....	42
Graph Code: .....	44
Graph View Code:.....	46
Grid Set Code: .....	50
Map Data Code: .....	53
Menu Selector Code: .....	59
Move Background Code: .....	61
Node Code: .....	62
Node View Code: .....	63
On Node Click Code: .....	65
Pathfinder Code: .....	66
Pause Code: .....	78
Priority Queue Code: .....	79
Settings Code: .....	81
Sort Type Code: .....	98
Toggle Instructions Code: .....	98
Toggle Settings Code: .....	100
<b>Testing</b> .....	101
<b>Evaluation</b> .....	105
Completeness of Objectives.....	105
Conclusion.....	114

# Analysis

## What is the problem area being investigated?

Pathfinding algorithms are used for finding the shortest path in satellite navigation systems, artificial intelligence in computer games, and routing data packets over the internet, as well as a variety of other purposes in computer systems. However, the universal problem encountered in each of these applications of pathfinding algorithms is that there is no 'perfect' pathfinder. Pathfinding algorithms that were guaranteed to find the shortest path were typically time and resource-consuming, whereas non-resource-intensive algorithms were unreliable in finding the ideal shortest path.

In order to mitigate the compensation between reliability and performance, computer scientists were able to develop algorithms that combined both the methods used in non-resource demanding pathfinders (Greedy Best-First search) with those utilised in 'optimal' pathfinders (Dijkstra's algorithm). Aiming for a balance between the two extremes, A\* (A Star) search algorithm became commonplace in the gaming industry, being flexible in purpose and proven as the most optimally efficient pathfinder possible. Still, A Star search is still not perfect for every application as the implementation is relatively complex compared to a simple pathfinding algorithm like Breadth-First Search, which utilises solely a queue data structure.

As a consequence of the variety of pathfinding algorithms available, many computer programmers are uncertain on what method to use to best suit the purpose of their program. A basic peer-to-peer network that would need to check every node in the network could simply use breadth-first search, whereas a navigation system with traffic warnings and different terrains would need to use Dijkstra's algorithm to take the cost of a certain path into account. Usually, the 'best' pathfinding algorithm to implement is subjective, but some algorithms are better suited and more optimised for a specific purpose than others.

The problem being investigated in my project is the method in which computers dynamically calculate the shortest path from a single starting point to a destination, and how to decide on the pathfinding algorithm with the most suitable characteristics for a particular program.

## How am I solving this problem?

My Pathfinding Algorithms project will investigate, as well as compare, the benefits and drawbacks of the more prevalent algorithms used in computer pathfinding. I have chosen to use Dijkstra's Algorithm, Breadth-First Search, A\*(A Star) Search Algorithm, and Greedy Best-First in this investigation. These algorithms will be run through either a preset maze, random maze, or a custom-maze, which can be altered by the user. This program aims to clearly demonstrate the variety of methods these pathfinding algorithms use as well as

clearly illustrate their function as they traverse through a graph of nodes, which will highlight the discrepancies between each pathfinding algorithm.

## Background

Pathfinding algorithms simply plot, close to, or the shortest route possible between two points and is used by computer applications. However, there are many different methods devised to do so, including Dijkstra's Algorithm, Breadth-First Search, A\*(A Star) Search Algorithm, and Greedy Best-First, which are all used in my project. They all have different code and methods in which they devise a shortest path. Here is a detailed summary of each pathfinder:

### Dijkstra's Algorithm

Dijkstra's Algorithm computes the cost of the shortest path from a starting node to all other nodes in a graph, however, it can be stopped when the destination is reached to give the shortest path to that node in particular. This pathfinding algorithm is commonly used in mapping road networks, IP routing, and telephone networks. First devised by its creator Edsger Dijkstra in 1956, and since then, has become one of the most popular pathfinding algorithms. There are several different implementations of Dijkstra's Algorithm (using a Fibonacci heap), but the one in this program uses a binary heap, which is a data structure in the form of a binary tree that is used for priority queues.

#### Pros:

- Guaranteed to find shortest possible path
- Simple to implement
- Takes cost of node into account

#### Cons:

- Explores every node so time taken is lengthy
- Cannot take negative weighted nodes into account (in finance)

#### Complexity:

- $O(n^2)$  where 'n' is number of nodes

### A\* (A Star) Search

A\* Search is frequently used in programs because of it being exceptionally efficient in searching a graph of nodes whilst also finding a near optimal solution each time. This pathfinding algorithm is used abundantly in video games for artificial intelligence, due to its efficiency. Computer scientists in Stanford Research Institute published the algorithm in 1968, as a modified version of Dijkstra's Algorithm, and originally designed it as a generic graph traversal

algorithm. A\* Search can be seen as a combination between Greedy Best First and Dijkstra's Algorithm as it approximates a path between the start and goal node, by using the heuristic  $f(n) = g(n) + h(n)$ , where  $g(n)$  is exact distance to reach the goal node from the start node, and  $h(n)$  is an estimation of distance from the goal node, to quickly find the shortest path. If the search goes too deep in a path where the heuristic function value is too high, then  $g(n)$  will pull the search algorithm's path back to a more promising path to find the shortest route.

**Pros:**

- Approximate (heuristic) makes for a short computation time
- Heuristic can be altered to prioritise speed or reliability by weighting the  $g(n)$  and  $h(n)$  value.
- Takes cost of node into account
- Is complete, so will always find a solution if it exists

**Cons:**

- Uses a sizeable amount of memory
- Worst case scenario offers no advantage over Dijkstra's algorithm

## Breadth-First Search

Breadth-First Search (BFS) traverses from the start node and explores all of the neighbour nodes at the current depth prior to moving to the next set of neighbours. The modern version of this pathfinding algorithm was invented in 1959 by Edward Moore, and has since been used in social media friend suggestion algorithms, GPS navigation systems, and peer-to-peer computer networks.

**Pros:**

- Easy to implement
- Guaranteed to find the shortest path if all nodes on the graph are traversed

**Cons:**

- Time taken is lengthy as BFS can only travel level by level
- Does not take cost of node into account
- Has to search every node on the graph

**Complexity:**

- $O(n)$  where 'n' is number of nodes

## Greedy Best First Search

Greedy Best First Search expands the node which is closest to the goal node and continues this pattern until the goal is reached. Uses the heuristic  $f(n) = h(n)$  where  $h(n)$  is distance from goal. It is similar to Dijkstra's algorithm, however, it is not regularly used in computer programs due to it being unreliable in finding the shortest path, albeit fast.

### Pros:

-Very quick time to find path between two nodes if there are no obstacles in the way

-Simple to implement

### Cons:

-Not guaranteed to find the shortest possible path

-An obstacle/wall blocking the goal will greatly increase the time taken

-Does not take the cost of a node into account

## Who is my end-user?

**Contact: [matthewlang@lgs.slough.sch.uk](mailto:matthewlang@lgs.slough.sch.uk)**

My end-user is a further maths teacher at my school who teaches pathfinding algorithms, such as Dijkstra, to students annually. I have had a detailed interview with him initially to obtain a deeper understanding of the objectives he would like for me to meet in my project as well as what features my project should contain so it could be of use to him as a teaching/learning tool for him and his students. Throughout the course of this project I have consulted with him for any additional advice or changes I would need to add to my project for it to be more useful.

My project is also suitable to be used as a teaching as well learning tool for computer science students. Pathfinders such as Dijkstra's algorithm are part of the further maths OCR MEI specification and is also an important topic in computing. Since my project will demonstrate detailed instructions and information on each pathfinding algorithm (including the data structures used), I believe many students will greatly benefit from seeing algorithms, such as Dijkstra's and Breadth-First Search, run through their own user-created maze so they can grasp a better understanding of the function and purpose of particular pathfinders.

## How was this problem researched?

In order to learn more about the problems involved in the usage of each pathfinder and what the most commonplace pathfinding algorithms were, I

consulted various sources of information to gather information. These sources included websites, interviews, prototyping, and a survey carried out personally.

The websites I found to be both accurate and detailed in their information on this subject matter were (*Patel, A., 2020. Amit's A\* Pages*) and (*A\* Search Algorithm - GeeksforGeeks, 2020*) which are included among the list of websites I have used in the references section of this document. Another website which I greatly gained understanding of the different pathfinders from was (*PathFinding.js, 2020*) which allow users to create their own start and goal positions and run a large list of pathfinding algorithms on.

I have used prototyping during the process of my project as initially I had made a single preset maze which ran all of the four primary pathfinding algorithms I had decided on in conjunction with my end-user. I also have ensured that all algorithms were working correctly on the initial graph of nodes and had followed the characteristics of the algorithms according to the numerous website sources I have used to gather information from. The use of prototyping during the creation of my project was so that I could focus on the main objective and functions in my list of objectives before completing supplementary goals of my program.

## Survey

To aid in the research of my pathfinding project, I had created a paper-based survey which I had passed along to classmates and teachers in order to gather data and information on what my primary objectives should be. Opportunity sampling was used in order to choose the 20 people who completed my survey. My survey took roughly six minutes to complete and included questions which required both qualitative and quantitative answers. I had given each person a short summary on what the subject matter of my project is and what it means. I have also ensured that all questions asked were not leading questions and to not show my personal bias.

### Question 1)

**Should my project should contain a guide on how each pathfinder works?**

**Yes/No**

Out of 20 people, 16 people chose '**Yes**' whilst 4 people chose '**No**'.

This told me that an overwhelming majority of people would like to see an instruction guide on how each pathfinder functions.

### Question 2)

**What is the reason for your answer to Question 1?**

.....



Most reasons stated for **'Yes'** were that they had very basic or no existing knowledge of the pathfinders used in my program, so as a user they would like help on how to use my program and what it was doing. Another reason given was that they would like a guide so they knew that I was informed on the function of my program and fully understood the algorithms I was using.

The reason given for **'No'** was that they believed that a guide was not vital to my program's use and assumed that anyone using my program would already have knowledge on the subject matter at hand (Pathfinding Algorithms).

After reviewing the majority **'Yes'** responses to the first question and the very valid reasons given, I had added an instruction page as an objective to meet in my program.

### Question 3)

**On a scale of 1-5 (5 being the most important), how important is it to include different variations of the same pathfinding algorithm in my project, e.g. D\* (D Star) as a variation of A\* (A Star)?**

**1      2      3      4      5**

Out of 20 people: 9 people chose **'2'**, 5 people chose **'3'**, 3 people chose **'1'**, 3 people chose **'4'**, and no one chose **'5'**.

From these rankings, I was able to determine that the majority of people did not see the inclusive of lesser known pathfinding algorithms, such as D\*, as being an important feature to my program. Therefore, I did not include variations of Dijkstra, A Star, Breadth-First, and Greedy Best First in my list of objectives. Furthermore, the lesser-known algorithms are not as popular as the four mentioned above and are used less frequently in programs.

### Question 4)

**What information would you want to know from a pathfinding algorithm that has been run?**

.....

The most common answer to this question was **'speed'**, which was to be expected. Other different answers given were how many different **'nodes'** the pathfinder explored, and how many lines of code the algorithm was.

As a result of this information, I have added to the list of objectives to get a value for **'speed'** which is calculated from the values for **'distance'** and **'time'** I get from the pathfinder ( $\text{speed} = \text{distance}/\text{time}$ ). In addition to this, number of nodes explored is also an objective in this program, however, I have not opted to include **'lines of code an algorithm takes'** because it is not at all a reliable indicator of how efficient or fast a pathfinding algorithm is.

**Question 5)**

**Should my project include nodes of a different cost (nodes of different distance values)?**

**Yes/No**

Out of 20 people, 14 people chose **'Yes'** and 6 people chose **'No'**.

This told me that the majority of people would like to see the option to include nodes of a different cost in the graph.

**Question 6)**

**What is the reason for your answer to Question 5?**

.....

Most reasons stated for the answer **'Yes'** were that the option to include different weighted nodes would allow my project to have more uses in the real world. Another popular reason given was that this feature would allow the user to see the benefits of Dijkstra's algorithm and A\* (A Star) search, which takes the cost of node into account, whereas Breadth First Search and Greedy Best First does not.

The only reason for **'No'** that was given was that including nodes of a different cost is unnecessary to the main functionality of my program.

Due to this information, I have decided to include having nodes of a different cost in my list of objectives. I entirely agree with the people who voted **'Yes'**, I also believe that it is very important to highlight the fact that Dijkstra's algorithm and A\* search takes the cost of node into account, unlike Breadth First Search and Greedy Best First. I disagree with the reason given for **'No'**, because one of the aims of my program is to clearly show users the difference in function between algorithms. Omitting the ability for the user to see if the pathfinding algorithm can or cannot perform a task, such as taking cost of node into consideration, is an important piece of information to demonstrate in my program.

**Question 7)**

**Would you rather have the graph of nodes displayed in a Two-Dimensional or Three-Dimensional view?**

.....

Out of 20 people, 9 responded with **'Two-Dimensional'** and 11 people responded with **'Three-Dimensional'**.

This information told me that there was demand for both a three dimensional and two-dimensional camera view as the results were almost evenly split. The benefits of a 2D view is that it is easier for a user to navigate and see what is

happening on the whole graph of nodes clearly, whilst the benefits of 3D view is that it is more visually appealing for the user and adds depth to each graph making the pathfinder more 'realistic' to the user. From these results, I have decided to include having the option of both a '**Three-Dimensional**' and '**Two-Dimensional**' view for the user in my list of objectives.

### Question 8)

**Would a square graph with same width and height dimensions (3x3, 5x5, 25x25) or a rectangle graph of any width and height dimensions (3x6, 5x5, 35x24) be better suited to my project?**

.....  
Out of 20 people, 15 people responded with '**Any Dimensions**' and only 5 people responded with '**Fixed Dimensions**'.

From this information I concluded that the user should be able to input the size of their maze in '**Any Dimensions**' because that would allow my program to be more flexible in different scenarios that the user may want to model where the graph is not a square. The only drawback to making the graph take any dimension is that the user could input a width of 3 and height of 100, which would create a peculiar looking maze that may be difficult to show clearly on one screen using a 2D or 3D camera. As a result of this, I have added the user being able to make a maze of any dimensions to my list of objectives, as well as a zoom in, zoom out, and drag feature to my camera.

## Interview

Before I began my project, my interview was carried out with a further maths teacher at my school (Mr. Lang) who is familiar with pathfinding algorithms.

Contact: [matthewlang@lgs.slough.sch.uk](mailto:matthewlang@lgs.slough.sch.uk)

### Question 1: Which pathfinding algorithms should I include in my project?

From this question, he was able to tell me that 'Dijkstra's algorithm' and mentioned 'A Star pathfinding' because those were the most 'popular' pathfinding algorithms that he was aware of, so I made sure to include these two in my list of objectives. 'Prim's' (a variation of Dijkstra) and 'Kruskal's' algorithm were also brought up, as they were taught in the further maths course. However, these two algorithms are for finding a minimum spanning tree for a graph, and although they could be adapted into a shortest-path finding algorithm, I felt they were not prevalent or distinct enough from Dijkstra and A Star to be included in the list of objectives. With further discussion, he came up with 'Breadth first Search' which is a very common algorithm used in computing and for that reason I had also included it in my list of objectives.

**Question 2: What data would you like to see from a pathfinding algorithm that has been run?**

In our discussion regarding this question, he suggested that 'time taken', 'number of nodes travelled', and 'distance travelled', as important information to be recorded. 'Time taken' is significant because how quickly algorithm completes its task is a large factor in its consideration to be used. Similarly, 'number of nodes travelled' shows how efficient the algorithm is in getting between two points and travelling the least number of nodes, where a lower number of nodes travelled means less memory is taken up, and is therefore less taxing on the computer system. 'Distance travelled' is an important indicator to determine whether the algorithm is working correctly to determine close to, or indeed the shortest path, which is the ultimate aim of these pathfinding algorithms. Using this information discussed in the survey, I have decided to add all of these as objectives and data to record in my program. In addition to this, I have also added the functionality of recording the 'speed' of each pathfinding algorithm by dividing the 'distance travelled' by 'time taken'. I have decided to add 'speed' to show how 'fast' the algorithm travels through nodes in the graph.

**Question 3: How would you like the data that was recorded to be displayed for the different pathfinding algorithms?**

Feedback from this question included discussing around the fact that a straight list of the different information recorded could be too unclear and confusing for a user because the abundance of information would be difficult for them to compare and understand. So, to sort this problem that arose, he suggested that a simple table showing all the pathfinding algorithms with their respective, time, distance, and speed. Then these tables could be sorted based on the user's choice and ranked accordingly. As well as this feature, another suggestion was a simple list showing the relevant information for all pathfinders on the side in the order in which they were run. For pathfinders that have not been run yet, a warning message reading 'This pathfinder has not been run yet' will display on the table so the user is aware of what pathfinding algorithms they have already run and which ones they have not. All these suggestions have been added to my list of objectives for this program.

**Question 4: How important is saving the results that a user has recorded for their pathfinding algorithms?**

In our discussion, he stated that saving the results of a user's previous pathfinding algorithm results was 'extremely important'. The reasons given for this were so that a user did not have to run every algorithm again every time they wanted the results for a particular scenario. This will make the program more user-friendly, and provide more incentive for users to use this program as both a teaching and learning tool. The method in which to save these results were also discussed, where the ideas of saving it internally in the app, externally

on excel, or perhaps a simple text file would be suitable. Saving the results inside the application would allow it to be accessed again reliably, however, it would be difficult to export out of the program if the user would want to use this data in another form. Using these facts, I have decided to externally save the results of the pathfinders of a maze into a text file named 'Log.txt' which will display the graph parameters, as well as the results recorded for all the pathfinders. I have decided upon this method because I believe it will be the most convenient and useful for the user as they can export that data into an excel (or any other application) if they choose to, giving more freedom and usability over the program's purpose.

### **Question 5: Do you believe you could use a program with the functionalities of my program as a teaching tool?**

Given the aim of my program, I desired for it to be both suitable as a learning tool in computer science and further mathematics for people who are not too familiar with pathfinding algorithms, so they can learn the purpose and functionality of them through practical use and a visual demonstration, and for complete beginners to pathfinding algorithms. From this question, he replied that if I met all my objectives and ensured that the program was working both correctly and clearly, that it would be very suitable to be a tool used in classrooms that covered the topic of pathfinding algorithms. He suggested that an instructions page be constructed in my program so a user would have no further questions regarding the function or use of a particular pathfinding algorithm as it all should be covered in the instructions page. As a result of this information, I have added a detailed instruction page to my list of objectives for this project.

## Objectives

1. A main menu which the user can navigate through
  - a. A button to begin the making of the maze
  - b. A button to view the instructions of the program
2. An instructions page which can be accessed from the pathfinder and the main menu
  - a. Information on the following topics to be displayed when clicked on:
    - i. About Program
    - ii. Dijkstra's Algorithm
    - iii. Breadth-First Search
    - iv. Greedy Best First
    - v. Controls
  - b. For the main menu instruction page
    - i. Exit button to go back to main menu
  - c. For the pathfinder instruction page
    - i. Exit button to go back to selecting pathfinder
3. A settings page where the user can set the desired values for the maze
  - a. A button to go back to the main menu

- b. A button to confirm the values and go on to the next page
- c. A list of Integer variables with a default value which the user can change (restricted between a certain range) that include:
  - i. Maze width
    - 1. Default: 25
    - 2. Range: 3-100
  - ii. Maze height
    - 1. Default: 25
    - 2. Range: 3-100
  - iii. Starting x-coordinate
    - 1. Default: 0
    - 2. Range: 0-99 below width minus one
  - iv. Start y-coordinate
    - 1. Default: 0
    - 2. Range: 0-99 and below height minus one
  - v. End x-coordinate
    - 1. Default: 24
    - 2. Range: 0-99 and below width minus one
  - vi. End y-coordinate
    - 1. Default: 24
    - 2. Range: 0-99 and below height minus one
- d. A list of Boolean variables which the user can change that include:
  - i. Pathfinder exit on goal reached
    - 1. Default: True
  - ii. Show arrows displaying the previous node where the pathfinder traversed from
    - 1. Default: True
  - iii. Allow the pathfinder to traverse in eight directions
    - 1. Default: True (False means travels in four directions)
  - iv. Show the pathfinder as it traverses through the graph
    - 1. Default: True (False means just the final path is displayed)
- e. A list of Float variables which the user can change that include
  - i. The size of the gap between adjacent nodes (border size)
    - 1. Default: 0.1
    - 2. Range: 0-0.5
  - ii. When show iterations is true, the time interval in seconds between each step in the pathfinder
    - 1. Default: 0.01
    - 2. Range: 0.01-5
- f. A dynamic display that shows what will happen in the case of each Boolean variable being true or false with their descriptions being changed accordingly
- g. An error check for if any value entered is out of range when the confirm button is clicked
- h. An error check for the start and end nodes being the same coordinate as that distance is always zero

- i. A list of all erroneous values that have been inputted incorrectly by the user to be displayed clearly.
    - j. An entry box next to each variable so the user can see the default value as well as their edited one
    - k. Disable the time step entry box if the option to show iterations is set to false.
  4. A menu to select what type of maze type the user would like
    - a. Display all the parameters which the user has entered clearly onto a list
    - b. A button for a preset maze
    - c. A button for a random maze
    - d. A button for a custom maze
    - e. An option to go back to the main menu
    - f. A button to confirm the choice of maze type
    - g. An error check to remind the user if they have not selected an option
  5. A preset maze menu from which the user can select premade graphs
    - a. An accurate picture of the preset maze displayed next to the option to select the respective maze
    - b. Description below each preset maze displaying its width, height, starting x coordinate, starting y coordinate, ending x-coordinate, and ending y-coordinate.
    - c. Premade maze types
      - i. A simple perfect maze
      - ii. A maze with a few small gaps awkwardly placed to test the pathfinder
      - iii. A massive maze with a straight path to the goal as well as a winding path
      - iv. A maze with straight paths to the goal but with different weights on each path
    - d. A button to go back to the previous maze type selection menu
    - e. A button to confirm the choice of preset maze
    - f. An error check to remind the user if they have not selected an option
    - g. A warning that their custom variable values for the width, height, starting x coordinate, starting y coordinate, ending x-coordinate, and ending y-coordinate will not be used in the preset maze
    - h. A scrollable window to select these maze presets from
  6. A random maze menu from which the user can select different types of random mazes
    - a. An example of the random maze displayed next to the option to select the respective maze
    - b. Random maze types
      - i. A dense random maze (33% chance of a wall)
      - ii. A sparse random maze (25% chance of a wall)
      - iii. A very sparse random maze (20% chance of a wall)
    - c. A short description below each random maze displaying the likelihood of a node being a wall.

- d. A button to go back to the previous maze type selection menu
  - e. A button to confirm the choice of random maze
  - f. An error check to remind the user if they have not selected an option
  - g. A scrollable window to select the random maze type from
  - h. A list of parameters set by the user displayed clearly on the screen
7. A custom maze menu where the user can create their own customised graph
- a. Instructions on how to use the custom maze editor
  - b. A key displaying all the node type distances and behaviour
  - c. A palette from which the user can choose their desired node type which includes:
    - i. Blocked (black); cannot be traversed by the pathfinder
    - ii. Open node (white); horizontal/vertical distance of one (default)
    - iii. Light terrain node (light brown): horizontal/vertical distance of two
    - iv. Medium terrain node (brown): horizontal/vertical distance of three
    - v. Heavy terrain node (dark brown): horizontal/vertical distance of four
    - vi. Very heavy terrain node (very dark brown): horizontal/vertical distance of five
  - d. A graph accurate to the width and height of the variable values entered, showing the graph of nodes as a grid of white squares, except for the start node which is green, and the goal node which is red.
    - i. The start and goal nodes cannot be edited so the pathfinder will always have a start and a goal
  - e. Allow the user to zoom in and out the graph (using the scroll wheel), drag across the graph (clicking the scroll wheel), and reset the graph by using the right mouse button
  - f. The user can edit the graph by holding and dragging the left mouse button across the desired nodes in the graph, which will then change to the node type selected in the palette
  - g. A button to go back to the previous maze type selection menu
  - h. A button to confirm and create the custom maze that was drawn
8. A menu from which the desired pathfinder to be run can be selected by the user
- a. A button to go to the instructions page
  - b. A button to confirm the pathfinder selected
  - c. An error check to remind the user if they have not selected an option
  - d. Buttons for each pathfinder in the program that include:
    - i. Dijkstra's Algorithm
    - ii. A\* (A Star)
    - iii. Breadth First Search
    - iv. Greedy Best First

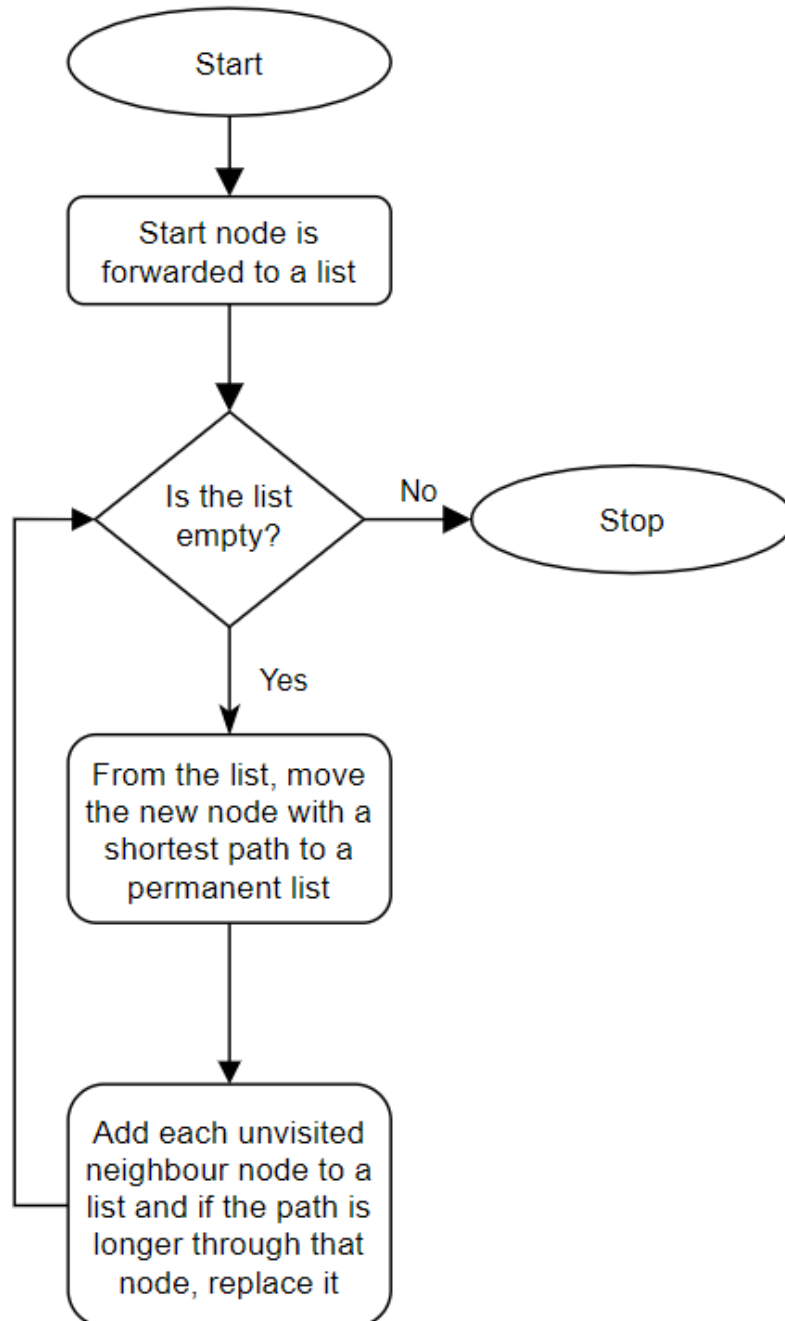


- e. Relevant information with a short summary of each pathfinder to be displayed clearly when the mouse is hovered over the relevant pathfinder
9. The screen for which the pathfinder is carried out on the maze selected by the user
- a. A fully modelled three-dimensional graph that the pathfinder will be carried out on with shadows and dynamic lighting from a light source
  - b. Various buttons to change/view settings:
    - i. 'Toggle parameters' which will display the variable setting that were set by the user
      - 1. A key that displays the meaning of the colours on the graph that include:
        - a. Blocked node (black) which the pathfinder cannot traverse through
        - b. Open node (white) which has horizontal/vertical distance of one
        - c. Explored node (light grey) which shows nodes that the pathfinder has explored
        - d. Neighbour node (light blue) which shows nodes that are neighbouring the explored nodes of the pathfinder (next to be explored by the pathfinder)
        - e. Path nodes (orange) which show the final path determined by the pathfinding algorithm
        - f. Start node (green) which shows the position in the graph where the pathfinder starts from
        - g. Goal node (red) which shows the position in the graph the pathfinding algorithm ends at
        - h. Old neighbour (dark blue) which shows the neighbouring node of the previous pathfinding algorithm that was run
        - i. Old explored (dark grey) which shows the old nodes which the previous pathfinder had explored
      - ii. 'Select another pathfinder' that will allow the user to select another pathfinder to run
      - iii. 'Toggle results' will show the user the time taken, distance travelled, speed, and nodes explored for the pathfinders that have been run, (in order in which they were run) in a scrollable list
        - 1. A button to return back to menu will appear
          - a. A warning displaying that the current maze data will be lost if the pathfinders have not all been run yet
        - 2. A button to toggle the information on the various pathfinders

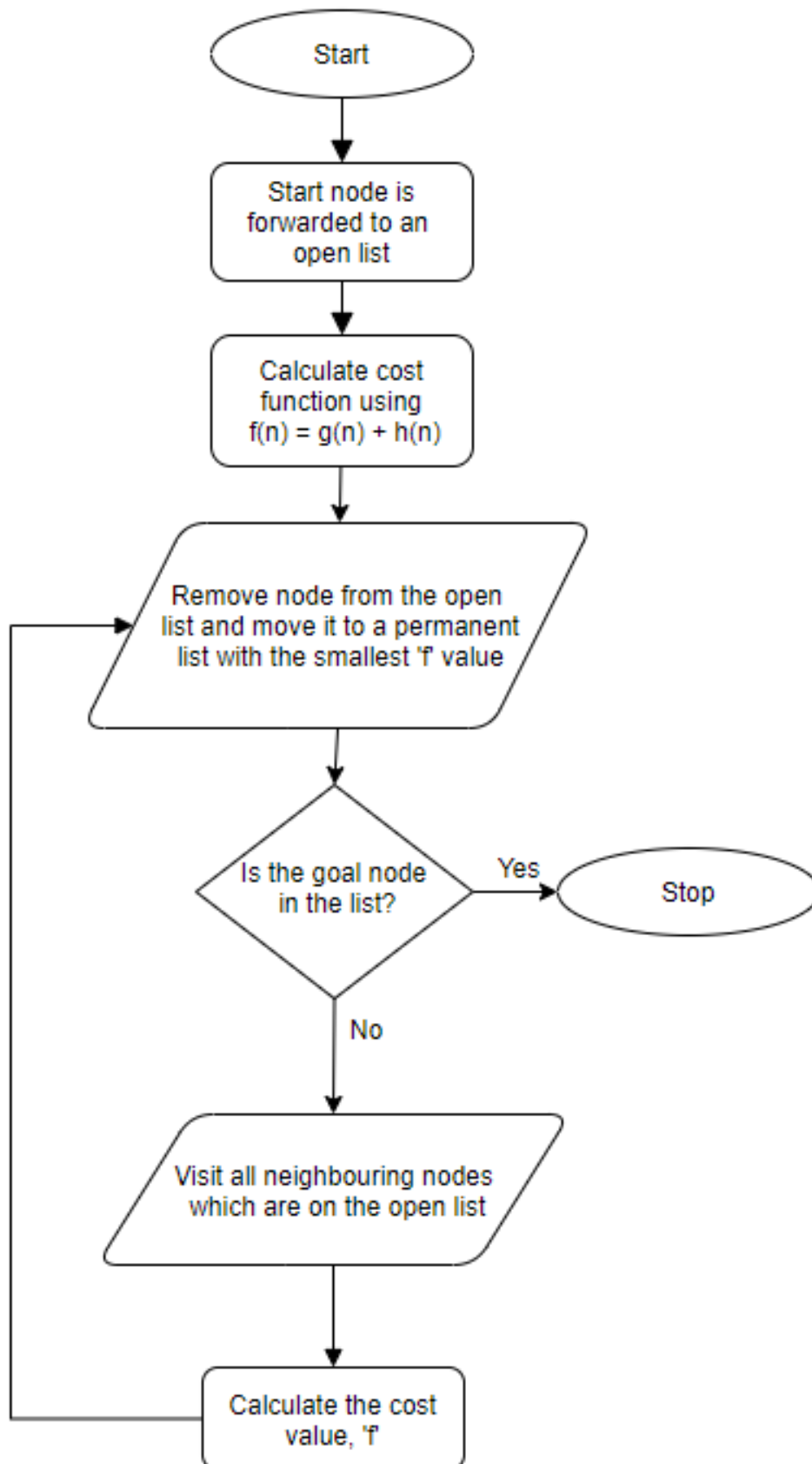
- a. Dynamic table with the ability to sort by distance, time, or speed where the rankings of each algorithm will be shown
  - b. Algorithms that have not been run will display a reminder that they have not been selected yet
  - iv. A 3-d/2-d camera option that will change the view of the graph from a top-down two-dimensional view of the graph, to that of a fully modelled three-dimensional view of the graph, or vice versa
    - 1. Two-dimensional camera will be default when the program is first run for the clearest view of the graph for the user
    - 2. Allow the user to zoom in and out the graph (using the scroll wheel), drag across the graph (clicking the scroll wheel), and reset the graph by using the right mouse button
    - 3. Three-dimensional camera will pivot around the centre point of the graph and rotate horizontally and vertically according to the mouse's movement. Prevent the user from rotating the camera below the horizontal x axis of the graph.
    - 4. The three-dimensional camera can be zoomed in and out using the scroll wheel
  - v. A pause/play button that will pause the timer and the pathfinder while it is being run, whilst also maintaining the functionality of both dimensional camera types
  - vi. A drop-down menu where the user can change various graphical settings that include:
    - 1. Direction of light incident on the graph which will change the appearance of shadows
    - 2. The colour of the background which can be changed by the user from the default grey to any RGB value
  - c. A button to select another pathfinder to carry out on the maze
10. The program will save all relevant information in an external text file when all pathfinding algorithms have been run
- a. Create external text file named 'Log.txt' when all the pathfinding algorithms have been run if a 'Log.txt' file does not already exist
  - b. Record the date and time at which the set of pathfinding algorithms were run
  - c. Record the variable values that were set by the user for the maze in question
  - d. Record all the time taken, distance travelled, speed, and nodes explored into the text file
  - e. Save/Update the text file in 'Log.txt' which will be created next to the program file

## Modelling of pathfinding algorithms

### Dijkstra's Algorithm



## A\* (A Star) Search



# Documented Design

## How does the system work?

To make my program, I have decided to use Unity, a real-time development platform. I have settled upon Unity because of its great capabilities in 3D modelling and I believe that having a 3d display for my maze to showcase the pathfinders is very important to clearly display the function of the different pathfinding algorithms. Another reason for using Unity is because of the capability of designing clear and navigable user interface so people not too familiar with computer programs can still use my project. My entire code has been written in C# and uses Object Oriented Programming.

## Pseudocode:

### Dijkstra's Algorithm

```
1) Create a set X (shortest path tree set) that keeps track of
vertices included in shortest path tree, i.e., whose minimum
distance from source is calculated and finalized. Initially, this
set is empty.

2) Assign a distance value to all vertices in the input graph.
Initialize all distance values as INFINITE. Assign distance value as
0 for the source vertex so that it is picked first.

3) While X doesn't include all vertices
    a) Pick a vertex u which is not there in X and has minimum
distance value.
    b) Include u to X.
    c) Update distance value of all adjacent vertices of u. To
update the distance values, iterate through all adjacent vertices.
For every adjacent vertex v, if sum of distance value of u (from
source) and weight of edge u-v, is less than the distance value of
v, then update the distance value of v.

End
```

### A\* (A Star)

```
1. Initialize the open list
2. Initialize the closed list
   put the starting node on the open
   list (you can leave its f at zero)
3. while the open list is not empty
```

- ```
a) find the node with the least f on
   the open list, call it "q"

b) pop q off the open list

c) generate q's 8 successors and set their
   parents to q

d) for each successor
   i) if successor is the goal, stop search
      successor.g = q.g + distance between
                   successor and q
      successor.h = distance from goal to
                   successor (This can be done using many
                   ways, we will discuss three heuristics-
                   Manhattan, Diagonal and Euclidean
                   Heuristics)

      successor.f = successor.g + successor.h

   ii) if a node with the same position as
        successor is in the OPEN list which has a
        lower f than successor, skip this successor

   iii) if a node with the same position as
        successor is in the CLOSED list which has
        a lower f than successor, skip this successor
        otherwise, add the node to the open list
End (for loop)

e) push q on the closed list
End (while loop)
```

## Greedy Best First Search

- ```
1) Create an empty PriorityQueue
   PriorityQueue X;
2) Insert "start" in X.
   X.insert(start)
3) Until PriorityQueue is empty
   u = PriorityQueue.DeleteMin
   If u is the goal
     Exit
   Else
     Foreach neighbour v of u
       If v "Unvisited"
         Mark v "Visited"
         X.insert(v)
     Mark u "Examined"
```

End

## Breadth-First Search

```
BFS (G, s) //Where G is the graph and s is the source node
    let Q be queue.
    Q.enqueue( s ) //Inserting s in queue until all its neighbour
vertices are marked.

    mark s as visited.
    while (Q is not empty) //Removing that vertex from queue,
whose neighbour will be visited now
        v = Q.dequeue( ) //processing all the neighbours of v
        for all neighbours w of v in Graph G
            if w is not visited
                Q.enqueue( w ) //Stores w in Q to further
visit its neighbour
                mark w as visited.
```

## Data Structures

### Lists

I have used a Lists in my program to store information of the nodes for the pathfinding algorithms which need to check which nodes have been previously visited. The lists will simply store which nodes each separate pathfinder has traversed and store this value into a file. Another reason for using a list is to store all the nodes which are determined to be a wall and therefore not traversable.

### Graphs

I have used a graph to connect all the nodes together so the pathfinding algorithms can determine which nodes are traversable and which are not, based on the user's choice (eight-directions/four-directions). Without a graph, the pathfinders would not be able to function correctly.

### Queues

I have used a queue, specifically priority queue, so pathfinding algorithms such as Dijkstra's and A Star search algorithm, which takes the cost (distance) of a certain node into account for determining a shortest path, can order the order in which the nodes should be traversed. The priority queue is also used to rank the nodes to travel, regardless of the order of insertion, by using the heuristic present in A Star search and Greedy Best First search.

### Multi-dimensional Array

My program uses two-dimensional arrays to store the x and y coordinate of every node in a graph. It is used to determine how many nodes are in the graph and the limits of the graph.

## Dictionary

A dictionary is used in my program to define the different type of nodes, such as light terrain, medium terrain, heavy terrain, open node, blocked node etc. A dictionary is used so when the appropriate key of a node is received, the correct node type can be returned

## Variables

My program uses a large number of variables including integers, floats, decimals, string, and boolean to store and define multiple parts of the project.

## File Structure

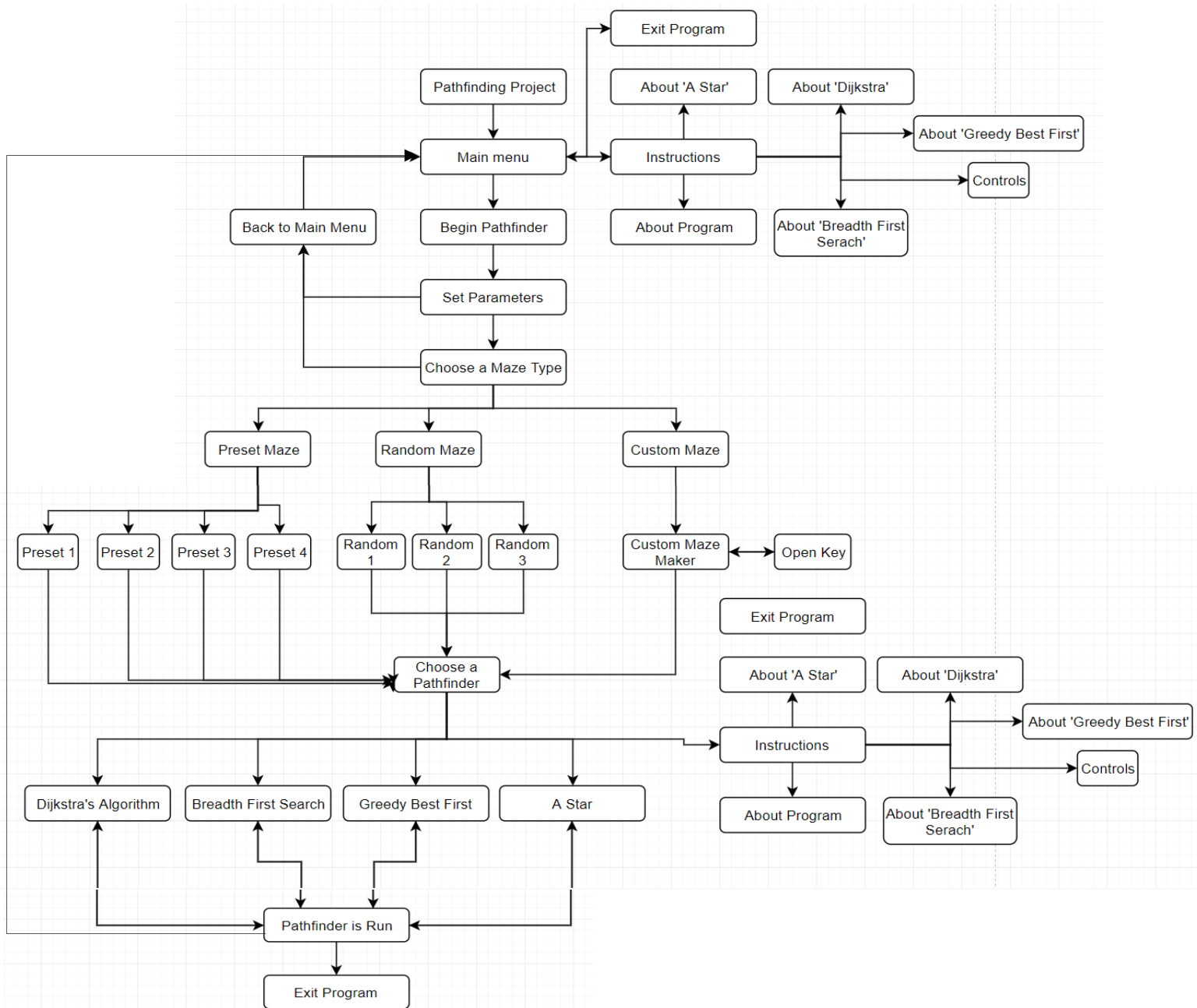
My program creates an external file, when all the pathfinding algorithms have been run. First my project creates an external text file named 'Log.txt' if a 'Log.txt' file does not already exist. Then the date and time at which the set of pathfinding algorithms were run is recorded. After that, it records the variable values that were set by the user for the maze in question and all the time taken, distance travelled, speed, and nodes explored values into the text file. 'Log.txt' which will be created next to the program file. *(Pictured below is an example)*

<pre> Log - Notepad File Edit Format View Help \\\\\\ Pathfinder Log: \\\\/\  \\ (New Maze) Log date: 4/17/2020 11:16:41 PM//  \\ Parameters: //  Maze Width: 100 Maze Height: 100 Start X: 0 Start Y: 50 Goal X: 99 Goal Y: 50   Border Size: 0.1  Exit On Goal: True Show Arrows: True Show Iterations: False Eight Directions: True  \\ Results: //  Pathfinder Mode: Dijkstra  Path Length: 99 metres  Time taken: 0.6730694 seconds  Average Speed: 147.0874 metres per second  Nodes explored: 3752 nodes  Pathfinder Mode: </pre>	<pre> AStar  Path Length: 99 metres  Time taken: 0.251276 seconds  Average Speed: 393.9891 metres per second  Nodes explored: 99 nodes  Pathfinder Mode: BreadthFirst  Path Length: 126.3382 metres  Time taken: 0.5805596 seconds  Average Speed: 217.6145 metres per second  Nodes explored: 3771 nodes  Pathfinder Mode: GreedyBestFirst  Path Length: 99 metres  Time taken: 0.2672676 seconds  Average Speed:  370.4153 metres per second  Nodes explored: 99 nodes </pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



## Structure diagram

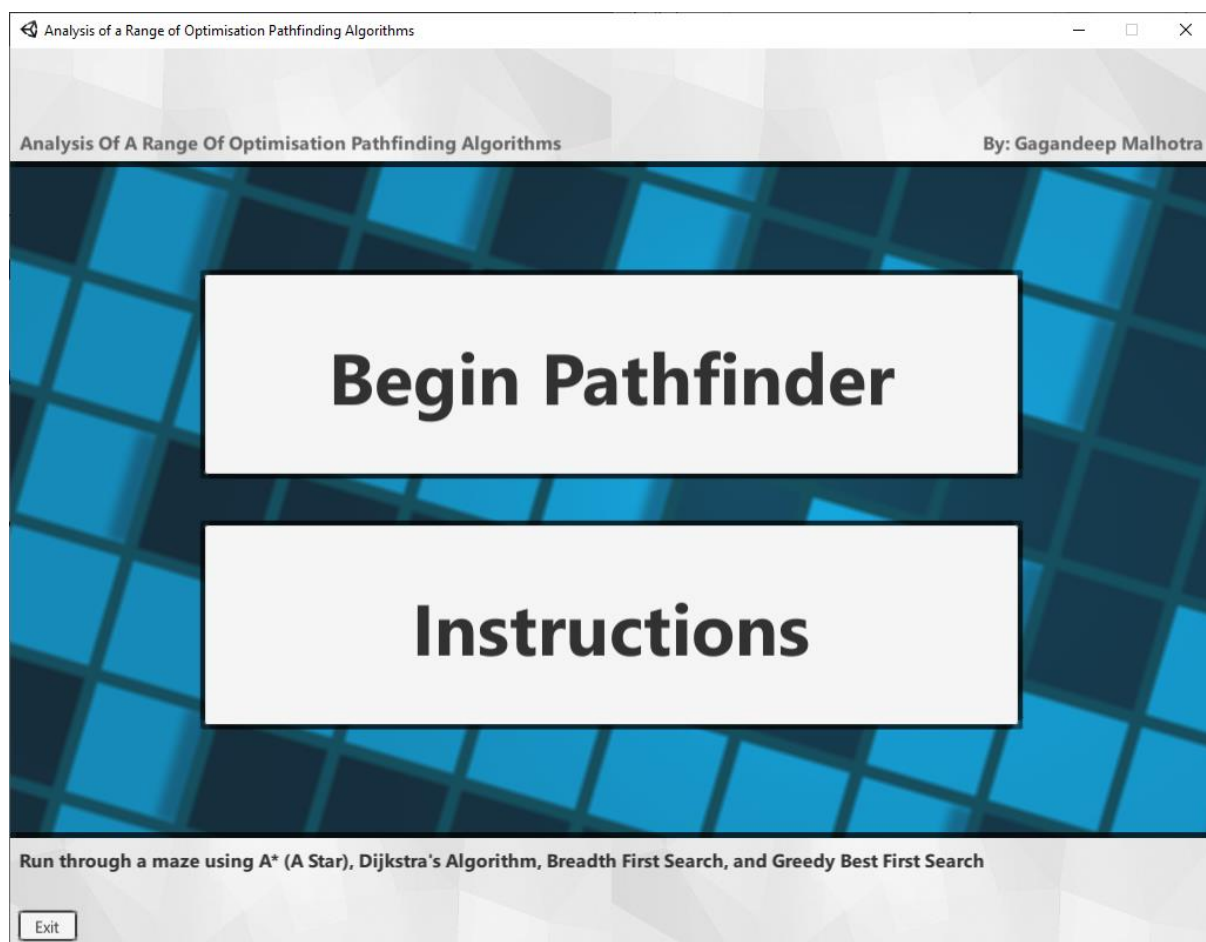
This diagram showcased the arrangement and layout of all the menus in my program and how they interconnect with one another. Menus displayed with a single arrow are one way, whereas double arrows indicate that the menus can lead back to each other (main menu and instructions).



## HCI (Human Computer Interaction)

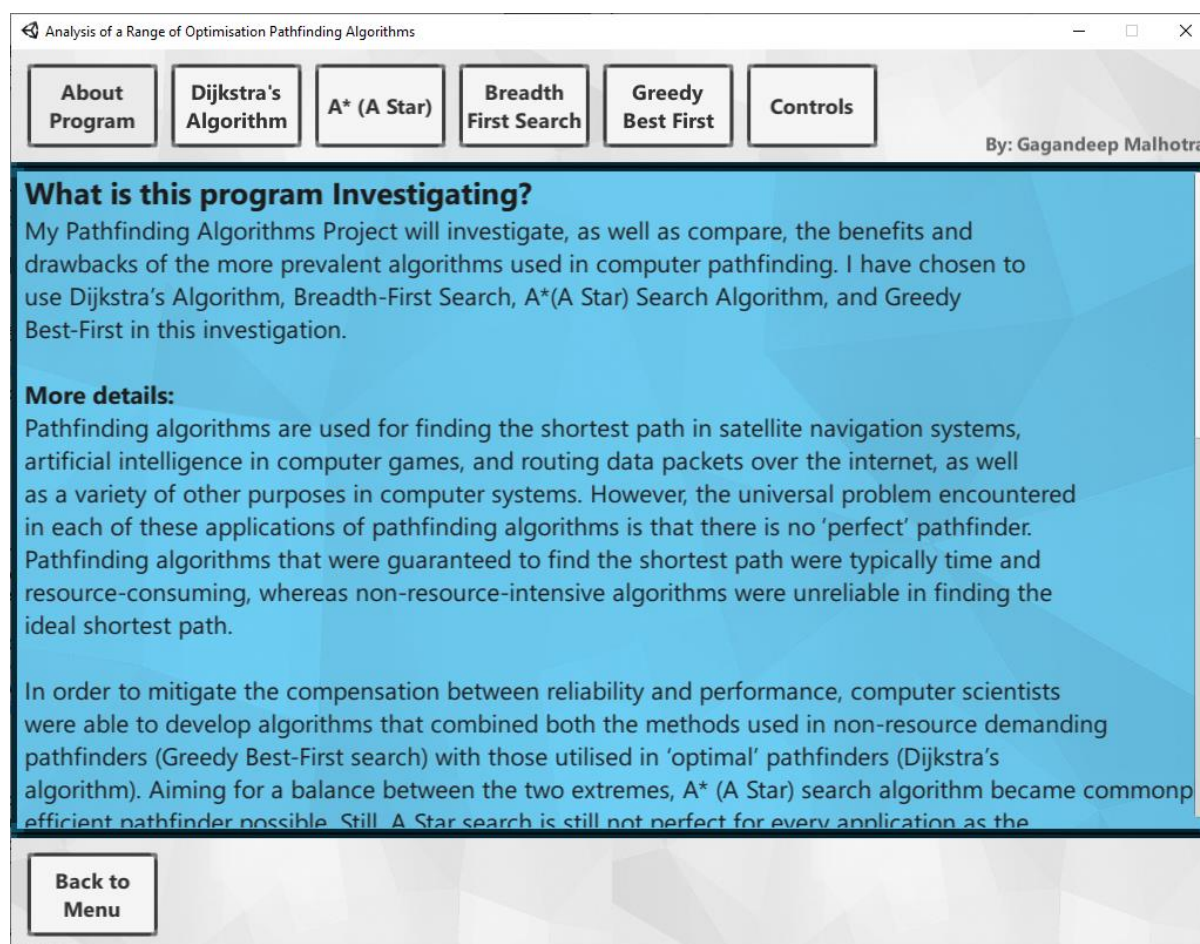
### Main Menu

The main menu is very simple with three options for the user to click, and like all menus can be navigated by using the mouse. The background is a scrolling example image of what a maze the user might generate could look like, to give the user an idea of what to expect from my project, I have chosen the large distinct buttons of 'begin pathfinder' and 'instructions' so the user is not overwhelmed by a large variety of options. A short description of what the button will do when clicked on is displayed on the bottom panel, with the name of the project and my name on the top panel. The small exit button is in the bottom left corner so it is not accidentally clicked on by the user.



### Instructions Page

The instructions page has six different panels where if the user clicks on a panel, the relevant information based on that subject is shown. There is a light blue background to make sure the background is not too plain and boring for a user who is looking to learn. There is also an option to go back to the menu for the user.

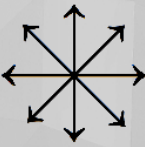


## Settings Page

The settings page has all the parameters which the user can edit clearly displayed in five sections (maze dimensions, start coordinates, goal coordinates, Behaviour of pathfinder, and visual settings). The behaviour of pathfinder section is all boolean, and the description on the right of it will change based on if the box is ticked or not by the user. The description boxes show relevant description and ranges of each setting, and the image at the top right will indicate whether the pathfinders can travel in four or eight directions. The user is able to input their number in the space with the grey italic text where if they click 'confirm' and do not change the number, the default value is used. When the user clicks on 'confirm' and an erroneous value is inputted, there will be a list to the right of the screen, in bold red, of all the parameters which are incorrect. The user is also able to go back to the main menu if they wish to exit the program, reset the values, or go to the instructions page.

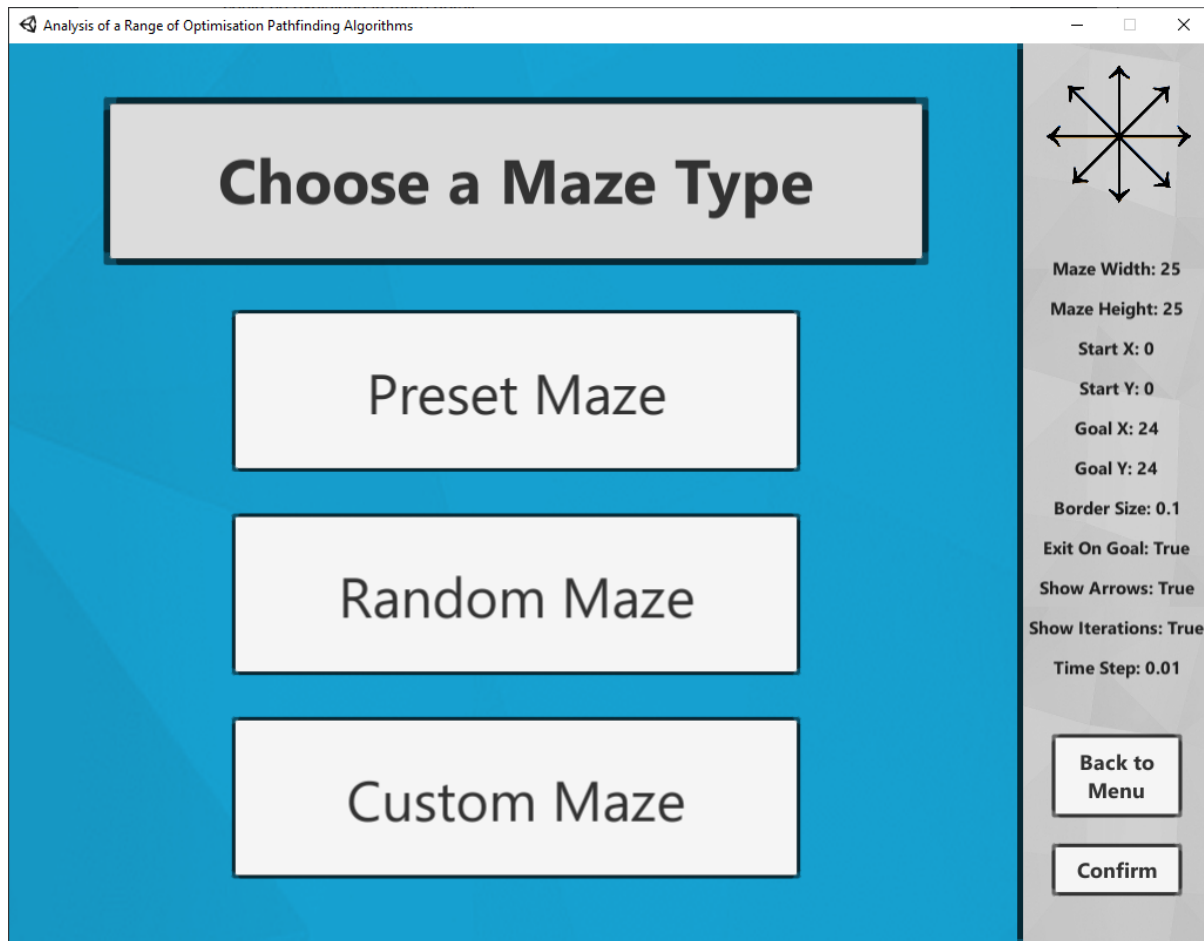
Analysis of a Range of Optimisation Pathfinding Algorithms

<b>Maze Width:</b>	25	Width of the maze in metres, <b>range (3-100)</b>
<b>Maze Height:</b>	25	Height of the maze in metres, <b>range (3-100)</b>
<b>Start X:</b>	0	X-axis position of the Start node, <b>range (0-99) &amp; below Width-1</b>
<b>Start Y:</b>	0	Y-axis position of the Start node, <b>range (0-99) &amp; below Height-1</b>
<b>Goal X:</b>	24	X-axis position of the Goal node, <b>range (0-99) &amp; below Width-1</b>
<b>Goal Y:</b>	24	Y-axis position of the Goal node, <b>range (0-99) &amp; below Height-1</b>
<b>Exit on Goal</b>	<input checked="" type="checkbox"/>	The pathfinder will stop when the goal is reached
<b>Show Arrows</b>	<input checked="" type="checkbox"/>	The arrows pointing to the previous node will be shown
<b>8 Directions</b>	<input checked="" type="checkbox"/>	The pathfinder will travel <b>vertical/horizontal/diagonal</b>
<b>Show Iterations</b>	<input checked="" type="checkbox"/>	The pathfinder searching through the maze will be shown
<b>Border Size:</b>	0.1	The visual size of the gap between the nodes, <b>range (0-0.5)</b>
<b>Time Step:</b>	0.01	Time interval (seconds) for each step in pathfinder, <b>range (0.01-5)</b>



## Maze Type Selection

On this page all the user parameters which were entered on the previous settings page will be displayed on the right panel. The three clear options of the maze type will be able to be selected and then 'confirmed' by the user to enter that section of the program. This is so the user does not click on an option by accident without reading through all the options.




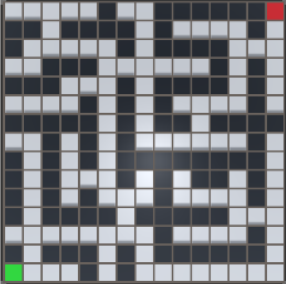
### Preset/Random Maze Menu

The Preset/Random page are similarly laid out for the user to maintain consistency and familiarity throughout the program. There are several options in both menus of the type of maze the user would like to generate with further information about the mazes to a panel on the right of a picture showing an example of the relevant maze. There is a far right panel which for the preset maze, reminds the user that their own entered values for the width and height of the maze will not be used as they are selecting a premade maze, whilst for the random maze this panel again displays all the parameters which the user has chosen earlier in the settings page. The user is able to scroll through the list of mazes, select one, and click 'confirm' to proceed with the program.

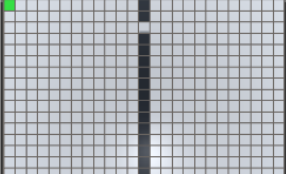
Analysis of a Range of Optimisation Pathfinding Algorithms

## Choose a Preset Maze

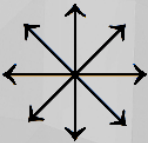




**Perfect Maze**  
Only one possible path with no loops  
Width: 15    Height: 15  
Start X: 0    Goal X: 14  
Start Y: 0    Goal Y: 14



**Small Gaps**  
Small gaps placed in difficult positions  
Width: 25    Height: 25



**WARNING:**  
Width  
Height  
Start X  
Start Y  
Goal X  
Goal Y

Will be **RESET** if a Preset Maze is chosen

**Confirm**

Analysis of a Range of Optimisation Pathfinding Algorithms

## Choose a Random Maze





**Dense Random Maze**  
A Maze with many Blocked Nodes  
(33% chance of a Blocked Node)



**Sparse Random Maze**  
A Maze with many Blocked Nodes

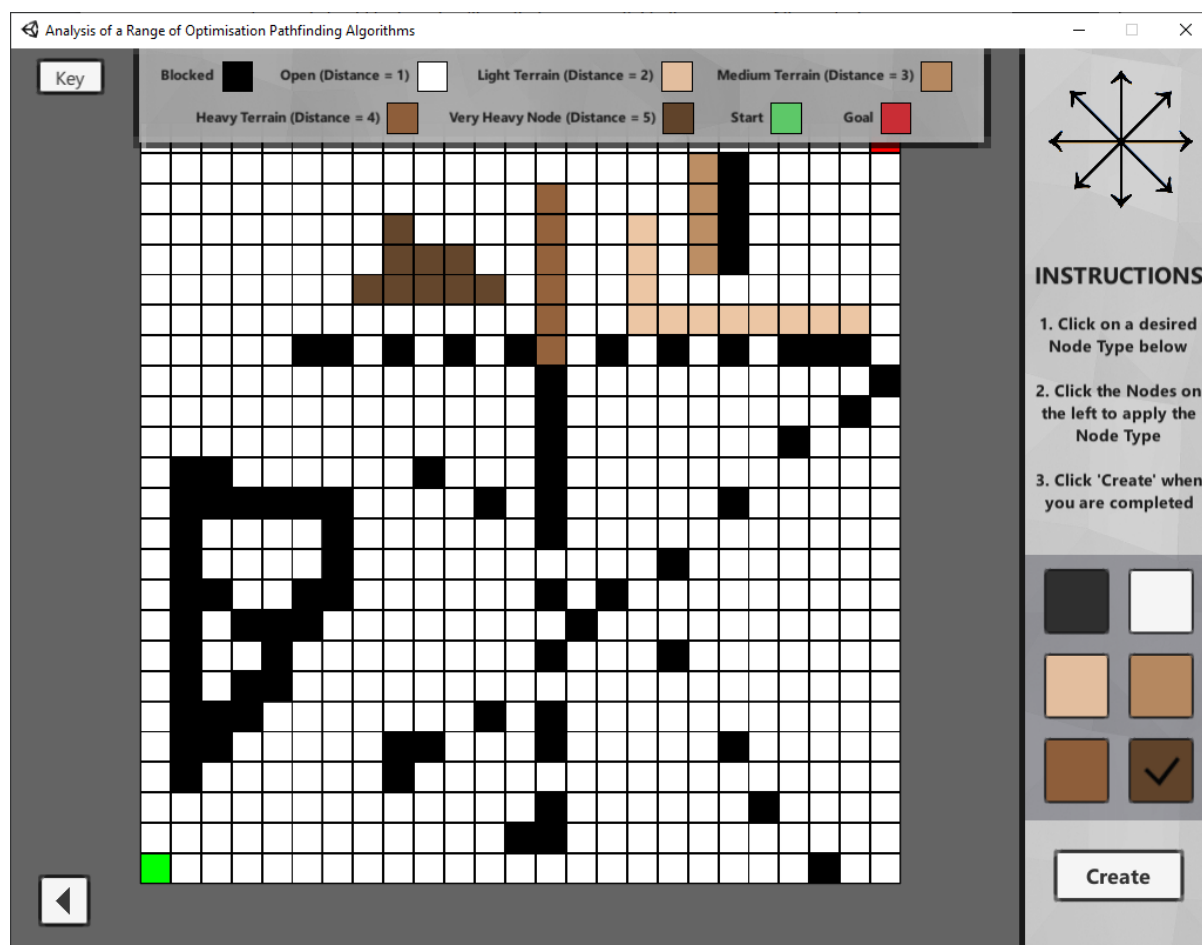


Maze Width: 25  
Maze Height: 25  
Start X: 0  
Start Y: 0  
Goal X: 24  
Goal Y: 24  
Border Size: 0.1  
Exit On Goal: True  
Show Arrows: True  
Show Iterations: True  
Time Step: 0.01

**Confirm**

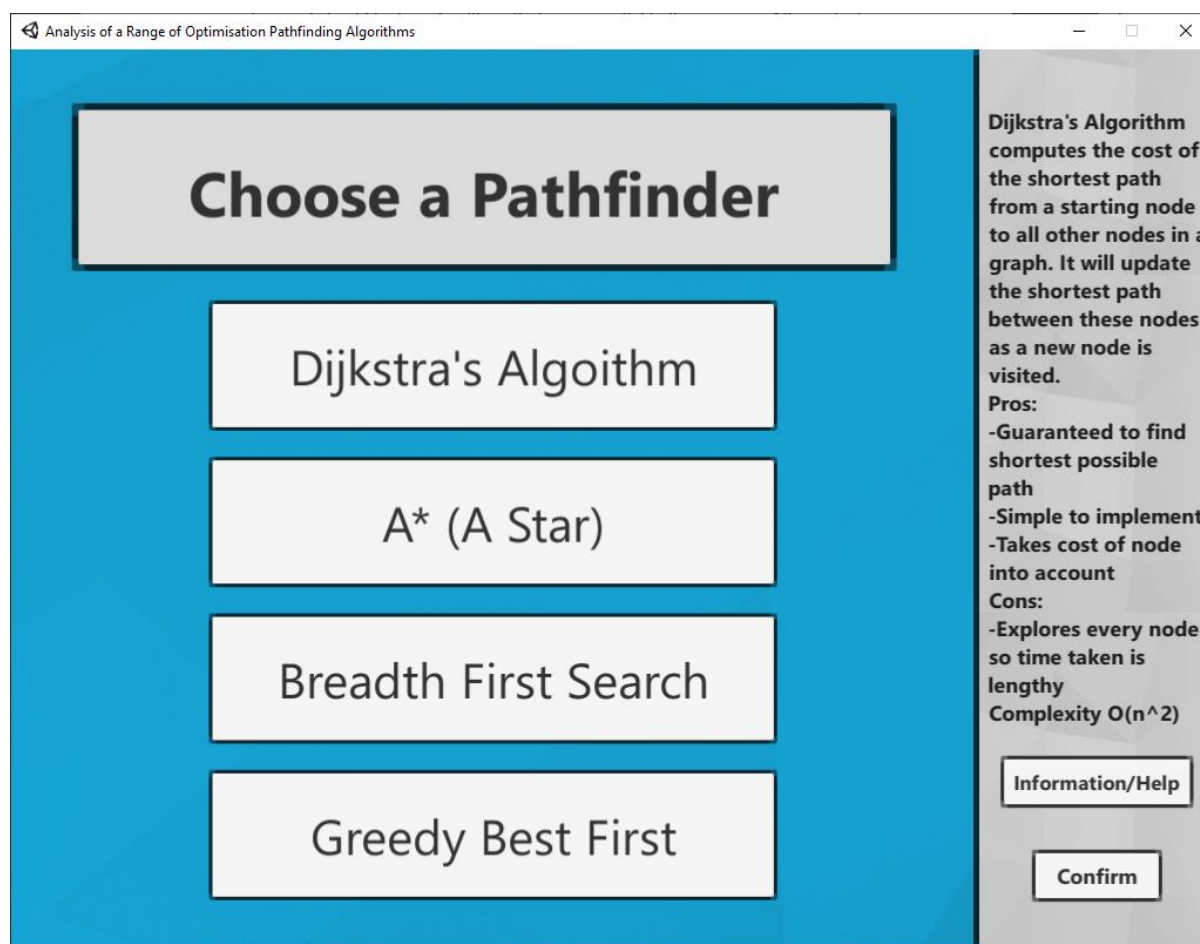
## Custom Maze Menu

This page has functionality similar to a painting application where the user is able to 'draw' the maze of their choice on the grid they set earlier in the settings page. A key is available for the user to see what the different nodes mean, as well as instructions on the far-right panel so the user knows how to draw their maze. When a node type is selected from the panel on the right, the user can draw this node type on the initially blank maze by holding and dragging their mouse over the nodes they desire, the start and goal nodes cannot be edited and are shown clearly by green and red respectively. The user is also able to zoom in and out using the scroll wheel on their mouse.



## Pathfinder Choosing Menu

This menu has a link to the instructions page if the user wishes to view it, and displays information on the four pathfinders in the far-right panel when a pathfinder is hovered over by the cursor. The user can easily select the pathfinder of their choice and click 'confirm' to proceed with the program. There is an error check, like on all these menus, to ensure the user has indeed selected an option before clicking 'confirm'.



## Pathfinders are run

When a pathfinder has been selected, the user will be brought to this page which initially displays the pathfinder being run clearly on a 2d graph. However, there are many easily accessible options for the user to edit the look and function of this page by, pausing the pathfinder by using the pause button. The user can also change the background colour and light direction in the project. Select another pathfinder will bring the user to the previous menu, and the 3-d camera option at the bottom-middle will change the camera view to three dimensions. The user can navigate the 2d camera view by using the scroll wheel to zoom in and out, left clicking and dragging to move the camera view across the maze, and right clicking to reset the camera position. The 3d camera pivots around the centre of the maze and can be zoomed in/out using the scroll wheel. The results of the pathfinder which have been run are displayed on a collapsible, scrollable panel on the right whereas the parameters set for the maze are shown on a collapsible panel on the left. The 'toggle information button' will clearly show to the user all the pathfinders which have been run in a ranking of 'time', 'distance', or 'speed', according to the user's choice. This menu is also navigable using the arrow keys and spacebar button as convenience for the user. There is also a warning page for the user if they click the 'Back to main menu button' to prevent them exiting the current pathfinder by accident.



The image shows two screenshots of a pathfinding software application. The top screenshot displays a 25x25 maze with a path highlighted in orange and blue. The interface includes a 'Toggle Parameters' button, a 'Toggle Results' button, and a '3-D Camera' button. Below the maze are sliders for 'R', 'G', and 'B' and a 'Select Another Pathfinder' button.

The bottom screenshot shows the same maze with a green background. It features a 'Key' legend with the following items: Blocked (black), Open (white), Explored (grey), Neighbour (light blue), Path (orange), Start (green), Goal (red), Old Neighbour (blue), and Old Explored (dark grey). The interface includes a 'Toggle Parameters' button, a 'Toggle Results' button, and a '3-D Camera' button. Below the maze are sliders for 'R', 'G', and 'B' and a 'Select Another Pathfinder' button.

**Pathfinder Mode: GreedyBestFirst**

**Path Length: 44.87006 metres**

**Time taken: 1.004292 seconds**

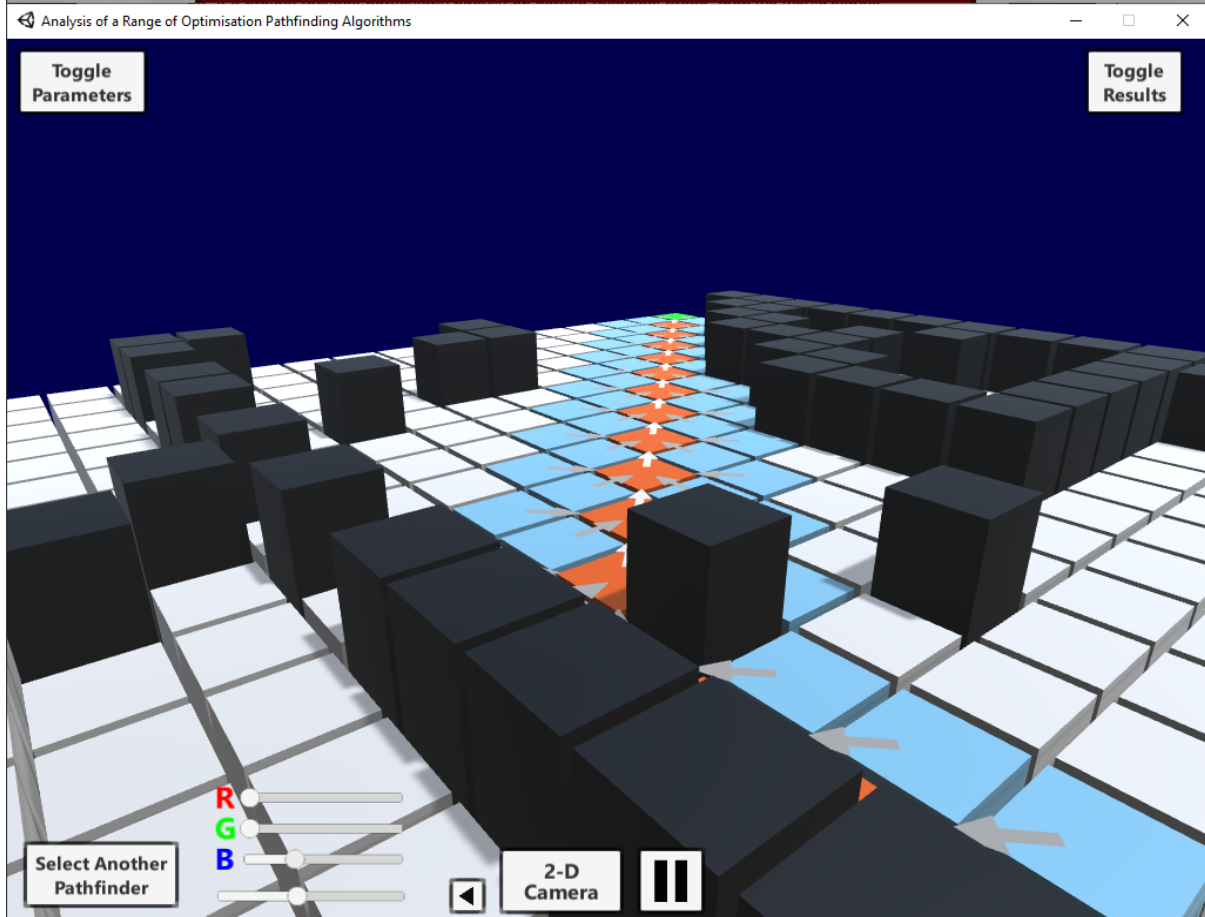
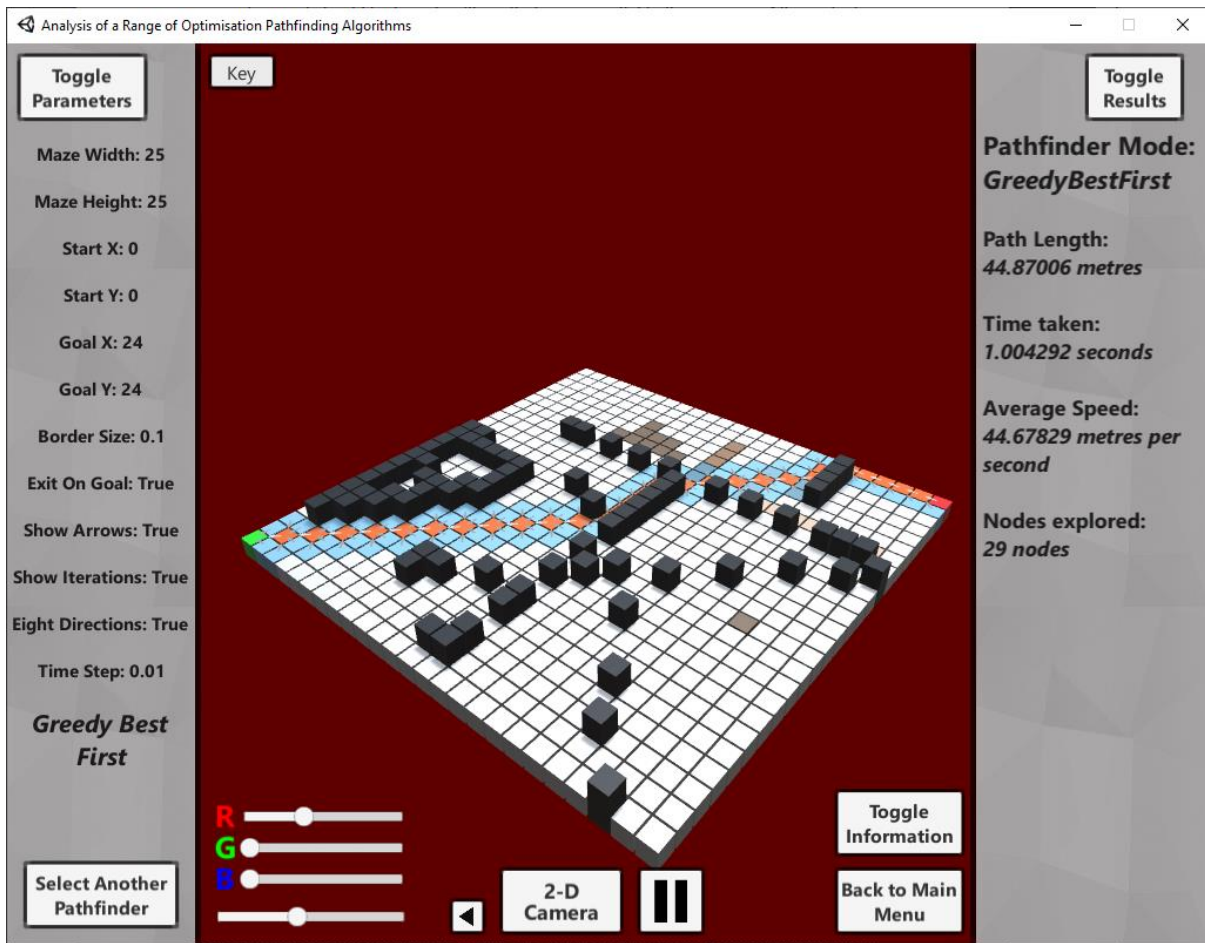
**Average Speed: 44.67829 metres per second**

**Nodes explored: 29 nodes**

**Greedy Best First**

**Toggle Information**

**Back to Main Menu**



Analysis of a Range of Optimisation Pathfinding Algorithms

Toggle Parameters

Sort by Distance | Sort by Time | Sort by Speed

**Greedy Best First:**  
Path Length: 44.87006 metres  
Time taken: 1.004292 seconds  
Average Speed: 44.67829 m/s  
#1

**Breadth First Search:**  
Has not been run yet  
#2

**A\* (A Star):**  
Has not been run yet  
#3

**Dijkstra's Algorithm:**  
Has not been run yet  
#4

Toggle Results

**Pathfinder Mode: GreedyBestFirst**

Path Length: 44.87006 metres

Time taken: 1.004292 seconds

Average Speed: 44.67829 metres per second

Nodes explored: 29 nodes

Select Another Pathfinder | R | G | B | 2-D Camera | Toggle Information | Back to Main Menu

Analysis of a Range of Optimisation Pathfinding Algorithms

Toggle Parameters

Toggle Results

**Pathfinder Mode: GreedyBestFirst**

Path Length: 44.87006 metres

Time taken: 1.004292 seconds

Average Speed: 44.67829 metres per second

Nodes explored: 29 nodes

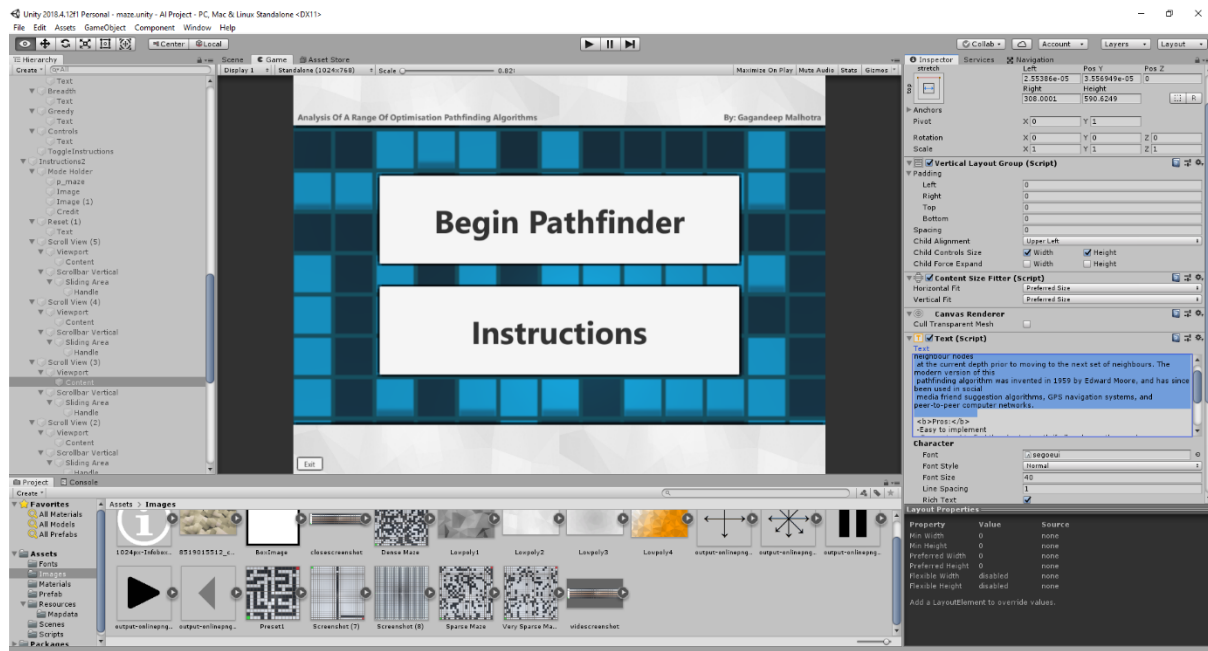
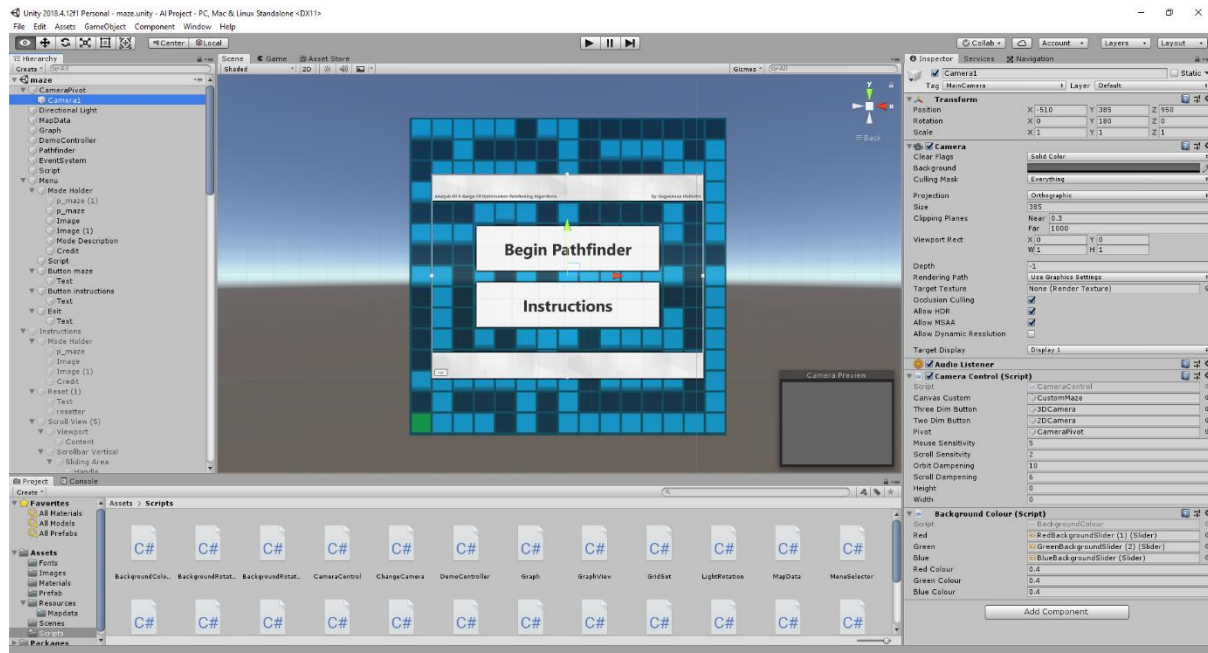
**WARNING: The current maze with all its data will be RESET if you choose to Exit to Menu**

Confirm | Cancel

Select Another Pathfinder | R | G | B | 3-D Camera | Toggle Information | Back to Main Menu

### Unity Engine Interface

Here is an example of the interface I used to create this program, in Unity.



# Technical Solution:

## Background Colour Code:

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

public class BackgroundColour : MonoBehaviour
{
    //Assigning the Sliders' values to relevant variable
    public Slider red;
    public Slider green;
    public Slider blue;

    //Setting default background colour values
    public float redColour = 0.4f;
    public float greenColour = 0.4f;
    public float blueColour = 0.4f;

    //When the Slider value is changed
    public void OnEdit()
    {
        //Updates the values of each Slider respectively
        redColour = red.value;
        greenColour = green.value;
        blueColour = blue.value;

        //Assigns these values to the background colour
        Camera.main.backgroundColor = new Color(redColour,greenColour,blueColour);
    }
}
```

## Background Rotation Code:

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

public class BackgroundRotation : MonoBehaviour
{
    //Assigns the background image
    public GameObject backgroundImage;

    // Update is called once per frame
    void Update()
    {
        //When the background image is visible, rotate it slowly around the z-axis
        if (backgroundImage.activeInHierarchy == true)
        {
            backgroundImage.transform.Rotate(0, 0, 0.33f);
        }
    }
}
```

## Camera Control Code:

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

public class CameraControl : MonoBehaviour
{
    //Declaring coordinates of each variable (0,0,0)
    private Vector3 ResetCamera;
    private Vector3 Origin;
    private Vector3 Diference;
    protected Vector3 _LocalRotation;

    //Assigns a gameobject to each variable
    public GameObject canvasCustom;
    public GameObject ThreeDimButton;
    public GameObject TwoDimButton;
    public GameObject Pivot;

    //Declaring values of camera control using a mouse
    public float MouseSensitivity = 4f;
    public float ScrollSensitivty = 2f;
    public float OrbitDampening = 10f;
    public float ScrollDampening = 6f;
    protected float _CameraDistance = 10f;

    protected Transform _XForm_Camera;
    protected Transform _XForm_Parent;

    //Declaring the camera being draggable as false
    private bool Drag = false;

    //Declaring the default variable values
    public int height;
    public int width;

    //Called when the program is initialised
    private void Start()
    {
        this._XForm_Camera = this.transform;
        this._XForm_Parent = this.transform.parent;
    }

    //Called once a frame
    void LateUpdate()
    {
        //Check for if the right menu is open
        if (GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().zoom
== true && ThreeDimButton.activeInHierarchy == true ||
        canvasCustom.activeInHierarchy == true &&
        GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().zoom == false)
        {
            //Middle MouseButton/Scroll Wheel is pressed
            if (Input.GetMouseButton(2))
            {
                //Check for if the mouse is not over a UI component
                if (!EventSystem.current.IsPointerOverGameObject())
                {
```

```

        //Get the value for how far the mouse has dragged from its
original position
        Diference = (Camera.main.ScreenToWorldPoint(Input.mousePosition))
- Camera.main.transform.position;
        if (Drag == false)
        {
            //Setting the camera to draggable
            Drag = true;
            Origin = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        }
    }
else
{
    //Set camera draggable to false
    Drag = false;

    //Check for if camera is draggable
    if (Drag == true)
    {
        //Move the camera according to the drag direction
        Camera.main.transform.position = Origin - Diference;
    }

    //Check for if camera zoom is enabled
    if
(GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().zoom == true)
    {
        //Reset camera to starting position on right-click
        if (Input.GetMouseButton(1))
        {
            //Reset camera Vector3 Coordinate
            Camera.main.transform.position = ResetCamera;
            //Reset camera zoom
            Camera.main.orthographicSize = (height / 2 + 5);
        }
    }
else
{
    //Reset camera to starting position on right-click
    if (Input.GetMouseButton(1))
    {
        //Reset camera Vector3 Coordinate
        Camera.main.transform.localPosition = new Vector3(-510, 385, 950);
        //Reset camera zoom
        Camera.main.orthographicSize = (385);
    }
}
}

//Update is called once per frame
public void Update()
{
    //Check for if the camera is in three-dimensional mode
    if (GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().zoom
== true && TwoDimButton.activeInHierarchy == true)
    {
        //Check for if the camera is in orthographic view
        if (Camera.main.orthographic == true)
        {

```

```

        //Change camera to perspective view
        Camera.main.orthographic = false;
        //Set 3D camera rotation to zero and position to the origin
        Camera.main.transform.eulerAngles = new Vector3(0, 0, 0);
        Camera.main.transform.localEulerAngles = new Vector3(0, 0, 0);
        Camera.main.transform.position = new Vector3(0, 0, 0);

        //Move the camera's position to the center of the graph of nodes
        Pivot.transform.position = new Vector3((width - 1) / 2, 2, (height -
1) / 2);
    }
    //Rotation of the Camera based on Mouse Coordinates
    if (Input.GetAxis("Mouse X") != 0 || Input.GetAxis("Mouse Y") != 0)
    {
        _LocalRotation.x += Input.GetAxis("Mouse X") * MouseSensitivity;
        _LocalRotation.y += Input.GetAxis("Mouse Y") * MouseSensitivity;

        //Clamp the y Rotation to horizontal so it does not flip over and see
        under the graph of nodes
        if (_LocalRotation.y < 0f)
            _LocalRotation.y = 0f;
        else if (_LocalRotation.y > 90f)
            _LocalRotation.y = 90f;
    }
    //Zooming Input from our Mouse Scroll Wheel
    if (Input.GetAxis("Mouse ScrollWheel") != 0f)
    {
        //Declaring the amount at which the zoom function should zoom in the
camera
        float ScrollAmount = Input.GetAxis("Mouse ScrollWheel") *
ScrollSensitivity;
        ScrollAmount *= (this._CameraDistance * 0.3f);
        this._CameraDistance += ScrollAmount * -1f;

        //Setting the maximum and minimum zoom of the camera
        this._CameraDistance = Mathf.Clamp(this._CameraDistance, 2f, 100f);
    }

    //Transformations/Movement of the camera
    Quaternion QT = Quaternion.Euler(_LocalRotation.y, _LocalRotation.x, 0);
    this._XForm_Parent.rotation = Quaternion.Lerp(this._XForm_Parent.rotation,
QT, Time.unscaledDeltaTime * OrbitDampening);

    //Smoothly move the camera's position along the z-axis based on the
mouse's movement
    if (this._XForm_Camera.localPosition.z != this._CameraDistance * -1f)
    {
        this._XForm_Camera.localPosition = new Vector3(0f, 0f, Mathf.Lerp
(this._XForm_Camera.localPosition.z, this._CameraDistance * -1f,
Time.unscaledDeltaTime * ScrollDampening));
    }
}

//Check for if the camera is in two-dimensional mode
if (GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().zoom
== true && ThreeDimButton.activeInHierarchy == true)
{
    //Check for if the camera is in perspective view
    if (Camera.main.orthographic == false)
    {
        //Change camera view to orthographic view

```



```

        Camera.main.orthographic = true;

        //Declaring the rotation of the orthographic camera
        Camera.main.transform.eulerAngles = new Vector3(90, 0, 0);
        CameraController();
    }
    //Check for if mouse is not over a UI component
    if (!EventSystem.current.IsPointerOverGameObject())
    {
        //Check if scroll wheel is scrolled forwards
        if (Input.GetAxis("Mouse ScrollWheel") > 0)
        {
            //Zoom in by two units

ZoomOrthoCamera(Camera.main.ScreenToWorldPoint(Input.mousePosition), 2f);
        }

        //Check if scroll wheel is scrolled backwards
        if (Input.GetAxis("Mouse ScrollWheel") < 0)
        {
            //Zoom out by two units

ZoomOrthoCamera(Camera.main.ScreenToWorldPoint(Input.mousePosition), -2f);
        }
    }

    //Check for if the custom maze menu is open
    else if (canvasCustom.activeInHierarchy == true &&
    GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().zoom == false)
    {
        //Check for if mouse is not over a UI component
        if (!EventSystem.current.IsPointerOverGameObject())
        {
            //Check if scroll wheel is scrolled forwards
            if (Input.GetAxis("Mouse ScrollWheel") > 0)
            {
                //Zoom in by twenty five units

ZoomOrthoCamera(Camera.main.ScreenToWorldPoint(Input.mousePosition), 25f);
            }

            //Check if scroll wheel is scrolled backwards
            if (Input.GetAxis("Mouse ScrollWheel") < 0)
            {
                //Zoom out by twenty five units

ZoomOrthoCamera(Camera.main.ScreenToWorldPoint(Input.mousePosition), -25f);
            }
        }
    }

    //Declares the default position and rotation of the camera according to the graph
    size
    public void CameraController()
    {
        //Declaring width and height of graph
        width =
    GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().width;
        height =
    GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().height;

```

```

        //Declaring default rotation of camera and finding the midpoint of the graph
        Camera.main.transform.eulerAngles = new Vector3(90, 0, 0);
        Camera.main.transform.position = new Vector3(((width - 1) / 2) + 0.5f, 5,
(height - 1) / 2);
        //Storing the midpoint of the graph
        ResetCamera = Camera.main.transform.position;

        // Aligns the camera size with the dimension of the grid of nodes, allowing
the user to clearly see the whole grid
        Camera.main.orthographicSize = (height / 2 +5);
    }

    //Zooming in and out the two-dimensional camera
    void ZoomOrthoCamera(Vector3 zoomTowards, float amount)
    {
        //Check if camera zoom is enabled
        if (GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().zoom
== true)
        {
            // Zoom camera out
            Camera.main.orthographicSize -= amount;

            // Limit zoom of the camera
            Camera.main.orthographicSize = Mathf.Clamp(Camera.main.orthographicSize,
3, 55);
        }

        //Check if the custom maze menu is open
        else if (canvasCustom.activeInHierarchy == true)
        {
            // Zoom camera
            Camera.main.orthographicSize -= amount;

            // Limit zoom of the camera
            Camera.main.orthographicSize = Mathf.Clamp(Camera.main.orthographicSize,
100, 385);
        }
    }
}

```

## Change Camera Code:

```

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

public class ChangeCamera : MonoBehaviour
{
    //Assigning Gameobjects to each variable
    public GameObject TwoDimButton;
    public GameObject ThreeDimButton;

    //Called when the camera three-dimensional mode is turned on
    public void ThreeDim()
    {
        //Hide the button for three-dimensional mode and show the button for two-
dimensional mode
        ThreeDimButton.SetActive(false);
        TwoDimButton.SetActive(true);
    }
}

```

```

    }

    //Called when the camera two-dimensional mode is turned on
    public void TwoDim()
    {
        //Hide the button for two-dimensional mode and show the button for three-
        dimensional mode
        TwoDimButton.SetActive(false);
        ThreeDimButton.SetActive(true);
    }
}

```

## Demo Controller Code:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class DemoController : MonoBehaviour
{
    //Assigning Gameobjects with scripts to each variable
    public MapData mapData;
    public Graph graph;
    public Pathfinder pathfinder;

    //Declaring default variable values
    public int startX;
    public int startY;
    public int goalX;
    public int goalY;
    public int pass = 0;
    public float timeStep;

    //Two-dimensional array to store the dimensions of graph of nodes
    public int[,] mapInstance;

    //Called when the maze is created
    public void BeginMaze()
    {
        //Declare all the variable to what the user has inputted in the setting menu
        int startX =
        GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().startX;
        int startY =
        GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().startY;
        int goalX =
        GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().goalX;
        int goalY =
        GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().goalY;
        int width =
        GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().width;
        int height =
        GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().height;

        //Check if a subsequent pathfinding algorithm is being run
        if (pass >= 1)
        {
            //For all nodes in the graph
            for (int row = 0; row < height; row++)
            {

```

```

        for (int col = 0; col < width; col++)
        {
            //Destroy the node in the graph (to save memory space)
            GameObject toDestroy = GameObject.Find("Node (" + row + ", " + col
+ ")");
            Destroy(toDestroy);
        }
    }

//Check if there is mapdata and a graph
if (mapData != null && graph != null)
{
    //Check if the graph of nodes has been created yet
    if (pass == 0)
    {
        //Collect data for graph of nodes
        mapInstance = mapData.MakeMap();
    }

    //Initialise the graph of nodes
    graph.Init(mapInstance);
    //Assign the script of the graphview to a gameobject
    GraphView graphView = graph.gameObject.GetComponent<GraphView>();

    //Check whether the graphview script is attached
    if (graphView != null)
    {
        //Initialise the graph of nodes
        graphView.Init(graph);
    }
    //If the graph has valid dimensions
    if (graph.IsWithinBounds(startX, startY) && graph.IsWithinBounds(goalX,
goalY))
    {
        //Set the start and goal coordinates of the graph
        Node startNode = graph.nodes[startX, startY];
        Node goalNode = graph.nodes[goalX, goalY];

        //Initialise the pathfinder with the variables above
        pathfinder.Init(graph, graphView, startNode, goalNode);

        //Begin the pathfinder
        StartCoroutine(pathfinder.SearchRoutine(timeStep));
    }

    //Check if the pathfinder is being run for the first time
    if (pass == 0)
    {
        //Sort the information of the pathfinder by shortest distance
        GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().SortDistanceUpdate
        ();
    }
}
//Set the camera settings (position, rotation, mode)
GameObject.Find("Camera1").GetComponent<CameraControl>().CameraController();
//Increment the number of times a pathfinder has been run
pass += 1;
}
}

```

## Graph Code:

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

//Class used to manage the entire set of nodes
public class Graph : MonoBehaviour
{
    //Default the variable eight directions as true
    public bool eightDir = true;

    //Two dimensional array to hold each node's coordinates
    public Node[,] nodes;
    int[,] m_mapData;

    //List to hold all the wall/blocked nodes in the graph
    public List<Node> walls = new List<Node>();

    //Get and return the width and height of the graph of nodes
    int m_width;
    public int Width { get { return m_width; } }
    int m_height;
    public int Height { get { return m_height; } }

    //List of two dimensional vectors in eight directions
    public Vector2[] eightDirections =
    {
        //horizontal and vertical
        new Vector2(0f,1f),
        new Vector2(1f,0f),
        new Vector2(0f,-1f),
        new Vector2(-1f,0f),
        //diagonals
        new Vector2(1f,1f),
        new Vector2(1f,-1f),
        new Vector2(-1f,-1f),
        new Vector2(-1f,1f),
    };

    //List of two dimensional vectors in eight directions
    public Vector2[] fourDirections =
    {
        //horizontal and vertical
        new Vector2(0f,1f),
        new Vector2(1f,0f),
        new Vector2(0f,-1f),
        new Vector2(-1f,0f),
    };

    //Initialises the graph of nodes using the mapdata
    public void Init(int[,] mapData)
    {
        //Store the mapdata
        m_mapData = mapData;
        //Gets length of x-values in two dimensional array
        m_width = mapData.GetLength(0);
        //Gets length of y-values in two dimensional array
        m_height = mapData.GetLength(1);

        //Initialises the array
        nodes = new Node[m_width, m_height];
    }
}
```

```
//For each(x,y) position in the array
for (int y = 0; y < m_height; y++)
{
    for(int x = 0; x < m_width; x++)
    {
        //Set the node type based on information from mapdata
        NodeType type = (NodeType)mapData[x, y];

        //Generate a new node and store it in the array
        Node newNode = new Node(x, y, type);
        nodes[x, y] = newNode;

        //Set the Vector3 (x,y,z) position of the node
        newNode.position = new Vector3(x, 0, y);

        //Check if the node type is blocked
        if (type == NodeType.Blocked)
        {
            //Store the blocked node in the list of walls
            walls.Add((newNode));
        }
    }
}

//Determine the neighbouring nodes for each node in the array
for (int y=0; y < m_height; y++)
{
    for (int x = 0; x < m_width; x++)
    {
        //Check to see if the neighbouring node is not a wall/blocked
        if (nodes[x,y].nodeType != NodeType.Blocked)
        {
            //Stores the neighbours which are not walls
            nodes[x, y].neighbours = GetNeighbours(x, y);
        }
    }
}

}

//Check to see if the coordinates of a neighbouring node is in the graph
public bool IsWithinBounds(int x, int y)
{
    return (x >= 0 && x < m_width && y >= 0 && y < m_height);
}

//Returns a list of neighbouring nodes based on the node's coordinates and number
of directions
List<Node> GetNeighbours(int x, int y, Node[,] nodeArray, Vector2[] directions)
{
    //Creates a new empty list of nodes
    List<Node> neighbourNodes = new List<Node>();

    //In either four or eight directions
    foreach (Vector2 dir in directions)
    {
        //Find the offset in both the x and y direction between the nodes
        int newX = x + (int)dir.x;
        int newY = y + (int)dir.y;

        //If this new node position is within the graph and not blocked, add it to
        the list of neighbours
    }
}
```

```

        if (IsWithinBounds(newX,newY) && nodeArray[newX,newY] != null &&
            nodeArray[newX,newY].nodeType != NodeType.Blocked)
        {
            neighbourNodes.Add(nodeArray[newX, newY]);
        }
    }
    //Returns the list of neighbours to a node
    return neighbourNodes;
}

//Returns the list of neighbouring nodes
List<Node> GetNeighbours(int x, int y)
{
    //Check if eight directions is set to either true or false
    eightDir =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().eightDirections;
    if (eightDir == true)
    {
        //Return list of neighbouring nodes in eight directions
        return GetNeighbours(x, y, nodes, eightDirections);
    }
    else
    {
        //Return list of neighbouring nodes in four directions
        return GetNeighbours(x, y, nodes, fourDirections);
    }
}

//Get the approximate distance between nodes (square root of two has been
approximated to 1.41421356)
public float GetNodeDistance(Node source, Node target)
{
    //Find difference in x distance and y distance
    float dx = Mathf.Abs(source.xIndex - target.xIndex);
    float dy = Mathf.Abs(source.yIndex - target.yIndex);

    //Find which of these two differences are larger and smaller
    float min = Mathf.Min(dx, dy);
    float max = Mathf.Max(dx, dy);

    //Find the diagonal and straight steps needed from start to goal
    float diagonalSteps = min;
    float straightSteps = max - min;

    //Measures distance from current node to goal node
    return (1.41421356f* diagonalSteps) + straightSteps;
}
}

```

## Graph View Code:

```

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

//User interface in unity used to represent a set of nodeview, which is attached to the
graph gameobject
[RequireComponent(typeof(Graph))]
public class GraphView : MonoBehaviour
{

```

```

//Node object with existing dimensions and components
public GameObject nodeViewPrefab;

//Two-dimensional array of nodeviews
public NodeView[,] nodeViews;

//Colours of all the different node types
public Color openColor = Color.white;
public Color wallColor = Color.black;
public Color lightTerrainColor = new Color32(236,198,164,255);
public Color mediumTerrainColor = new Color32(188,142,100,255);
public Color heavyTerrainColor = new Color32(148,98,60,255);
public Color veryHeavyTerrainColor = new Color32(100,70,44,255);

//Called before the program begins
private void Awake()
{
    //Calls the dictionary of node types
    SetupLookupTable();
}
//Creates a dictionary of the node type and the colours associated
Dictionary<Color, NodeType> terrainLookupTable = new Dictionary<Color,
NodeType>();

//Initialise using the graph script
public void Init(Graph graph)
{
    //Error check to see if graph exists
    if(graph==null)
    {
        Debug.LogWarning("There is NO GRAPH to initialise!");
        return;
    }

    //setup two dimensional array of nodeviews
    nodeViews = new NodeView[graph.Width, graph.Height];

    //For each node in the graph
    foreach (Node n in graph.nodes)
    {
        //Creates a nodeview for each node
        GameObject instance = Instantiate(nodeViewPrefab, Vector3.zero,
Quaternion.identity);
        NodeView nodeView = instance.GetComponent<NodeView>();

        //Check if nodeview is not null
        if (nodeView != null)
        {
            //initialise each nodeview
            nodeView.Init(n);
            //Store each nodeview in the array
            nodeViews[n.xIndex, n.yIndex] = nodeView;

            //Check if the node is blocked
            if (n.nodeType == NodeType.Blocked)
            {
                //Color the node black
                nodeView.ColorNode(wallColor);
                //Increases the walls' y-scale so it visually appears as a wall
                //surrounding the nodes
                nodeView.transform.localScale += new Vector3(0, 1, 0);
                //Moves the walls to the correct y-position after scaling

```



```

        nodeView.transform.Translate(0, 0.5f, 0);
    }

    //Check if node is light terrain (horizontal distance = 2 metres)
    else if (n.nodeType == NodeType.LightTerrain)
    {
        //Colour the node very light brown
        nodeView.ColorNode(lightTerrainColor);
    }
    //Check if node is medium terrain (horizontal distance = 3 metres)
    else if (n.nodeType == NodeType.MediumTerrain)
    {
        //Colour the node light brown
        nodeView.ColorNode(mediumTerrainColor);
    }
    //Check if node is heavy terrain (horizontal distance = 4 metres)
    else if (n.nodeType == NodeType.HeavyTerrain)
    {
        //Colour the node brown
        nodeView.ColorNode(heavyTerrainColor);
    }
    //Check if node is very heavy terrain (horizontal distance = 5 metres)
    else if (n.nodeType == NodeType.veryHeavyTerrain)
    {
        //Colour the node dark brown
        nodeView.ColorNode(veryHeavyTerrainColor);
    }
    //Check if node is open (horizontal distance = 1 metre)
    else
    {
        //Colour the node white
        nodeView.ColorNode(openColor);
    }
}
}
}

//Colour a list of nodeviews, blending the new colour with the original colour
public void ColorNodes(List<Node> nodes, Color color, bool lerpColor = false,
float lerpValue = 0.5f)
{
    //For each node
    foreach (Node n in nodes)
    {
        //Check is node is not null
        if(n != null)
        {
            //Find the corresponding nodeview
            NodeView nodeView = nodeViews[n.xIndex, n.yIndex];

            //Store the default colour
            Color newColor = color;

            //If colours are being blended
            if (lerpColor)
            {
                //Find the original colour
                Color originalColor = GetColorFromNodeType(n.nodeType);

                //Calculate the blended colour
                newColor = Color.Lerp(originalColor, newColor, lerpValue);
            }
        }
    }
}

```

```

        //Check if nodeview is not null
        if (nodeView != null)
        {
            //Colour the nodeview in the new blended colour
            nodeView.ColorNode(newColor);
        }
    }
}

//Show the path arrows for a node
public void ShowNodeArrows(Node node, Color color)
{
    //Check if node is not null
    if(node != null)
    {
        //Store coordinates of a node
        NodeView nodeView = nodeViews[node.xIndex, node.yIndex];
        //Check if the nodeview is not null
        if(nodeView != null)
        {
            //Colour the path arrows
            nodeView.ShowArrow(color);
        }
    }
}

//Show the path arrows for a list of nodes
public void ShowNodeArrows(List<Node> nodes, Color color)
{
    //For each node in the graph
    foreach(Node n in nodes)
    {
        //Colour the path arrows
        ShowNodeArrows(n, color);
    }
}

//A dictionary holding all the node types and the corresponding colour and
distance
void SetupLookupTable()
{
    terrainLookupTable.Add(openColor, NodeType.Open);
    terrainLookupTable.Add(wallColor, NodeType.Blocked);
    terrainLookupTable.Add(lightTerrainColor, NodeType.LightTerrain);
    terrainLookupTable.Add(mediumTerrainColor, NodeType.MediumTerrain);
    terrainLookupTable.Add(heavyTerrainColor, NodeType.HeavyTerrain);
    terrainLookupTable.Add(veryHeavyTerrainColor, NodeType.veryHeavyTerrain);
}

//Searches the table for the colour of a node
public Color GetColorFromNodeType(NodeType nodeType)
{
    //Check if node type is in the dictionary
    if (terrainLookupTable.ContainsValue(nodeType))
    {
        //Stores the colour of the current node type
        Color colorKey = terrainLookupTable.FirstOrDefault(x => x.Value ==
nodeType).Key;
        //Returns this colour
        return colorKey;
    }
}

```

```
    }  
    //Default colour is white  
    return Color.white;  
  }  
}
```

## Grid Set Code:

```
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
  
//The GridSet class is for the custom maze menu where the user can create their own  
graph  
public class GridSet : MonoBehaviour  
{  
    //Setting the grid size and position  
    [SerializeField]  
    private Vector2 gridSize;  
    [SerializeField]  
    private Vector2 gridOffset;  
  
    //Setting the cells in the grid size and spacing  
    [SerializeField]  
    private Sprite cellSprite;  
    private Vector2 cellSize;  
    private Vector2 cellScale;  
  
    //Assigning toggles and variables to each node type the user can select  
    public Toggle iBlockedNode;  
    public bool blockedNode;  
    public Toggle iOpenNode;  
    public bool openNode;  
    public Toggle iLightTerrainNode;  
    public bool lightTerrainNode;  
    public Toggle iMediumTerrainNode;  
    public bool mediumTerrainNode;  
    public Toggle iHeavyTerrainNode;  
    public bool heavyTerrainNode;  
    public Toggle iVeryHeavyTerrainNode;  
    public bool veryHeavyTerrainNode;  
  
    //Called when an open node is clicked  
    public void OpenClicked()  
    {  
        //Sets all other node types to false  
        blockedNode = false;  
        lightTerrainNode = false;  
        mediumTerrainNode = false;  
        heavyTerrainNode = false;  
        //Sets open node type to true  
        openNode = true;  
    }  
    //Called when a blocked node is clicked  
    public void BlockedClicked()  
    {  
        //Sets all other node types to false  
        openNode = false;  
        lightTerrainNode = false;  
        mediumTerrainNode = false;  
    }  
}
```

```
        heavyTerrainNode = false;
        veryHeavyTerrainNode = false;
        //Sets blocked node type to true
        blockedNode = true;
    }
    //Called when a light terrain node is clicked
    public void LightTerrainClicked()
    {
        //Sets all other node types to false
        openNode = false;
        blockedNode = false;
        mediumTerrainNode = false;
        heavyTerrainNode = false;
        veryHeavyTerrainNode = false;
        //Sets light terrain node type to true
        lightTerrainNode = true;
    }
    //Called when a medium terrain node is clicked
    public void MediumTerrainClicked()
    {
        //Sets all other node types to false
        openNode = false;
        blockedNode = false;
        lightTerrainNode = false;
        heavyTerrainNode = false;
        veryHeavyTerrainNode = false;
        //Sets medium terrain node type to true
        mediumTerrainNode = true;
    }
    //Called when an heavy terrain node is clicked
    public void HeavyTerrainClicked()
    {
        //Sets all other node types to false
        openNode = false;
        blockedNode = false;
        lightTerrainNode = false;
        mediumTerrainNode = false;
        veryHeavyTerrainNode = false;
        //Sets heavy terrain node type to true
        heavyTerrainNode = true;
    }
    //Called when a very heavy terrain node is clicked
    public void VeryHeavyTerrainClicked()
    {
        //Sets all other node types to false
        openNode = false;
        blockedNode = false;
        lightTerrainNode = false;
        mediumTerrainNode = false;
        heavyTerrainNode = false;
        //Sets very heavy terrain node type to true
        veryHeavyTerrainNode = true;
    }
}

//Called when custom maze menu is opened
public void CustomMazeLayout()
{
    //Stores the dimensions of the graph of nodes
    int width =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().width;
    int height =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().height;
```

```

//Creates a new gameobject for the nodes to be placed alongside
GameObject cellObject = new GameObject();

//Adds a sprite renderer component to the gameobject
cellObject.AddComponent<SpriteRenderer>().sprite = cellSprite;
//Adds a collider so the gameobject has set boundaries
cellObject.AddComponent<BoxCollider>();
//Adds a detector for when the node is clicked on
cellObject.AddComponent<OnClick>();

//Catch the size of the cellsprite
cellSize = cellSprite.bounds.size;

//Get the new cell size and adjust the size of the cells to fit the size of
the grid
Vector2 newCellSize = new Vector2(gridSize.x / (float)width, gridSize.y /
(float)height);

//Get the scales to scale the cells and change their size to fit the grid
cellScale.x = newCellSize.x / cellSize.x;
cellScale.y = newCellSize.y / cellSize.y;

//The original size cell object will be replaced by the new calculated size
cellSize = newCellSize;
cellObject.transform.localScale = new Vector2(cellScale.x, cellScale.y);

//Fix the cells to the grid by getting the dimensions of the grid so the cell
objects can fit
gridOffset.x = -(gridSize.x / 2) + cellSize.x / 2;
gridOffset.y = -(gridSize.y / 2) + cellSize.y / 2;

//Fill the grid with cells by using instantiate gameobject for each node
for (int row = 0; row < height; row++)
{
    for (int col = 0; col < width; col++)
    {
        //Add the cell size so that no two cells will have the same x and y
position
        Vector2 pos = new Vector2(col * cellSize.x + gridOffset.x +
transform.position.x, row * cellSize.y + gridOffset.y + transform.position.y);

        //Instantiate the game object, at position pos, with rotation set to
zero
        GameObject c0 = Instantiate(cellObject, pos, Quaternion.identity) as
GameObject;

        //Uniquely name each cell object relevant to its row and column
c0.name = "Node("+row+", "+col+")";

        //Set the parents of the cell object to the grid the user can move the
cells together with the grid
c0.transform.parent = transform;

        //Check for the start node coordinates
        if (col ==
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().startX &&
            row ==
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().startY)
        {
            //Change the start node coordinate to green
            Renderer rend = c0.GetComponent<SpriteRenderer>();

```

```

        rend.material.color = Color.green;
    }

    //Check for goal node coordinate
    if (col ==
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().goalX &&
        row ==
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().goalY)
    {
        //Change goal node coordinate to red
        Renderer rend = c0.GetComponent<SpriteRenderer>();
        rend.material.color = Color.red;
    }
}

}

//Destroy the original object used to instantiate the cells
Destroy(cellObject);
}
}
}

```

## Map Data Code:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Linq;
using System;
using UnityEngine.SceneManagement;
using System.IO;
using UnityEditor;

public class MapData : MonoBehaviour
{
    //Declaring all variable values
    public int width;
    public int height;
    public int presetNum;

    //Assigning a game object to variable
    public GameObject canvas;

    //Declaring the dimensions of a two dimensional array
    public int[,] MakeMap()
    {
        //Getting the width and height of the graph
        int width =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().width;
        int height =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().height;
        //Storing it as the dimensions of the (x,y) of the array
        int[,] map = new int[width, height];

        //For each node in the graph
        for (int i = 0; i < height; i++)
        {
            for (int j = 0; j < width; j++)
            {
                //Make each node appear open/white

```

```
        map[j, i] = 0;
    }
}
//Check if preset maze one is chosen
if
(GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().iPresetMaze1.isOn
== true)
{
    //Drawing the maze using preset node type values
    map[4, 0] = 10;
    map[6, 0] = 10;
    map[0, 1] = 10;
    map[2, 1] = 10;
    map[3, 1] = 10;
    map[4, 1] = 10;
    map[6, 1] = 10;
    map[8, 1] = 10;
    map[9, 1] = 10;
    map[10, 1] = 10;
    map[11, 1] = 10;
    map[12, 1] = 10;
    map[13, 1] = 10;
    map[13, 2] = 10;
    map[8, 2] = 10;
    map[11, 3] = 10;
    map[10, 3] = 10;
    map[9, 3] = 10;
    map[8, 3] = 10;
    map[7, 3] = 10;
    map[5, 3] = 10;
    map[4, 3] = 10;
    map[3, 3] = 10;
    map[2, 3] = 10;
    map[0, 3] = 10;
    map[0, 4] = 10;
    map[2, 4] = 10;
    map[4, 4] = 10;
    map[8, 4] = 10;
    map[13, 4] = 10;
    map[0, 5] = 10;
    map[2, 5] = 10;
    map[6, 5] = 10;
    map[8, 5] = 10;
    map[10, 5] = 10;
    map[11, 5] = 10;
    map[13, 5] = 10;
    map[13, 6] = 10;
    map[12, 6] = 10;
    map[11, 6] = 10;
    map[10, 6] = 10;
    map[9, 6] = 10;
    map[8, 6] = 10;
    map[7, 6] = 10;
    map[6, 6] = 10;
    map[4, 6] = 10;
    map[2, 6] = 10;
    map[0, 6] = 10;
    map[2, 7] = 10;
    map[4, 7] = 10;
    map[6, 7] = 10;
    map[13, 7] = 10;
    map[0, 8] = 10;
}
```

```
map[1, 8] = 10;
map[2, 8] = 10;
map[3, 8] = 10;
map[4, 8] = 10;
map[6, 8] = 10;
map[8, 8] = 10;
map[9, 8] = 10;
map[11, 8] = 10;
map[12, 8] = 10;
map[13, 8] = 10;
map[14, 8] = 10;
map[13, 9] = 10;
map[8, 9] = 10;
map[6, 9] = 10;
map[4, 9] = 10;
map[0, 10] = 10;
map[1, 10] = 10;
map[2, 10] = 10;
map[6, 10] = 10;
map[8, 10] = 10;
map[9, 10] = 10;
map[10, 10] = 10;
map[11, 10] = 10;
map[13, 10] = 10;
map[13, 11] = 10;
map[11, 11] = 10;
map[4, 11] = 10;
map[3, 11] = 10;
map[2, 11] = 10;
map[0, 12] = 10;
map[6, 12] = 10;
map[8, 12] = 10;
map[9, 12] = 10;
map[10, 12] = 10;
map[11, 12] = 10;
map[13, 13] = 10;
map[8, 13] = 10;
map[6, 13] = 10;
map[5, 13] = 10;
map[4, 13] = 10;
map[3, 13] = 10;
map[1, 13] = 10;
map[0, 13] = 10;
map[5, 14] = 10;
map[8, 14] = 10;
map[9, 14] = 10;
map[10, 14] = 10;
map[11, 14] = 10;
map[12, 14] = 10;
map[13, 14] = 10;
}
//Check if preset maze two is chosen
else if
(GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().iPresetMaze2.isOn
== true)
{
    //Drawing the maze using preset node type values
    map[12, 0] = 10;
    map[12, 1] = 10;
    map[12, 2] = 10;
    map[12, 3] = 10;
    map[12, 4] = 10;
```



```
map[12, 5] = 10;
map[12, 6] = 10;
map[12, 7] = 10;
map[12, 8] = 10;
map[12, 9] = 10;
map[12,10] = 10;
map[12,11] = 10;
map[12,13] = 10;
map[12, 12] = 10;
map[12, 14] = 10;
map[12, 15] = 10;
map[12, 16] = 10;
map[12, 17] = 10;
map[12, 18] = 10;
map[12, 19] = 10;
map[12, 20] = 10;
map[12, 21] = 10;
map[12, 23] = 10;
map[12, 24] = 10;

map[24, 1] = 10;

map[23, 1] = 10;
map[22, 1] = 10;
map[21, 1] = 10;
map[20, 1] = 10;
map[19, 1] = 10;
map[18, 1] = 10;
map[17, 1] = 10;
map[16, 1] = 10;
map[15, 1] = 10;
map[14, 1] = 10;
}
//Check if preset maze three is chosen
else if
(GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().iPresetMaze3.isOn
== true)
{
    //Drawing the maze using preset node type values (Draws vertical walls
every second x-value in graph)
    int incr = 1;
    for (int k = 0; k < 100; k+=2)
    {
        for (int l = 0; l < 100; l++)
        {
            map[k, l] = 10;
        }
        incr += 1;
        if (incr % 2 == 0)
        {
            map[k, 0] = 0;
        }
        else
        {
            map[k, 99] = 0;
        }
    }
    for (int m = 0; m < 100; m++)
    {
        map[m, 50] = 0;
    }
}
//Check if preset maze four is chosen
```

```
        else if
(GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().iPresetMaze4.isOn
== true)
    {
        //Drawing the maze using preset node type values with different weights
        for (int n = 0; n < 9; n+=2)
        {
            for (int l = 2; l < 48; l++)
            {
                map[l, n] = 10;
            }
        }
        for (int i = 2; i < 48; i++)
        {
            map[i, 1] = 4;
        }
        for (int i = 2; i < 48; i++)
        {
            map[i, 3] = 3;
        }
        for (int i = 2; i < 48; i++)
        {
            map[i, 5] = 2;
        }
    }

    //Check if random maze one is chosen
    else if
(GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().iRandomMaze1.isOn
== true)
    {
        //Generate a random number between 0 and 2
        System.Random rnd = new System.Random();
        for (int i = 0; i < height; i++)
        {
            for (int j = 0; j < width; j++)
            {
                int randomState = rnd.Next(2);
                //Check if the random number is zero (one in three)
                if (randomState == 0)
                {
                    //Make particular node a wall
                    map[j, i] = 10;
                }
            }
        }
    }
    //Check if random maze two is chosen
    else if
(GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().iRandomMaze2.isOn
== true)
    {
        //Generate a random number between 0 and 3
        System.Random rnd = new System.Random();
        for (int i = 0; i < height; i++)
        {
            for (int j = 0; j < width; j++)
            {
                int randomState = rnd.Next(3);
                //Check if the random number is zero (one in four)
                if (randomState == 0)
                {
```

```
        //Make particular node a wall
        map[j, i] = 10;
    }
}
}
}
//Check if random maze three is chosen
else if
(GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().iRandomMaze3.isOn
== true)
{
    //Generate a random number between 0 and 4
    System.Random rnd = new System.Random();
    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            int randomState = rnd.Next(4);
            //Check if the random number is zero (one in five)
            if (randomState == 0)
            {
                //Make particular node a wall
                map[j, i] = 10;
            }
        }
    }
}

//Check if custom maze is chosen
else if
(GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().iCustomMaze.isOn
== true)
{
    //Set the custom maze menu as active
    canvas.SetActive(true);

    //For each cell object in the custom maze grid, set the node type in the
graph view as the respective node type
    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            //Find the relevant cell object
            Renderer rend = GameObject.Find("Node(" + i + "," + j +
)")").GetComponent<SpriteRenderer>();
            //Check if the cell object is a wall and make the node in graph
view a wall
            if (rend.material.color == Color.black)
            {
                map[j, i] = 10;
            }
            //Check if the cell object is open and make the node in graph view
open
            else if (rend.material.color == new Color32(236, 198, 164, 255))
            {
                map[j, i] = 1;
            }
            //Check if the cell object is light terrain and make the node in
graph view light terrain
            else if (rend.material.color == new Color32(188, 142, 100, 255))
            {
                map[j, i] = 2;
            }
        }
    }
}
}
```

```

    }
    //Check if the cell object is medium terrain and make the node in
graph view medium terrain
    else if (rend.material.color == new Color32(148, 98, 60, 255))
    {
        map[j, i] = 3;
    }
    //Check if the cell object is heavy terrain and make the node in
graph view heavy terrain
    else if (rend.material.color == new Color32(100, 70, 44, 255))
    {
        map[j, i] = 4;
    }
    //Check if the cell object is very heavy terrain and make the node
in graph view very heavy terrain
    else
    {
        map[j, i] = 0;
    }
    }
}
//Set the custom maze menu as inactive
canvas.SetActive(false);
}

//Get the start(x,y) and goal(x,y) values from the user
int startX =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().startX;
int startY =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().startY;
int goalX =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().goalX;
int goalY =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().goalY;

//Make sure that the start and goal nodes are not a wall
map[startX, startY] = 0;
map[goalX, goalY] = 0;

//Return the map's values
return map;
}
}

```

## Menu Selector Code:

```

using System.Collections;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

//This class controls the function of the main menu
public class MenuSelector : MonoBehaviour {

    //Declares variables
    GameObject maze;
    GameObject instructions;
    GameObject descriptionObject;

    //Assigns game objects to each variable
    public GameObject backgroundImage;
    public GameObject mazeCanvas;
}

```

```
public GameObject menuCanvas;
public GameObject instructionCanvas;
public GameObject instructionCanvas2;
public GameObject pathfinderCanvas;

//Set the description text
Text description;

//Called when the program is initialised
void Start () {
    //Assigns game objects to each variable
    maze = GameObject.Find("p_maze");
    instructions = GameObject.Find("p_instructions");
    descriptionObject = GameObject.Find("Mode Description");

    //Gets the text for the description of menu titles
    description = descriptionObject.GetComponent<Text>();
    //Sets the description
    description.text = "Select one of the boxes above ";
}

//Update is called once per frame
void Update()
{
    //Rotate background slowly
    backgroundImage.transform.Rotate(0, 0, 0.1f);
}

//Called when the mouse hovers over a button
public void ButtonHover (Button button)
{
    //Check what button the mouse is above and sets the description accordingly
    if (button.name == "Button maze")
    {
        description.text = "Run through a maze using A* (A Star), Dijkstra's
Algorithm, Breadth First Search, and Greedy Best First Search";
    }
    else if (button.name == "Button instructions")
    {
        description.text = "Detailed explanation and instructions on each section
of the program.";
    }
    //Update the new description
    else
    {
        description.transform.SetAsLastSibling();
    }
}

//Called when the pathfinder option is clicked
public void MazePathfinder (Button button)
{
    //Sets the main menu as inactive and the variable settings screen as active
    menuCanvas.SetActive(false);
    mazeCanvas.SetActive(true);
}

//Called when the instructions option is clicked
public void Instruction(Button button)
{
    //Sets the main menu as inactive and instruction page as active
    menuCanvas.SetActive(false);
}
```

```

        instructionCanvas.SetActive(true);
    }

    public void InstructionPageBack(Button button)
    {
        //Sets the pathfinder screen as active and instructions page as inactive
        instructionCanvas2.SetActive(false);
        pathfinderCanvas.SetActive(true);
    }

    public void InstructionPage2(Button button)
    {
        //Sets the pathfinder screen as inactive and instructions page 2 as active
        pathfinderCanvas.SetActive(false);
        instructionCanvas2.SetActive(true);
    }

    //Called when the program is being closed
    public void Quit()
    {
        //Quits the program
        Application.Quit();
    }
}

```

## Move Background Code:

```

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

public class MoveBackground : MonoBehaviour
{
    public float maxSize;
    public float growFactor;
    public float waitTime;

    void Start()
    {
        StartCoroutine(Scale());
    }

    IEnumerator Scale()
    {
        float timer = 0;

        while (true) // this could also be a condition indicating "alive or dead"
        {
            // we scale all axis, so they will have the same value,
            // so we can work with a float instead of comparing vectors
            while (maxSize > transform.localScale.x)
            {
                timer += Time.deltaTime;
                transform.localScale += new Vector3(1, 1, 1) * Time.deltaTime *
growFactor;
                yield return null;
            }
            // reset the timer

            yield return new WaitForSeconds(waitTime);
        }
    }
}

```

```

        timer = 0;
        while (1 < transform.localScale.x)
        {
            timer += Time.deltaTime;
            transform.localScale -= new Vector3(1, 1, 1) * Time.deltaTime *
growFactor;
            yield return null;
        }

        timer = 0;
        yield return new WaitForSeconds(waitTime);
    }
}
}

```

## Node Code:

```

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

//Enum used to declare a list of node types with integer constants
public enum NodeType
{
    Open = 0, //Distance 1 metres
    Blocked = 10, //Not traversable
    //
    LightTerrain = 1, //Distance: 2 metres
    MediumTerrain = 2, //Distance: 3 metres
    HeavyTerrain = 3, //Distance: 4 metres
    veryHeavyTerrain = 4 //Distance: 5 metres
    //
}

//Node class implements the interface IComparable
public class Node: IComparable<Node>
{
    //Set the terrain type of the node
    public NodeType nodeType = NodeType.Open;

    //x and y index in the graph array
    public int xIndex = -1;
    public int yIndex = -1;

    //Vector 3 (x,y,z) position
    public Vector3 position;
    //List of neighbour nodes
    public List<Node> neighbours = new List<Node>();

    //Total distance travelled from the start node
    public float distanceTravelled = Mathf.Infinity;
    //Reference to the previous node as null in the graph
    public Node previous = null;
    //Priority to set placement of node in queue
    public float priority;

    //Initialise the variables when the node object is created
    public Node(int xIndex, int yIndex, NodeType nodeType)
    {

```

```

        this.xIndex = xIndex;
        this.yIndex = yIndex;
        this.nodeType = nodeType;
    }

    //Required for the IComparable comparison method to compare this node with another
    node based on priority
    public int CompareTo(Node other)
    {
        //Check if priority is less, more, or equal to the other node and return a
        value accordingly
        if (this.priority < other.priority)
        {
            return -1;
        }
        else if (this.priority > other.priority)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }

    //Set the preceding node to null
    public void Reset()
    {
        previous = null;
    }
}

```

## Node View Code:

```

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

//Class for managing the appearance of a node
public class NodeView : MonoBehaviour
{
    //Assigning game objects to variables
    public GameObject tile;
    public GameObject arrow;

    //Corresponding node
    Node m_node;

    //Declaring the default size between nodes
    public float borderSize = 0.1f;

    //Initialise the NodeView with the corresponding node
    public void Init(Node node)
    {
        //Sets the border size to what the user has set
        borderSize =
        GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().borderSize;
        //Check is node tile exists
        if (tile != null)
        {
            //Name the tile according to its (x,y) position

```



```

        gameObject.name = "Node (" + node.xIndex + "," + node.yIndex + ")";
        gameObject.transform.position = node.position;
        //Change the scale based on border size
        tile.transform.localScale = new Vector3(1f - borderSize, 1f, 1f -
borderSize);
        m_node = node;
        //Disable arrows
        EnableObject(arrow, false);
    }
}

//Method to colour the tile gameobject
void ColorNode(Color color, GameObject go)
{
    //Check if the game object exists
    if (go != null)
    {
        //Get the renderer of the game object
        Renderer goRenderer = go.GetComponent<Renderer>();
        //Check if renderer exists
        if (goRenderer != null)
        {
            //Set the colour of the game object to the specified colour
            goRenderer.material.color = color;
        }
    }
}

//Method to colour the tile gameobject
public void ColorNode(Color color)
{
    ColorNode(color, tile);
}

//Generic method to toggle a game object on and off
void EnableObject(GameObject go, bool state)
{
    if(go != null)
    {
        go.SetActive(state);
    }
}

//Show the arrow with a relevant color
public void ShowArrow(Color color)
{
    // verify there is a corresponding node, arrow model, and a target for the
arrow
    if (m_node != null && arrow != null && m_node.previous != null)
    {
        //Turn the arrow geometry on
        EnableObject(arrow, true);

        //Calculate the arrows direction to point to its next node
        Vector3 dirToPrevious = (m_node.previous.position -
m_node.position).normalized;

        //Rotate the arrow towards the next node
        arrow.transform.rotation = Quaternion.LookRotation(dirToPrevious);

        //Change the colour of the arrow
        Renderer arrowRenderer = arrow.GetComponent<Renderer>();

```

```

        if (arrowRenderer != null)
        {
            arrowRenderer.material.color = color;
        }
    }
}

```

## On Node Click Code:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

//Class to colour in the nodes in the custom maze
public class OnNodeClick : MonoBehaviour
{
    //Declare all variables
    public bool openNode;
    public bool blockedNode;
    public bool lightNode;
    public bool mediumNode;
    public bool heavyNode;
    public bool veryHeavyNode;
    public bool is_down;

    //Called once per frame
    private void Update()
    {
        //Check if left mouse button is down
        if (Input.GetMouseButton(0))
        {
            //Set mouse down as true
            is_down = true;
        }
        else
        {
            //Set mouse down to false
            is_down = false;
        }
    }

    //Called when the mouse is over a gameobject
    public void OnMouseOver()
    {
        //Check if mouse is held down
        if (is_down == true)
        {
            //Find which node type is set to true
            openNode = GameObject.Find("GridHolder").GetComponent<GridSet>().openNode;
            blockedNode =
GameObject.Find("GridHolder").GetComponent<GridSet>().blockedNode;
            lightNode =
GameObject.Find("GridHolder").GetComponent<GridSet>().lightTerrainNode;
            mediumNode =
GameObject.Find("GridHolder").GetComponent<GridSet>().mediumTerrainNode;
            heavyNode =
GameObject.Find("GridHolder").GetComponent<GridSet>().heavyTerrainNode;
            veryHeavyNode =
GameObject.Find("GridHolder").GetComponent<GridSet>().veryHeavyTerrainNode;

```

```

//Get the component to colour the game object
Renderer rend = gameObject.GetComponent<SpriteRenderer>();

//Check if the node type open node is true and colour cell object white
if (openNode == true && rend.material.color != Color.green &&
rend.material.color != Color.red)
{
    rend.material.color = Color.white;
}
//Check if the node type blocked node is true and colour cell object black
else if (blockedNode == true && rend.material.color != Color.green &&
rend.material.color != Color.red)
{
    rend.material.color = Color.black;
}
//Check if the node type light terrain node is true and colour cell object
very light brown
else if (lightNode == true && rend.material.color != Color.green &&
rend.material.color != Color.red)
{
    rend.material.color = new Color32(236, 198, 164, 255);
}
//Check if the node type medium terrain node is true and colour cell
object light brown
else if (mediumNode == true && rend.material.color != Color.green &&
rend.material.color != Color.red)
{
    rend.material.color = new Color32(188, 142, 100, 255);
}
//Check if the node type heavy terrain node is true and colour cell object
brown
else if (heavyNode == true && rend.material.color != Color.green &&
rend.material.color != Color.red)
{
    rend.material.color = new Color32(148, 98, 60, 255);
}
//Check if the node type very heavy terrain node is true and colour cell
object dark brown
else if (veryHeavyNode == true && rend.material.color != Color.green &&
rend.material.color != Color.red)
{
    rend.material.color = new Color32(100, 70, 44, 255);
}
}
}
}

```

## Pathfinder Code:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.IO;
using System.Linq;
using System.Timers;

//This class controls the pathfinding of the program
public class Pathfinder : MonoBehaviour
{

```

```
//The starting node
Node m_startNode;

//The goal node
Node m_goalNode;

//Graph and graphview components
Graph m_graph;
GraphView m_graphView;

//The 'open' set of nodes that are to be explored
PriorityQueue<Node> m_frontierNodes;

//The 'closed' set of Nodes that have already been explored
List<Node> m_exploredNodes;

//The list of nodes that make up the path from the start to goal node
List<Node> m_pathNodes;

//Setting the colour of each type of node so the user can identify them
public Color startColor = Color.green;
public Color goalColor = Color.red;
public Color frontierColor;
public Color exploredColor = Color.grey;
public Color pathColor;
public Color arrowColor = new Color(0.85f, 0.85f, 0.85f, 1f); //Light grey Color
public Color highlightColor;

//Declaring all user settings as their default values
//Show the pathfinder as it goes through the maze
public bool showIterations = true;
//Show the colours of the nodes
public bool showColors = true;
//Show the arrows showing the path and explored nodes
public bool showArrows = true;
//Should the pathfinder exit when the goal is found
public bool exitOnGoal = true;
//Setting the goal as not found by default
public bool isComplete = false;

//Number of iterations used
int m_iterations = 0;

//Assigning gameobjects to each variable
public GameObject ChooseAnother;
public GameObject ToggleInformation;

//Text to display information about the pathfinders
public Text Information;
public Text Parameters;
public Text textDijkstra;
public Text textAStar;
public Text textBreadth;
public Text textGreedy;

//Setting the default time of all the algorithms
public float DijkstraTime = 999999;
public float AStarTime = 999999;
public float BreadthTime = 999999;
public float GreedyTime = 999999;

//Setting the default distance of all the algorithms
```

```
public float DijkstraDistance = 999999;
public float AStarDistance = 999999;
public float BreadthDistance = 999999;
public float GreedyDistance = 999999;

//Setting the default speed of all algorithms
public float DijkstraSpeed;
public float AStarSpeed;
public float BreadthSpeed;
public float GreedySpeed;

//Booleans to check if specific algorithm has been run
public bool completeDijkstra = false;
public bool completeAStar = false;
public bool completeBreadth = false;
public bool completeGreedy = false;

//Set-up the stopwatch
public System.Diagnostics.Stopwatch stopwatch;

//All the pathfinding algorithms
public enum Mode
{
    BreadthFirst = 0,
    Dijkstra = 1,
    GreedyBestFirst = 2,
    AStar = 3
}

//Default the pathfinder as Breadth First Search
public Mode mode = Mode.BreadthFirst;

//Called when the pathfinder is initialised
public void Init(Graph graph, GraphView graphView, Node start, Node goal)
{
    //Error check for if the start,goal, graph or graphview is missing
    if (start == null || goal == null || graph == null || graphView == null)
    {
        Debug.LogWarning("PATHFINDER Init error: missing component(s)");
        return;
    }
    //Ensure that the start and goal nodes are not a wall node type
    if (start.nodeType == NodeType.Blocked || goal.nodeType == NodeType.Blocked)
    {
        Debug.LogWarning("PATHFINDER Init error: start and goal nodes must be
unblocked");
        return;
    }

    //Store the graph, graphview, starting and goal node
    m_graph = graph;
    m_graphView = graphView;
    m_startNode = start;
    m_goalNode = goal;

    //Draw the colours of the map
    ShowColors(graphView, start, goal);

    //Begin the frontier node list with only the start node
    m_frontierNodes = new PriorityQueue<Node>();
    m_frontierNodes.Enqueue(start);
}
```

```

//Initialise the explored and path node as empty lists
m_exploredNodes = new List<Node>();
m_pathNodes = new List<Node>();

//Reset all nodes in the graph
for (int x = 0; x < m_graph.Width; x++)
{
    for (int y = 0; y < m_graph.Height; y++)
    {
        m_graph.nodes[x, y].Reset();
    }
}

//Setup the starting values
isComplete = false;
m_iterations = 0;
m_startNode.distanceTravelled = 0;
}

//Show colours in the graph view
void ShowColors(bool lerpColor = false, float lerpValue = 0.5f)
{
    ShowColors(m_graphView, m_startNode, m_goalNode, lerpColor, lerpValue);
}
void ShowColors(GraphView graphView, Node start, Node goal, bool lerpColor =
false, float lerpValue = 0.5f)
{
    //Error check if graph view, start or goal node does not exist
    if(graphView == null || start == null || goal == null)
    {
        return;
    }
    //Colour all the frontier nodes
    if (m_frontierNodes != null)
    {
        graphView.ColorNodes(m_frontierNodes.ToList(), frontierColor, lerpColor,
lerpValue);
    }
    //Colour all the explored nodes
    if (m_exploredNodes != null)
    {
        graphView.ColorNodes(m_exploredNodes, exploredColor, lerpColor,
lerpValue);
    }
    //Colour all the path nodes
    if (m_pathNodes != null && m_pathNodes.Count > 0)
    {
        graphView.ColorNodes(m_pathNodes, pathColor, lerpColor, lerpValue * 2);
    }

    //Colour the start nodeview directly
    NodeView startNodeView = graphView.nodeViews[start.xIndex, start.yIndex];
    if (startNodeView != null)
    {
        startNodeView.ColorNode(startColor);
    }

    //Colour the start nodeview directly
    NodeView goalNodeView = graphView.nodeViews[goal.xIndex, goal.yIndex];
    if (goalNodeView != null)
    {

```

```
        goalNodeView.ColorNode(goalColor);
    }
}

//The main pathfinding algorithm search methods
public IEnumerator SearchRoutine(float timeStep = 0.01f)
{
    //Reset the stopwatch
    stopwatch = new System.Diagnostics.Stopwatch();

    //Check if Dijkstra search is selected and set the pathfinding mode
    if
(GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().iDijkstra.isOn ==
true)
    {
        mode = Mode.Dijkstra;
    }
    //Check if Dijkstra search is selected and set the pathfinding mode
    else if
(GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().iAStar.isOn ==
true)
    {
        mode = Mode.AStar;
    }
    //Check if Dijkstra search is selected and set the pathfinding mode
    else if
(GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().iBreadthFirstSearch.isOn == true)
    {
        mode = Mode.BreadthFirst;
    }
    //Check if Dijkstra search is selected and set the pathfinding mode
    else
    {
        mode = Mode.GreedyBestFirst;
    }
    //Reset the stopwatch time
    stopwatch.Reset();
    //Start the stopwatch timer
    stopwatch.Start();

    //Wait until the next frame
    yield return null;

    //While the pathfinder has not found the goal
    while (!isComplete)
    {
        //Check if there are still open nodes to explore
        if (m_frontierNodes.Count > 0)
        {
            //Get the next available open node from the priority queue
            Node currentNode = m_frontierNodes.Dequeue();
            //Increment the iteration counter
            m_iterations++;

            //Mark the current node as explored
            if(!m_exploredNodes.Contains(currentNode))
            {
                m_exploredNodes.Add(currentNode);
            }

            //Check if the pathfinder mode selected is breadth first search
```

```

if(mode == Mode.BreadthFirst)
{
    //Search using breadth first search
    ExpandFrontierBreadthFirstSearch(currentNode);
}
//Check if the pathfinder mode selected is dijkstra's algorithm
else if (mode == Mode.Dijkstra)
{
    //Search using dijkstra's algorithm
    ExpandFrontierDijkstra(currentNode);
}
//Check if the pathfinder mode selected is greedy best first search
else if (mode == Mode.GreedyBestFirst)
{
    //Search using greedy best first search
    ExpandFrontierGreedyBestFirst(currentNode);
}
//Check if the pathfinder mode selected is A Star search
else
{
    //Search using A Star search
    ExpandFrontierAStar(currentNode);
}

//Check if the list of frontier nodes includes the goal node
if (m_frontierNodes.Contains(m_goalNode))
{
    //Set the path nodes
    m_pathNodes = GetPathNodes(m_goalNode);
    //Set the exitOnGoal variable to what the user has chosen on the
settings page
    exitOnGoal =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().ExitOnGoal;
    //Check if exitOnGoal is turned on
    if (exitOnGoal)
    {
        //Mark the search as complete
        isComplete = true;
    }
    if (!exitOnGoal)
    {
        //Pause the timer when the goal is found if exitOnGoal is set
to false
        stopwatch.Stop();
    }
}
//Check if show iterations setting is true
showIterations =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().showIterations;
if (showIterations)
{
    //Colour each node individually
    ShowDiagnostics(true, 0.5f);
    //Get the users inputted value for time step variable
    timeStep =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().timeStep;
    //Wait a specified amount of time before exploring another node
    yield return new WaitForSeconds(timeStep);
}
}
else
{

```



```

        //Mark the search as complete if there are no more frontier nodes
        isComplete = true;
    }
}
//Colour all nodes
ShowDiagnostics(true,0.5f);
//Stop the stopwatch timer
stopwatch.Stop();

//Check if the pathfinder has run with a valid mode
if (!Information.text.Contains(mode.ToString()))
{
    //Output the pathfinder mode
    Information.text += "\n\n<size=42>Pathfinder Mode:\n<i>" +
mode.ToString() + "</i></size>";
    //Check for if no path was possible
    if(m_goalNode.distanceTravelled.ToString() == "Infinity")
    {
        //Fill in the text box with time, and nodes visited
        Information.text += "\n\n<i>There is no possible\ncpath</i>";
        Information.text += "\n\nTime taken:\n<i>" +
((float)stopwatch.ElapsedTicks / 10000000) + " seconds</i>";
        Information.text += "\n\nNodes explored:\n<i>" + (m_iterations)+
nodes</i>";
    }
    //Check if a path was found
    else
    {
        //Fill in the text box with time, distance, speed, and nodes visited
        Information.text += "\n\nPath Length:\n<i>" +
m_goalNode.distanceTravelled.ToString() + " metres</i>";
        Information.text += "\n\nTime taken:\n<i>" +
((float)stopwatch.ElapsedTicks / 10000000) + " seconds</i>";
        Information.text += "\n\nAverage Speed:\n<i>" +
(m_goalNode.distanceTravelled / ((float)stopwatch.ElapsedTicks / 10000000)) + " metres
per\nsecond</i>";
        Information.text += "\n\nNodes explored:\n<i>" + (m_iterations) + "
nodes</i>";
    }
    //Check if dijkstra pathfinder mode was run
    if (mode == Mode.Dijkstra)
    {
        completeDijkstra = true;
        //Output the pathfinder mode in the dijkstra textbox
        textDijkstra.text = "<size=42>Dijkstra's Algorithm:</size>";
        //Check for if no path was possible
        if (m_goalNode.distanceTravelled.ToString() == "Infinity")
        {
            //Fill in the text box with time in the dijkstra textbox
            textDijkstra.text += "\n\n<i>No Possible Path</i>";
            textDijkstra.text += "\n\nTime taken:\n<i>" +
((float)stopwatch.ElapsedTicks / 10000000) + " seconds</i>";
            DijkstraTime = ((float)stopwatch.ElapsedTicks / 10000000);
        }
        else
        {
            //Fill in the text box with time, distance, and speed in the
dijkstra textbox
            textDijkstra.text += "\n\nPath Length:\n<i>" +
m_goalNode.distanceTravelled.ToString() + " metres</i>";
            textDijkstra.text += "\n\nTime taken:\n<i>" +
((float)stopwatch.ElapsedTicks / 10000000) + " seconds</i>";

```

```

        textDijkstra.text += "\nAverage Speed:\n<i>" +
(m_goalNode.distanceTravelled / ((float)stopwatch.ElapsedTicks / 10000000)) + "
m/s</i>";
        DijkstraDistance =
float.Parse(m_goalNode.distanceTravelled.ToString());
        DijkstraTime = ((float)stopwatch.ElapsedTicks / 10000000);
        DijkstraSpeed = (m_goalNode.distanceTravelled /
((float)stopwatch.ElapsedTicks / 10000000));
    }
}
else if (mode == Mode.AStar)
{
    completeAStar = true;
    //Output the pathfinder mode in the A Star textbox
    textAStar.text = "<size=42>A* (A Star):</size>";
    //Check for if no path was possible
    if (m_goalNode.distanceTravelled.ToString() == "Infinity")
    {
        //Fill in the text box with time in the A Star textbox
        textAStar.text += "\n<i>No Possible Path</i>";
        textAStar.text += "\nTime taken:\n<i>" +
((float)stopwatch.ElapsedTicks / 10000000) + " seconds</i>";
        AStarTime = ((float)stopwatch.ElapsedTicks / 10000000);
    }
    else
    {
        //Fill in the text box with time, distance, and speed in the A
Star textbox
        textAStar.text += "\nPath Length:\n<i>" +
m_goalNode.distanceTravelled.ToString() + " metres</i>";
        textAStar.text += "\nTime taken:\n<i>" +
((float)stopwatch.ElapsedTicks / 10000000) + " seconds</i>";
        textAStar.text += "\nAverage Speed:\n<i>" +
(m_goalNode.distanceTravelled / ((float)stopwatch.ElapsedTicks / 10000000)) + "
m/s</i>";
        AStarDistance =
float.Parse(m_goalNode.distanceTravelled.ToString());
        AStarTime = ((float)stopwatch.ElapsedTicks / 10000000);
        AStarSpeed = (m_goalNode.distanceTravelled /
((float)stopwatch.ElapsedTicks / 10000000));
    }
}
else if (mode == Mode.BreadthFirst)
{
    completeBreadth = true;
    //Output the pathfinder mode in the breadth first textbox
    textBreadth.text = "<size=42>Breadth First Search:</size>";
    //Check for if no path was possible
    if (m_goalNode.distanceTravelled.ToString() == "Infinity")
    {
        //Fill in the text box with time in the breadth first search
textbox
        textBreadth.text += "\n<i>No Possible Path</i>";
        textBreadth.text += "\nTime taken:\n<i>" +
((float)stopwatch.ElapsedTicks / 10000000) + " seconds</i>";
        BreadthTime = ((float)stopwatch.ElapsedTicks / 10000000);
    }
    else
    {
        //Fill in the text box with time, distance, and speed in the
breadth first search textbox

```

```

        textBreadth.text += "\nPath Length:\n<i>" +
m_goalNode.distanceTravelled.ToString() + " metres</i>";
        textBreadth.text += "\nTime taken:\n<i>" +
((float)stopwatch.ElapsedTicks / 10000000) + " seconds</i>";
        textBreadth.text += "\nAverage Speed:\n<i>" +
(m_goalNode.distanceTravelled / ((float)stopwatch.ElapsedTicks / 10000000)) + "
m/s</i>";
        BreadthDistance =
float.Parse(m_goalNode.distanceTravelled.ToString());
        BreadthTime = ((float)stopwatch.ElapsedTicks / 10000000);
        BreadthSpeed = (m_goalNode.distanceTravelled /
((float)stopwatch.ElapsedTicks / 10000000));
    }
}
else if (mode == Mode.GreedyBestFirst)
{
    completeGreedy = true;
    //Output the pathfinder mode in the greedy best first textbox
    textGreedy.text = "<size=42>Greedy Best First:</size>";
    //Check for if no path was possible
    if (m_goalNode.distanceTravelled.ToString() == "Infinity")
    {
        //Fill in the text box with time in the greedy best first textbox
        textGreedy.text += "\n<i>No Possible Path</i>";
        textGreedy.text += "\nTime taken:\n<i>" +
((float)stopwatch.ElapsedTicks / 10000000) + " seconds</i>";
        GreedyTime = ((float)stopwatch.ElapsedTicks / 10000000);
    }
    else
    {
        //Fill in the text box with time, distance, and speed in the
greedy best first textbox
        textGreedy.text += "\nPath Length:\n<i>" +
m_goalNode.distanceTravelled.ToString() + " metres</i>";
        textGreedy.text += "\nTime taken:\n<i>" +
((float)stopwatch.ElapsedTicks / 10000000) + " seconds</i>";
        textGreedy.text += "\nAverage Speed:\n<i>" +
(m_goalNode.distanceTravelled / ((float)stopwatch.ElapsedTicks / 10000000)) + "
m/s</i>";
        GreedyDistance =
float.Parse(m_goalNode.distanceTravelled.ToString());
        GreedyTime = ((float)stopwatch.ElapsedTicks / 10000000);
        GreedySpeed = (m_goalNode.distanceTravelled /
((float)stopwatch.ElapsedTicks / 10000000));
    }
}
//Check if all pathfinders have run at least once
if (completeDijkstra == true && completeAStar == true && completeBreadth
== true && completeGreedy == true)
{
    //Call function to write information to external text file
    CreateText();
}

}
//Allow the user to choose another pathfinder to run
ChooseAnother.SetActive(true);
//Check if more than one pathfinder has been run
if (GameObject.Find("DemoController").GetComponent<DemoController>().pass ==
1)
{
    //Keep information up

```

```

        ToggleInformation.SetActive(true);
    }
    //Check if the first pathfinder is being run
    if (GameObject.Find("DemoController").GetComponent<DemoController>().pass ==
0)
    {
        //Sort result in order of ascending distance by default
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().SortDistanceUpdate
();
    }
}

//Method called to show arrows and colours
private void ShowDiagnostics(bool lerpColor = false, float lerpValue = 0.5f)
{
    if (showColors)
    {
        //Show colours on the graph view
        ShowColors(lerpColor, lerpValue);
    }
    //Check if user has set showArrows to true
    showArrows =
GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().showArrows;
    if (m_graphView != null && showArrows)
    {
        //Show arrows from the frontier in the arrow colour
        m_graphView.ShowNodeArrows(m_frontierNodes.ToList(), arrowColor);

        //Check if the goal node has been reached
        if (m_frontierNodes.Contains(m_goalNode))
        {
            //Show the arrows indicating the correct path
            m_graphView.ShowNodeArrows(m_pathNodes, highlightColor);
        }
    }
}

//Expand the frontier nodes using dijkstra's algorithm
void ExpandFrontierDijkstra(Node node)
{
    if (node != null)
    {
        //Loop through all the neighbour nodes
        for (int i = 0; i < node.neighbours.Count; i++)
        {
            //Check if the current neighbor has not been explored
            if (!m_exploredNodes.Contains(node.neighbours[i]))
            {
                //Get distance values to neighbour and total distance
                float distanceToNeighbour = m_graph.GetNodeDistance(node,
node.neighbours[i]);
                float newDistanceTravelled = distanceToNeighbour +
node.distanceTravelled +(int)node.nodeType;

                //Check if a shorter path to the neighbour is available via this
node and re-route
                if (float.IsPositiveInfinity(node.neighbours[i].distanceTravelled)
||
                    newDistanceTravelled < node.neighbours[i].distanceTravelled)
                {
                    node.neighbours[i].previous = node;

```

```

        node.neighbours[i].distanceTravelled = newDistanceTravelled;
    }
    //Check if the current neighbour is not a part of the frontier
nodes
    if (!m_frontierNodes.Contains(node.neighbours[i]))
    {
        //Set the priority based on distance travelled from start node
and add to frontier
        node.neighbours[i].priority =
node.neighbours[i].distanceTravelled;
        //Distance travelled is what determines priority of the node
        m_frontierNodes.Enqueue(node.neighbours[i]);
    }
}
}
}

//Expand the frontier nodes using breadth first search
void ExpandFrontierBreadthFirstSearch(Node node)
{
    if (node != null)
    {
        //Loop through all the neighbour nodes
        for (int i = 0; i < node.neighbours.Count; i++)
        {
            //Check if the current neighbor has not been explored
            if (!m_exploredNodes.Contains(node.neighbours[i])
                && !m_frontierNodes.Contains(node.neighbours[i]))
            {
                //Get distance values to neighbour and total distance
                float distanceToNeighbour = m_graph.GetNodeDistance(node,
node.neighbours[i]);
                float newDistanceTravelled = distanceToNeighbour +
node.distanceTravelled + (int)node.nodeType;
                //Create a trail of nodes visited and update distance travelled
                node.neighbours[i].distanceTravelled = newDistanceTravelled;
                node.neighbours[i].previous = node;

                //Add the neighbour node to explored nodes queue
                node.neighbours[i].priority = m_exploredNodes.Count;
                m_frontierNodes.Enqueue(node.neighbours[i]);
            }
        }
    }
}

//Expand the frontier nodes using greedy best first search
void ExpandFrontierGreedyBestFirst(Node node)
{
    if (node != null)
    {
        //Loop through all the neighbour nodes
        for (int i = 0; i < node.neighbours.Count; i++)
        {
            //Check if the current neighbor has not been explored
            if (!m_exploredNodes.Contains(node.neighbours[i])
                && !m_frontierNodes.Contains(node.neighbours[i]))
            {
                //Get distance values to neighbour and total distance
                float distanceToNeighbour = m_graph.GetNodeDistance(node,
node.neighbours[i]);

```

```

        float newDistanceTravelled = distanceToNeighbour +
node.distanceTravelled + (int)node.nodeType;

        //Create a trail of nodes visited and update distance travelled
        node.neighbours[i].distanceTravelled = newDistanceTravelled;
        node.neighbours[i].previous = node;

        //Set the priority of a neighbour node based on distance to the
goal node
        if (m_graph != null)
        {
            node.neighbours[i].priority =
m_graph.GetNodeDistance(node.neighbours[i], m_goalNode);
        }
        m_frontierNodes.Enqueue(node.neighbours[i]);
    }
}

//Expand the frontier nodes using A Star search
void ExpandFrontierAStar(Node node)
{
    if (node != null)
    {
        //Loop through all the neighbour nodes
        for (int i = 0; i < node.neighbours.Count; i++)
        {
            //Check if the current neighbor has not been explored
            if (!m_exploredNodes.Contains(node.neighbours[i]))
            {
                //Get distance values to neighbour and total distance
                float distanceToNeighbour = m_graph.GetNodeDistance(node,
node.neighbours[i]);
                float newDistanceTravelled = distanceToNeighbour +
node.distanceTravelled + (int)node.nodeType;

                //Check if a shorter path exists to the neighbour and re-route the
path
                if (float.IsPositiveInfinity(node.neighbours[i].distanceTravelled)
||
                    newDistanceTravelled < node.neighbours[i].distanceTravelled)
                {
                    node.neighbours[i].previous = node;
                    node.neighbours[i].distanceTravelled = newDistanceTravelled;
                }
                //Check if a neighbor is not part of the frontier list and add it
to the priority queue
                if (!m_frontierNodes.Contains(node.neighbours[i]) && m_graph !=
null)
                {
                    //Priority queue order is Score = (distance from start) +
(estimated distance to goal)
                    float distanceToGoal =
m_graph.GetNodeDistance(node.neighbours[i], m_goalNode);
                    node.neighbours[i].priority =
node.neighbours[i].distanceTravelled
                        + distanceToGoal;
                    m_frontierNodes.Enqueue(node.neighbours[i]);
                }
            }
        }
    }
}

```

```

    }
}

//Generate list of path Nodes by traversing backwards from the goal Node
List<Node> GetPathNodes(Node endNode)
{
    List<Node> path = new List<Node>();
    if(endNode == null)
    {
        return path;
    }
    //Begin at the goal node
    path.Add(endNode);

    //Follow the trail of arrows back to the start node
    Node currentNode = endNode.previous;

    while (currentNode != null)
    {
        //Insert the previous node at the beginning of the path
        path.Insert(0, currentNode);
        //Continue traversing backwards through graph
        currentNode = currentNode.previous;
    }
    //Return the list of path nodes
    return path;
}

//Called to write pathfinder results onto a text file
public void CreateText()
{
    //Declaring the path of the file
    string path = Application.dataPath + "../Log.txt";
    //Create File if it doesn't exist
    if (!File.Exists(path))
    {
        File.WriteAllText(path, "\\\\\\\\\\\\\\\\ Pathfinder Log: /////");
    }
    Parameters.text = Parameters.text.Replace("Dijkstra's Algorithm", "");
    Parameters.text = Parameters.text.Replace("Greedy Best First", "");
    Parameters.text = Parameters.text.Replace("A* (A Star) Search", "");
    Parameters.text = Parameters.text.Replace("Breadth First Search", "");
    //Setting the Content of the file
    string content = "\n\n\\\\\\\\\\\\ (New Maze) Log date: " + System.DateTime.Now +
    "////\n\n\\\\\\\\ Parameters: //" + Parameters.text + "\n\\\\\\\\ Results: //" +
    Information.text;
    //Replacing unnecessary characters
    content = content.Replace("<i>", "");
    content = content.Replace("</i>", "");
    content = content.Replace("<size=42>", "");
    content = content.Replace("<size=40>", "");
    content = content.Replace("</size>", "");
    //Add the content to the file
    File.AppendAllText(path, content);
}
}

```

## Pause Code:

```

using System.Collections;
using System.Collections.Generic;

```

```

using UnityEngine;
using UnityEngine.UI;

//This class is to pause and resume the project
public class Pause : MonoBehaviour
{
    //Assigning game objects to each variable
    public GameObject pauseButton;
    public GameObject playButton;
    public GameObject selectPatfinder;

    //Called when the pause button is clicked
    public void pauseGame()
    {
        //Stops the pathfinding timer
        GameObject.Find("Pathfinder").GetComponent<Pathfinder>().stopwatch.Stop();
        //Pauses the in project time
        Time.timeScale = 0;
        //Sets the play button as visible and pause button as hidden
        pauseButton.SetActive(false);
        playButton.SetActive(true);
    }

    //Called when the play button is clicked
    public void continueGame()
    {
        //Starts the pathfinding timer
        GameObject.Find("Pathfinder").GetComponent<Pathfinder>().stopwatch.Start();
        //Resumes the in project time
        Time.timeScale = 1;
        //Sets the play button as visible and pause button as hidden
        playButton.SetActive(false);
        pauseButton.SetActive(true);
    }
}

```

## Priority Queue Code:

```

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

//Priority queue implemented using a list with binary heap (binary tree)
public class PriorityQueue<T> where T : IComparable<T>
{
    //List the nodes in the queue
    List<T> data;

    //Number of nodes in the queue (read-only)
    public int Count { get { return data.Count; } }

    //Called to initialise the list
    public PriorityQueue()
    {
        this.data = new List<T>();
    }

    //Called to add a node to the priority queue and sort using a binary tree
    public void Enqueue(T item)

```



```
{
    //Add the node at the end of the list
    data.Add(item);

    //Start at the last position in the binary tree
    int childindex = data.Count - 1;

    //Sort using a minum binary tree
    while (childindex > 0)
    {
        //Find the parent index of the child node
        int parentindex = (childindex - 1) / 2;
        //Check if the parent and child are already sorted, if so then break out
the while loop
        if(data[childindex].CompareTo(data[parentindex]) >= 0)
        {
            break;
        }
        //Otherwise swap the parent and child positions
        T tmp = data[childindex];
        data[childindex] = data[parentindex];
        data[parentindex] = tmp;

        //Move up one level in the binary tree
        childindex = parentindex;
    }
}

//Remove a node from the priority queue and sort
public T Dequeue()
{
    //Get the index of the last node
    int lastindex = data.Count - 1;

    //Store the first node in the list
    T frontItem = data[0];

    //Replace the first node with the last node
    data[0] = data[lastindex];

    //Remove the last index of the priority queue
    data.RemoveAt(lastindex);

    //Decrement node count by one
    lastindex--;

    //Start at the beginning of the queue
    int parentindex = 0;

    //Sort using binary heap (binary tree)
    while (true)
    {
        //Choose the left child node in the binary tree
        int childindex = parentindex * 2 + 1;

        //Check if there is no left child node, if not then break the while loop
        if (childindex > lastindex)
        {
            break;
        }

        //Store the right child
```

```

        int rightchild = childindex + 1;

        //Check if the value of the right child is less than the left child
        if (rightchild <= lastindex &&
data[rightchild].CompareTo(data[childindex]) < 0)
        {
            //Switch to the right child node of the binary tree
            childindex = rightchild;
        }

        //Check if the parent and child are already sorted, if so then break out
the while loop
        if (data[parentindex].CompareTo(data[childindex]) <= 0)
        {
            break;
        }

        //Otherwise swap the parent and child positions
        T tmp = data[parentindex];
        data[parentindex] = data[childindex];
        data[childindex] = tmp;

        //Move down one level in the binary tree
        parentindex = childindex;
    }

    //Return the original first node
    return frontItem;
}

//Look at the first node without dequeuing
public T Peek()
{
    T frontItem = data[0];
    return frontItem;
}

//Check if a node is in the list
public bool Contains(T item)
{
    return data.Contains(item);
}

//Return the data as a list
public List<T> ToList()
{
    return data;
}
}

```

## Settings Code:

```

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

//This class controls all menus, GUIs, and user settings
public class Settings : MonoBehaviour
{

```

```
//Declaring all default variable values
public InputField iWidth;
public int width = 25;
public InputField iHeight;
public int height = 25;
public InputField iStartX;
public int startX = 0;
public InputField iStartY;
public int startY = 0;
public InputField iGoalX;
public int goalX = 24;
public InputField iGoalY;
public int goalY = 24;
public InputField iBorderSize;
public int temp;
public int tempX;
public int tempY;

public float borderSize = 0.1f;
public float tempf;

public bool Pass = false;
public bool zoom = false;

//Assigning toggle buttons to each variable
public Toggle iShowArrows;
public bool showArrows;
public Toggle iExitOnGoal;
public bool ExitOnGoal;
public Toggle iShowIterations;
public bool showIterations;
public Toggle iEightDirections;
public bool eightDirections;
public InputField iTimeStep;
public float timeStep = 0.01f;

public Toggle iPresetMaze;
public Toggle iRandomMaze;
public Toggle iCustomMaze;

public Toggle iPresetMaze1;
public Toggle iPresetMaze2;
public Toggle iPresetMaze3;
public Toggle iPresetMaze4;

public Toggle iRandomMaze1;
public Toggle iRandomMaze2;
public Toggle iRandomMaze3;

public Toggle iDijkstra;
public Toggle iAStar;
public Toggle iBreadthFirstSearch;
public Toggle iGreedyBestFirst;

public Toggle iSortDistance;
public Toggle iSortTime;
public Toggle iSortSpeed;

//Assigning canvases and game objects to each variable
public GameObject eightDir;
public GameObject fourDir;
public GameObject eightDir2;
```

```
public GameObject fourDir2;
public GameObject eightDir3;
public GameObject fourDir3;
public GameObject eightDir4;
public GameObject fourDir4;

public GameObject menuCanvas;
public GameObject canvas1;
public GameObject canvas2;
public GameObject canvas3;
public GameObject canvas4;
public GameObject canvas5;
public GameObject canvas6;
public GameObject canvas7;

public GameObject backgroundImage;

public GameObject pauseButton;
public GameObject playButton;
public GameObject ThreeDimButton;
public GameObject TwoDimButton;

public GameObject ChooseAnother;
public GameObject Comparison;
public GameObject ScrollView;
public GameObject Warning;
public GameObject Key;
public GameObject Parameters;

public GameObject textDijkstra;
public GameObject textAStar;
public GameObject textBreadth;
public GameObject textGreedy;

//Assigning text boxes to all error code types
public Text errorCodes;
public Text errorCodes2;
public Text errorCodes3;
public Text errorCodes4;
public Text errorCodes5;
public Text errorCodes6;
public Text errorCodes7;

//Declaring all default speed, time and distance variables for the pathfinders
public float DijkstraTime = 999999;
public float AStarTime = 999998;
public float BreadthTime = 999997;
public float GreedyTime = 999996;

public float DijkstraDistance = 999999;
public float AStarDistance = 999998;
public float BreadthDistance = 999997;
public float GreedyDistance = 999996;

public float DijkstraSpeed;
public float AStarSpeed;
public float BreadthSpeed;
public float GreedySpeed;

//Update is called once every frame
void Update()
{
```

```
//Slowly rotate background image
backgroundImage.transform.Rotate(0, 0, 0.33f);
}

//Called to check user inputted values on the settings menu
public void CheckInput()
{
    //Clear error codes text box
    errorCodes.text = "";

    //Check if user input is in the correct data type and store value
    bool success = int.TryParse(iWidth.text, out temp);
    if (success && temp >= 3 && temp <= 100)
    {
        width = temp;
    }
    //If the text box is left blank use default value
    else if (iWidth.text == "")
    {
        width = 25;
    }
    //Otherwise, if incorrect input then display error message
    else
    {
        errorCodes.text += "\n\nIncorrect WIDTH value submitted";
    }

    //Check if user input is in the correct data type and store value
    success = int.TryParse(iHeight.text, out temp);
    if (success && temp >= 3f && temp <= 100f)
    {
        height = temp;
    }
    //If the text box is left blank use default value
    else if (iHeight.text == "")
    {
        height = 25;
    }
    //Otherwise, if incorrect input then display error message
    else
    {
        errorCodes.text += "\n\nIncorrect HEIGHT value submitted";
    }

    //Check if user input is in the correct data type and store value
    success = float.TryParse(iBorderSize.text, out tempf);
    if (success && tempf >= 0f && tempf <= 0.5f)
    {
        borderSize = tempf;
    }
    //If the text box is left blank use default value
    else if (iBorderSize.text == "")
    {
        borderSize = 0.1f;
    }
    //Otherwise, if incorrect input then display error message
    else
    {
        errorCodes.text += "\n\nIncorrect BORDER SIZE value submitted";
    }

    //Check if user input is in the correct data type and store value
```

```

    success = int.TryParse(iStartX.text, out temp);
    if ((success && temp >= 0 && temp <= 99) && (temp <= (width - 1) && temp >=
0))
    {
        startX = temp;
    }
//If the text box is left blank use default value
else if (iStartX.text == "")
{
    startX = 0;
}
//Otherwise, if incorrect input then display error message
else
{
    errorCodes.text += "\n\nIncorrect STARTX value submitted";
}

//Check if user input is in the correct data type and store value
success = int.TryParse(iStartY.text, out temp);
if ((success && temp >= 0 && temp <= 99) && (temp <= (height - 1) && temp >=
0))
{
    startY = temp;
}
//If the text box is left blank use default value
else if (iStartY.text == "")
{
    startY = 0;
}
//Otherwise, if incorrect input then display error message
else
{
    errorCodes.text += "\n\nIncorrect STARTY value submitted";
}

//Check if user input is in the correct data type and store value
bool successX = int.TryParse(iGoalX.text, out tempX);
if (successX && tempX >= 0 && tempX <= 99 && (tempX <= (width - 1) && tempX >=
0))
{
    goalX = tempX;
}
//If the text box is left blank use default value
else if (iGoalX.text == "" && goalX <= (width - 1) && goalX >= 0)
{
    goalX = 24;
}
//Otherwise, if incorrect input then display error message
else
{
    errorCodes.text += "\n\nIncorrect GOALX value submitted";
}

//Check if user input is in the correct data type, not the same as start y,
and store value
bool successY = int.TryParse(iGoalY.text, out tempY);
if ((successY && tempY >= 0 && tempY <= 99 && tempY <= (height - 1) && tempY
>= 0))
{
    goalY = tempY;
}

```

```
//If the text box is left blank use default value
else if (iGoalY.text == "" && goalY <= (height - 1) && goalY >= 0 && (goalX !=
startX && goalY != startY))
{
    goalY = 24;
}
//Otherwise, if incorrect input then display error message
else
{
    errorCodes.text += "\n\nIncorrect GOALY value submitted";
}

//Check if user input for goalX and goalY is the same as startX and startY,
and display error message
if (iGoalX.text != "" && iGoalY.text != "" && (tempX == startX && tempY ==
startY) && goalX == startX && goalY == startY)
{
    errorCodes.text += "\n\nThe GOAL and START nodes have the same
coordinates";
}

//Check for if the Show Arrows toggle is set to active
if (iShowArrows.isOn == true)
{
    //Set Show Arrows as true
    showArrows = true;
}
else
    //Set Show Arrows to false
    showArrows = false;

//Check for if the Exit On Goal toggle is set to active
if (iExitOnGoal.isOn == true)
{
    //Set Exit On Goal to true
    ExitOnGoal = true;
}
else
{
    //Set Exit On Goal to false
    ExitOnGoal = false;
}

//Check for if the Show Arrows toggle is set to active
if (iShowIterations.isOn == true)
{
    //Set Show Iterations to true
    showIterations = true;

    //Check if user input is in the correct data type and store value
    success = float.TryParse(iTimeStep.text, out tempf);
    if (success && tempf >= 0.01f && tempf <= 5)
    {
        timeStep = tempf;
    }
    //If the text box is left blank use default value
    else if (iTimeStep.text == "")
    {
        timeStep = 0.01f;
    }
    //Otherwise, if incorrect input then display error message
    else
```

```
        {
            errorCodes.text += "\n\nIncorrect TIME STEP value submitted";
        }
    }
else
    //Set Show Iterations as false
    showIterations = false;

//Check for if the Eight Directions toggle is set to active
if (iEightDirections.isOn == true)
{
    //Set Eight Directions to true
    eightDirections = true;
}
else
{
    //Set Eight Directions to false
    eightDirections = false;
}

//Check if there were no error codes
if (errorCodes.text == "")
{
    //Pass the user onto the next menu screen
    Pass = true;
}

//Set settings menu as inactive and type of maze menu as active
if (Pass == true)
{
    canvas1.SetActive(false);
    canvas2.SetActive(true);
}

//Change the image indicating number of directions depending on state of Eight
Directions
if (eightDirections == true)
{
    eightDir.SetActive(true);
    fourDir.SetActive(false);
}
else
{
    eightDir.SetActive(false);
    fourDir.SetActive(true);
}

//Display all the user inputted values so the user can see their previous
input
errorCodes2.text = "Maze Width: " + width + "\n\nMaze Height: " + height +
"\n\nStart X: " + startX +
    "\n\nStart Y: " + startY + "\n\nGoal X: " + goalX + "\n\nGoal Y: " + goalY
+
    "\n\nBorder Size: " + borderSize + "\n\nExit On Goal: " + ExitOnGoal +
"\n\nShow Arrows: " + showArrows +
    "\n\nShow Iterations: " + showIterations;

//Check if show iteration was enabled
if (showIterations == true)
{
    //Display the user inputted value for Time Step
    errorCodes2.text += "\n\nTime Step: " + timeStep;
```



```

    }
}

//Called to check user input on the maze type menu
public void CheckInput2()
{
    //Check if preset maze was chosen
    if (iPresetMaze.isOn == true)
    {
        //Go to the preset maze menu
        canvas2.SetActive(false);
        canvas3.SetActive(true);
        //Display warning on a preset maze using non-user defined values
        errorCodes3.text = "<b><color=red>WARNING:</color>\nWidth\nHeight\nStart
X\nStart Y\nGoal X\nGoal Y\n</b>\n" +
            "Will be <b><color=red>RESET</color></b> if a Preset Maze is chosen";
        //Change the image indicating number of directions depending on state of
Eight Directions
        if (eightDirections == true)
        {
            eightDir2.SetActive(true);
            fourDir2.SetActive(false);
        }
        else
        {
            eightDir2.SetActive(false);
            fourDir2.SetActive(true);
        }
    }

    //Check if random maze was chosen
    else if (iRandomMaze.isOn == true)
    {
        //Go to the random maze menu
        canvas2.SetActive(false);
        canvas4.SetActive(true);

//Display all the user inputted values so the user can see their previous inputs
        errorCodes4.text = "Maze Width: " + width + "\n\nMaze Height: " + height +
"\n\nStart X: " + startX +
        "\n\nStart Y: " + startY + "\n\nGoal X: " + goalX + "\n\nGoal Y: " + goalY
+
        "\n\nBorder Size: " + borderSize + "\n\nExit On Goal: " + ExitOnGoal +
"\n\nShow Arrows: " + showArrows +
        "\n\nShow Iterations: " + showIterations;

        //Check if show iteration was enabled
        if (showIterations == true)
        {
            //Display the user inputted value for Time Step
            errorCodes4.text += "\n\nTime Step: " + timeStep;
        }
        //Change the image indicating number of directions depending on state of
Eight Directions
        if (eightDirections == true)
        {
            eightDir3.SetActive(true);
            fourDir3.SetActive(false);
        }
        else
        {
            eightDir3.SetActive(false);

```

```

        fourDir3.SetActive(true);
    }

}

//Check if custom maze was chosen
else if (iCustomMaze.isOn == true)
{
    canvas2.SetActive(false);
    errorCodes5.text = "<b><size=39>INSTRUCTIONS</size>\n\n1. Click on a
desired Node Type below\n\n2. Click the Nodes on the left to apply the Node Type\n\n3.
Click 'Create' when you are completed</b>";
    //Change the image indicating number of directions depending on state of
Eight Directions
    if (eightDirections == true)
    {
        eightDir4.SetActive(true);
        fourDir4.SetActive(false);
    }
    else
    {
        eightDir4.SetActive(false);
        fourDir4.SetActive(true);
    }
    canvas5.SetActive(true);

    //Get the grid dimensions for the custom maze menu
    GameObject.Find("GridHolder").GetComponent<GridSet>().CustomMazeLayout();
}
//Check if the user has not selected either maze type
else
{
    //Display error code
    if (!errorCodes2.text.Contains("Choose an Option"))
        errorCodes2.text += "\n\n<color=red>Choose an Option</color>";
}
}

//Called to check user input values on the preset maze menu
public void CheckPreset()
{
    //Check if preset maze one is chosen
    if (iPresetMaze1.isOn == true)
    {
        //Declare the variable values according to the preset and set pathfinder
menu as active
        canvas3.SetActive(false);
        startX = 0;
        startY = 0;
        goalX = 14;
        goalY = 14;
        width = 15;
        height = 15;
        canvas6.SetActive(true);
    }

    //Check if preset maze two is chosen
    else if (iPresetMaze2.isOn == true)
    {
        //Declare the variable values according to the preset and set pathfinder
menu as active
        canvas3.SetActive(false);

```

```

        startX = 0;
        startY = 24;
        goalX = 24;
        goalY = 0;
        width = 25;
        height = 25;
        canvas6.SetActive(true);
    }

    //Check if preset maze three is chosen
    else if (iPresetMaze3.isOn == true)
    {
        //Declare the variable values according to the preset and set pathfinder
        menu as active
        canvas3.SetActive(false);
        startX = 0;
        startY = 50;
        goalX = 99;
        goalY = 50;
        width = 100;
        height = 100;
        canvas6.SetActive(true);
    }

    //Check if preset maze four is chosen
    else if (iPresetMaze4.isOn == true)
    {
        //Declare the variable values according to the preset and set pathfinder
        menu as active
        canvas3.SetActive(false);
        startX = 0;
        startY = 0;
        goalX = 49;
        goalY = 8;
        width = 50;
        height = 9;
        canvas6.SetActive(true);
    }
    //Check if the user has not selected a preset maze
    else
    {
        //Display error codes
        if (!errorCodes3.text.Contains("Choose an Option"))
            errorCodes3.text += "\n<b><size=29><color=red>\nChoose an
Option</color></size></b>";
    }
}

//Called to check user input on the random maze menu
public void CheckRandom()
{
    //Check if random maze one is chosen
    if (iRandomMaze1.isOn == true)
    {
        //Set pathfinder menu as active
        canvas4.SetActive(false);
        canvas6.SetActive(true);
    }
    //Check if random maze two is chosen
    else if (iRandomMaze2.isOn == true)

```

```
{
    //Set pathfinder menu as active
    canvas4.SetActive(false);
    canvas6.SetActive(true);
}
//Check if random maze three is chosen
else if (iRandomMaze3.isOn == true)
{
    //Set pathfinder menu as active
    canvas4.SetActive(false);
    canvas6.SetActive(true);
}
//Check if user has not selected a random maze
else
{
    //Display error codes
    if (!errorCodes4.text.Contains("Choose an Option"))
        errorCodes4.text += "\n<b><size=29><color=red>\nChoose an
Option</color></size></b>";
}
}

//Called to check user inputted values on the custom maze menu
public void CreateCustomMaze()
{
    //Set pathfinder menu as active
    canvas5.SetActive(false);
    canvas6.SetActive(true);
}

//Called when the user wants to create a new maze
public void ResetScene()
{
    //Resets all variable values and goes back to menu screen
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}

//Called to check what pathfinder is chosen to run by the user
public void CheckPathfinder()
{
    //Set timescale to real world time
    Time.timeScale = 1;

    //Check if Dijkstra's algorithm is selected
    if (iDijkstra.isOn == true)
    {
        //Go to graph of nodes and begin pathfinder
        canvas6.SetActive(false);

        GameObject.Find("DemoController").GetComponent<DemoController>().BeginMaze();
        canvas7.SetActive(true);
        zoom = true;
    }

    //Check if A Star search has been selected
    else if (iAStar.isOn == true)
    {
        //Go to graph of nodes and begin pathfinder
        canvas6.SetActive(false);
    }
}
```

```

GameObject.Find("DemoController").GetComponent<DemoController>().BeginMaze();

    canvas7.SetActive(true);
    zoom = true;
}

//Check if Breadth First Search has been selected
else if (iBreadthFirstSearch.isOn == true)
{
    //Go to graph of nodes and begin pathfinder
    canvas6.SetActive(false);

GameObject.Find("DemoController").GetComponent<DemoController>().BeginMaze();
    canvas7.SetActive(true);
    zoom = true;
}

//Check if Greedy Best First search has been selected
else if (iGreedyBestFirst.isOn == true)
{
    //Go to graph of nodes and begin pathfinder
    canvas6.SetActive(false);

GameObject.Find("DemoController").GetComponent<DemoController>().BeginMaze();
    canvas7.SetActive(true);
    zoom = true;
}

//Check if the user has not selected a pathfinder
else
{
    //Display error code
    if (!errorCodes6.Contains("Choose an Option"))
        errorCodes6.text += "\n<b><size=29><color=red>\nChoose an
Option</color></size></b>";
}

//Check if a pathfinder has been successfully chosen
if(zoom == true)
{
    //Display 3D/2D buttons as well as Play/Pause buttons
    playButton.SetActive(false);
    pauseButton.SetActive(true);
    TwoDimButton.SetActive(false);
    ThreeDimButton.SetActive(true);
}

//Display all the user inputted values so the user can see their previous
inputs
errorCodes7.text = "\n\n\n\nMaze Width: " + width + "\n\nMaze Height: " +
height + "\n\nStart X: " + startX +
    "\n\nStart Y: " + startY + "\n\nGoal X: " + goalX + "\n\nGoal Y: " + goalY
+
    "\n\nBorder Size: " + borderSize + "\n\nExit On Goal: " + ExitOnGoal +
"\n\nShow Arrows: " + showArrows +
    "\n\nShow Iterations: " + showIterations + "\n\nEight Directions: " +
eightDirections;

//Check if show iteration is set to true
if (showIterations == true)

```

```

    {
        //Display time step
        errorCodes7.text += "\n\nTime Step: " + timeStep;
    }

    //Check what pathfinding algorithm is being run and display its name
    if (iDijkstra.isOn == true)
    {
        errorCodes7.text += "\n\n<size=40><i>Dijkstra's Algorithm</i></size>";
    }
    else if (iAStar.isOn == true)
    {
        errorCodes7.text += "\n\n<size=40><i>A* (A Star) Search</i></size>";
    }
    else if (iBreadthFirstSearch.isOn == true)
    {
        errorCodes7.text += "\n\n<size=40><i>Breadth First Search</i></size>";
    }
    else if (iGreedyBestFirst.isOn == true)
    {
        errorCodes7.text += "\n\n<size=40><i>Greedy Best First</i></size>";
    }
}

//Called to go back to the main menu
public void Back()
{
    //Reset camera position, size, and type and reset selected variable values
    canvas2.SetActive(false);

    Time.timeScale = 1;
    Camera.main.orthographic = true;
    Camera.main.transform.eulerAngles = new Vector3(90, 0, 0);
    Camera.main.transform.position = new Vector3(510, 385, -950);
    Camera.main.orthographicSize = (385);

    GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().zoom =
false;
    iPresetMaze.isOn = false;
    iRandomMaze.isOn = false;
    iCustomMaze.isOn = false;

    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    //Set main menu screen as active
    menuCanvas.SetActive(true);
}

//Called to go back to the previous menu page
public void Back2()
{
    //Reset variable values set on current menu screen
    canvas3.SetActive(false);
    iPresetMaze1.isOn = false;
    iPresetMaze2.isOn = false;
    iPresetMaze3.isOn = false;
    iPresetMaze4.isOn = false;
    //Set previous menu screen as active
    canvas2.SetActive(true);
}

//Called to go back to the previous menu page
public void Back3()

```

```
{
    //Reset variable values set on current menu screen
    canvas4.SetActive(false);
    iRandomMaze1.isOn = false;
    iRandomMaze2.isOn = false;
    iRandomMaze3.isOn = false;
    //Set previous menu screen as active
    canvas2.SetActive(true);
}

//Called to go back to the previous menu page
public void Back4()
{
    //Reset variable values set on current menu screen
    canvas5.SetActive(false);
    iCustomMaze.isOn = false;
    //Set previous menu screen as active
    canvas2.SetActive(true);
}

//Called to go back to the previous menu page
public void Back5()
{
    //Reset variable values set on current menu screen
    ChooseAnother.SetActive(false);
    iSortDistance.isOn = false;
    iSortTime.isOn = false;
    iSortSpeed.isOn = false;
    Comparison.SetActive(false);
    Warning.SetActive(false);
    ScrollView.SetActive(false);
    Key.SetActive(false);
    Parameters.SetActive(false);
    canvas7.SetActive(false);
    Camera.main.transform.eulerAngles = new Vector3(-90, 0, 0);
    //Set previous menu screen as active
    canvas6.SetActive(true);
}

//Called when pathfinding information is to be sorted by distance
public void SortDistanceUpdate()
{
    //Update all time, distance, and speed values
    UpdateValues();
    //Get a list of distances for different pathfinders
    List<float> distanceList = new List<float>();
    //Clear the list
    distanceList.Clear();
    //Add all distances to the list
    distanceList.Add(DijkstraDistance);
    distanceList.Add(AStarDistance);
    distanceList.Add(BreadthDistance);
    distanceList.Add(GreedyDistance);
    //Sort the list in ascending order
    distanceList.Sort();
    //Change position of text based on placement in list
    SortList(distanceList, DijkstraDistance, AStarDistance, BreadthDistance,
GreedyDistance);
}

//Called when pathfinding information is to be sorted by time
public void SortTimeUpdate()
```

```

    {
        //Update all time, distance, and speed values
        UpdateValues();
        //Get a list of distances for different pathfinders
        List<float> timeList = new List<float>();
        //Clear the list
        timeList.Clear();
        //Add all distances to the list
        timeList.Add(DijkstraTime);
        timeList.Add(AStarTime);
        timeList.Add(BreadthTime);
        timeList.Add(GreedyTime);
        //Sort the list in ascending order
        timeList.Sort();
        //Change position of text based on placement in list
        SortList(timeList, DijkstraTime, AStarTime, BreadthTime, GreedyTime);
    }

    //Called when pathfinding information is to be sorted by speed
    public void SortSpeedUpdate()
    {
        //Update all time, distance, and speed values
        UpdateValues();
        //Get a list of distances for different pathfinders
        List<float> speedList = new List<float>();
        //Clear the list
        //Add all distances to the list

        speedList.Clear();
        speedList.Add(DijkstraSpeed);
        speedList.Add(AStarSpeed);
        speedList.Add(BreadthSpeed);
        speedList.Add(GreedySpeed);
        //Sort the list in descending order
        speedList.Sort();
        speedList.Reverse();
        //Change position of text based on placement in list
        SortList(speedList, DijkstraSpeed, AStarSpeed, BreadthSpeed, GreedySpeed);
    }

    //Called when the pathfinder has completed its search
    public void UpdateValues()
    {
        //Update time, distance and speed for Dijkstra's algorithm
        DijkstraDistance =
        GameObject.FindGameObjectWithTag("Pathfinder").GetComponent<Pathfinder>().DijkstraDistance;
        DijkstraTime =
        GameObject.FindGameObjectWithTag("Pathfinder").GetComponent<Pathfinder>().DijkstraTime
        ;
        DijkstraSpeed =
        GameObject.FindGameObjectWithTag("Pathfinder").GetComponent<Pathfinder>().DijkstraSpeed;
        DijkstraDistance = DijkstraDistance + 0.00001f;

        //Update time, distance and speed for A Star search algorithm
        AStarDistance =
        GameObject.FindGameObjectWithTag("Pathfinder").GetComponent<Pathfinder>().AStarDistance;
        AStarTime =
        GameObject.FindGameObjectWithTag("Pathfinder").GetComponent<Pathfinder>().AStarTime;
    }

```



```

    AStarSpeed =
GameObject.FindGameObjectWithTag("Pathfinder").GetComponent<Pathfinder>().AStarSpeed;
    AStarDistance = AStarDistance + 0.00002f;

    //Update time, distance and speed for Breadth first search algorithm
    BreadthDistance =
GameObject.FindGameObjectWithTag("Pathfinder").GetComponent<Pathfinder>().BreadthDistance;
    BreadthTime =
GameObject.FindGameObjectWithTag("Pathfinder").GetComponent<Pathfinder>().BreadthTime;
    BreadthSpeed =
GameObject.FindGameObjectWithTag("Pathfinder").GetComponent<Pathfinder>().BreadthSpeed;
;
    BreadthDistance = BreadthDistance + 0.00003f;

    //Update time, distance and speed for Greedy best first search algorithm
    GreedyDistance =
GameObject.FindGameObjectWithTag("Pathfinder").GetComponent<Pathfinder>().GreedyDistance;
    GreedyTime =
GameObject.FindGameObjectWithTag("Pathfinder").GetComponent<Pathfinder>().GreedyTime;
    GreedySpeed =
GameObject.FindGameObjectWithTag("Pathfinder").GetComponent<Pathfinder>().GreedySpeed;
    GreedyDistance = GreedyDistance + 0.00004f;
}

//Called to change position of text values based on placement in list of
Distance/Time/Speed
public void SortList(List<float> type, float iDijkstra, float iAStar, float
iBreadth, float iGreedy)
{
    //Check what pathfinder is first in the list and place in the top-left
    if (type[0] == iDijkstra && iDijkstra != iAStar && iDijkstra != iBreadth &&
iDijkstra != iGreedy)
    {
        textDijkstra.GetComponent<RectTransform>().localPosition = new Vector3(-
250, 200, 0);
    }
    else if (type[0] == iAStar && iAStar != iDijkstra && iAStar != iBreadth &&
iAStar != iGreedy)
    {
        textAStar.GetComponent<RectTransform>().localPosition = new Vector3(-250,
200, 0);
    }
    else if (type[0] == iBreadth && iBreadth != iDijkstra && iBreadth != iAStar &&
iBreadth != iGreedy)
    {
        textBreadth.GetComponent<RectTransform>().localPosition = new Vector3(-
250, 200, 0);
    }
    else if (type[0] == iGreedy && iGreedy != iDijkstra && iGreedy != iAStar &&
iGreedy != iBreadth)
    {
        textGreedy.GetComponent<RectTransform>().localPosition = new Vector3(-250,
200, 0);
    }

    //Check what pathfinder is second in the list and place in the top-right
    if (type[1] == iDijkstra && iDijkstra != iAStar && iDijkstra != iBreadth &&
iDijkstra != iGreedy)
    {

```

```

        textDijkstra.GetComponent<RectTransform>().localPosition = new
Vector3(250, 200, 0);
    }
    else if (type[1] == iAStar && iAStar != iDijkstra && iAStar != iBreadth &&
iAStar != iGreedy)
    {
        textAStar.GetComponent<RectTransform>().localPosition = new Vector3(250,
200, 0);
    }
    else if (type[1] == iBreadth && iBreadth != iDijkstra && iBreadth != iAStar &&
iBreadth != iGreedy)
    {
        textBreadth.GetComponent<RectTransform>().localPosition = new Vector3(250,
200, 0);
    }
    else if (type[1] == iGreedy && iGreedy != iDijkstra && iGreedy != iAStar &&
iGreedy != iBreadth)
    {
        textGreedy.GetComponent<RectTransform>().localPosition = new Vector3(250,
200, 0);
    }

    //Check what pathfinder is third in the list and place in the bottom-left
    if (type[2] == iDijkstra && iDijkstra != iAStar && iDijkstra != iBreadth &&
iDijkstra != iGreedy)
    {
        textDijkstra.GetComponent<RectTransform>().localPosition = new Vector3(-
250, -200, 0);
    }
    else if (type[2] == iAStar && iAStar != iDijkstra && iAStar != iBreadth &&
iAStar != iGreedy)
    {
        textAStar.GetComponent<RectTransform>().localPosition = new Vector3(-250,
-200, 0);
    }
    else if (type[2] == iBreadth && iBreadth != iDijkstra && iBreadth != iAStar &&
iBreadth != iGreedy)
    {
        textBreadth.GetComponent<RectTransform>().localPosition = new Vector3(-
250, -200, 0);
    }
    else if (type[2] == iGreedy && iGreedy != iDijkstra && iGreedy != iAStar &&
iGreedy != iBreadth)
    {
        textGreedy.GetComponent<RectTransform>().localPosition = new Vector3(-250,
-200, 0);
    }

    //Check what pathfinder is fourth in the list and place in the bottom-right
    if (type[3] == iDijkstra && iDijkstra != iAStar && iDijkstra != iBreadth &&
iDijkstra != iGreedy)
    {
        textDijkstra.GetComponent<RectTransform>().localPosition = new
Vector3(250, -200, 0);
    }
    else if (type[3] == iAStar && iAStar != iDijkstra && iAStar != iBreadth &&
iAStar != iGreedy)
    {
        textAStar.GetComponent<RectTransform>().localPosition = new Vector3(250, -
200, 0);
    }
}

```

```

        else if (type[3] == iBreadth && iBreadth != iDijkstra && iBreadth != iAStar &&
iBreadth != iGreedy)
        {
            textBreadth.GetComponent<RectTransform>().localPosition = new Vector3(250,
-200, 0);
        }
        else if (type[3] == iGreedy && iGreedy != iDijkstra && iGreedy != iAStar &&
iGreedy != iBreadth)
        {
            textGreedy.GetComponent<RectTransform>().localPosition = new Vector3(250,
-200, 0);
        }
    }
}

```

## Sort Type Code:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class SortType : MonoBehaviour
{
    //Assigning game objects to each variable
    public Toggle SortDistance;
    public Toggle SortTime;
    public Toggle SortSpeed;

    //Called when the results are being sorted
    public void SortCheck()
    {
        //Check for if user wants to sort by distance
        if (SortDistance.isOn == true)
        {
            GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().SortDistanceUpdate
();
        }
        //Check for if user wants to sort by time
        else if (SortTime.isOn == true)
        {
            GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().SortTimeUpdate();
        }
        //Check for if user wants to sort by distance
        else if (SortSpeed.isOn == true)
        {
            GameObject.FindGameObjectWithTag("Script").GetComponent<Settings>().SortSpeedUpdate();
        }
    }
}

```

## Toggle Instructions Code:

```

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

```

```
//Class used to show/hide information when a toggle is clicke in the menu
public class ToggleInstructions : MonoBehaviour
{
    //Assigning game objects to each variable
    public GameObject Panel;
    public GameObject Panel2;
    public GameObject Panel3;
    public GameObject Panel4;
    public GameObject Panel5;
    public GameObject Panel6;

    //Open instruction menu for about the program
    public void OpenScrollAbout()
    {
        Panel6.SetActive(false);
        Panel5.SetActive(false);
        Panel4.SetActive(false);
        Panel3.SetActive(false);
        Panel2.SetActive(false);
        Panel.SetActive(true);
    }

    //Open instruction menu for Dijkstra's Algorithm
    public void OpenScrollDijkstra()
    {
        Panel6.SetActive(false);
        Panel5.SetActive(false);
        Panel4.SetActive(false);
        Panel3.SetActive(false);
        Panel.SetActive(false);
        Panel2.SetActive(true);
    }

    //Open instruction menu for A Star Algorithm
    public void OpenScrollAStar()
    {
        Panel6.SetActive(false);
        Panel5.SetActive(false);
        Panel4.SetActive(false);
        Panel.SetActive(false);
        Panel2.SetActive(false);
        Panel3.SetActive(true);
    }

    //Open instruction menu for Breadth First Search
    public void OpenScrollBreadth()
    {
        Panel6.SetActive(false);
        Panel5.SetActive(false);
        Panel.SetActive(false);
        Panel3.SetActive(false);
        Panel2.SetActive(false);
        Panel4.SetActive(true);
    }

    //Open instruction menu for Greedy Best First Search
    public void OpenScrollGreedy()
    {
        Panel6.SetActive(false);
        Panel.SetActive(false);
        Panel4.SetActive(false);
    }
}
```

```
        Panel3.SetActive(false);
        Panel2.SetActive(false);
        Panel5.SetActive(true);
    }
    //Open instruction menu for the controls for the program
    public void OpenScrollControls()
    {
        Panel.SetActive(false);
        Panel5.SetActive(false);
        Panel4.SetActive(false);
        Panel3.SetActive(false);
        Panel2.SetActive(false);
        Panel6.SetActive(true);
    }
}
```

## Toggle Settings Code:

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

//Class used to show/hide information when a toggle is clicked in the menu
public class ToggleSettings : MonoBehaviour
{
    //Assigning game objects to each variable
    public GameObject Panel;
    public GameObject Panel2;
    public GameObject Panel3;
    public GameObject Panel4;

    //Called when the user clicks on a toggle
    public void OpenPanel()
    {
        //If clicked on, change state of panel to shown/hidden
        if (Panel != null)
        {
            bool isActive = Panel.activeSelf;
            Panel.SetActive(!isActive);
        }
        //If clicked on, change state of panel 2 to shown/hidden
        if (Panel2 != null)
        {
            bool isActive = Panel2.activeSelf;
            Panel2.SetActive(!isActive);
        }
        //If clicked on, change state of panel 3 to shown/hidden
        if (Panel3 != null)
        {
            bool isActive = Panel3.activeSelf;
            Panel3.SetActive(!isActive);
        }
        //If clicked on, change state of panel 4 to shown/hidden
        if (Panel4 != null)
        {
            bool isActive = Panel4.activeSelf;
            Panel4.SetActive(!isActive);
        }
    }
}
```

## Testing

Objective	Test #	Input	Expected Output	Actual Output	Pass/Fail	Comments
Maze width Default: 25 Range: 3-100	1	'No input'	25	25	Pass	Default value is used
	2	14	14	14	Pass	
	3	14.2	'Incorrect maze Width'	'Incorrect Maze Width'	Pass	
	4	200	'Incorrect maze Width'	'Incorrect maze Width'	Pass	
	5	-50	'Incorrect maze Width'	'Incorrect maze Width'	Pass	
Maze height Default: 25 Range: 3-100	1	'No input'	25	25	Pass	Default Value
	2	14	14	14	Pass	
	3	14.2	'Incorrect maze height'	'Incorrect Maze height'	Pass	
	4	200	'Incorrect maze height'	'Incorrect maze height'	Pass	
	5	-50	'Incorrect maze height'	'Incorrect maze height'	Pass	
Starting x-coordinate Default: 0 Range: 0-99 below width minus one	1	'No input'	0	0	Pass	
	2	14	14	14	Pass	
	3	14.2	'Incorrect x-coordinate'	'Incorrect x-coordinate'	Pass	
	4	200	'Incorrect x-	'Incorrect x-	Pass	

			coordinat e' 'Incorrec	coordinat e' 'Incorrec		
	5	-50	t x- coordinat e' 0	t x- coordinat e' 0	Pass	
Start y-coordinate Default: 0 Range: 0-99 and below height minus one	1	'No input'			Pass	
	2	14	14 'Incorrec	14 'Incorrec	Pass	
	3	14.2	t y- coordinat e' 'Incorrec	t y- coordinat e' 'Incorrec	Pass	
	4	200	t y- coordinat e' 'Incorrec	t y- coordinat e' 'Incorrec	Pass	
	5	-50	t y- coordinat e' 0	t y- coordinat e' 0	Pass	
End x-coordinate Default: 24 Range: 0-99 and below width minus one	1	'No input'			Pass	
	2	14	14 'Incorrec	14 'Incorrec	Pass	
	3	14.2	t x- coordinat e' 'Incorrec	t x- coordinat e' 'Incorrec	Pass	
	4	200	t x- coordinat e' 'Incorrec	t x- coordinat e' 'Incorrec	Pass	
	5	-50	t x- coordinat e' 24	t x- coordinat e' 24	Pass	
End y-coordinate Default: 24 Range: 0-99 and below height minus one	1	'No input'			Pass	
	2	14	14 'Incorrec	14 'Incorrec	Pass	
	3	14.2	t y- coordinat e' 'Incorrec	t y- coordinat e' 'Incorrec	Pass	
	4	200	t y-	t y-	Pass	

			coordinat e' 'Incorrec	coordinat e' 'Incorrec		
	5	-50	t y- coordinat e'	t y- coordinat e'	Pass	
A menu to select what type of maze type the user would like and an error check to remind the user if they have not selected an option	1	'Preset Maze'	'Preset Maze'	'Preset Maze' (On the video)	Pass	
	2	'Nothing Selected'	'Choose an option'	'Choose an option'	Pass	
A preset maze menu from which the user can select premade graphs and an error check to remind the user if they have not selected an option	1	'Preset Maze1'	'Preset Maze1'	'Preset Maze1'	Pass	
	2	'Nothing Selected'	'Choose an option'	'Choose an option'	Pass	
A random maze menu from which the user can select different types of random mazes and an error check to remind the user if they have not selected an option	1	'Random Maze1'	'Random Maze1'	'Random Maze1'	Pass	
	2	'Nothing Selected'	'Choose an option'	'Choose an option'	Pass	
Allow the user to zoom in and out the graph (using the scroll wheel),	1	'Scroll up'	Zoom in	Zoom in	Pass	
	2	'Scroll Down'	Zoom out	Zoom Out	Pass	
A menu from which the desired pathfinder to be run can be selected by the user and an error check to remind the user if they have not selected an option	1	'Dijkstra's Algorithm'	'Dijkstra's Algorithm'	'Dijkstra's Algorithm'	Pass	
	2	'Nothing Selected'	'Choose an Option'	'Choose an Option'	Pass	
			Zoom in	Zoom in		



The three-dimensional camera can be zoomed in and out using the scroll wheel	1	'Scroll up'			Pass	
	2	'Scroll Down'	Zoom out	Zoom Out	Pass	
A pause/play button that will pause the timer and the pathfinder	1	'Play'	Plays pathfinder	Plays Pathfinder	Pass	
	2	'Pause'	Pauses Pathfinder	Pauses Pathfinder	Pass	
User can change slider that changes the direction of light incident on the graph which will change the appearance of shadows	1	Move the slider's position	Direction of light changes	Direction of Light changes	Pass	
	1	Move the slider's position	Colour of background changes to RGB value	Colour of background changes to RGB value	Pass	
Create external text file named 'Log.txt' when all the pathfinding algorithms have been run if a 'Log.txt' file does not already exist	1	All pathfinders have been run	File is created/ added onto	File is created/ added onto (Video evidence)	Pass	
Record the date and time at which the set of pathfinding algorithms were run	1	All pathfinders have been run	The accurate date and time are recorded	The accurate date and time are recorded (Video evidence)		

Record all the time taken, distance travelled, speed, and nodes explored into the text file	1	All pathfinders have been run	All the values are written in the text file	All the values are written in the text file		
---------------------------------------------------------------------------------------------	---	-------------------------------	---------------------------------------------	---------------------------------------------	--	--

## Evaluation

### Completeness of Objectives

Objectives	Achieved	Evidence
1. A main menu which the user can navigate through <ol style="list-style-type: none"> <li>a. A button to begin the making of the maze</li> <li>b. A button to view the instructions of the program</li> </ol>	Yes	Menu Selector (Page 59)  Main Menu (Page 25)
2. An instructions page which can be accessed from the pathfinder and the main menu <ol style="list-style-type: none"> <li>a. Information on the following topics to be displayed when clicked on:               <ol style="list-style-type: none"> <li>i. About Program</li> <li>ii. Dijkstra's Algorithm</li> <li>iii. Breadth-First Search</li> <li>iv. Greedy Best First</li> <li>v. Controls</li> </ol> </li> <li>b. For the main menu instruction page               <ol style="list-style-type: none"> <li>i. Exit button to go back to main menu</li> </ol> </li> <li>c. For the pathfinder instruction page               <ol style="list-style-type: none"> <li>i. Exit button to go back to selecting pathfinder</li> </ol> </li> </ol>	Yes	Toggle Instructions Script (Page 98)  Instructions Page (Page 25)
3. A settings page where the user can set the desired values for the maze <ol style="list-style-type: none"> <li>a. A button to go back to the main menu</li> <li>b. A button to confirm the values and go on to the next page</li> <li>c. A list of Integer variables with a default value which the user can change (restricted between a certain range) that include:               <ol style="list-style-type: none"> <li>i. Maze width                   <ol style="list-style-type: none"> <li>1. Default: 25</li> <li>2. Range: 3-100</li> </ol> </li> <li>ii. Maze height</li> </ol> </li> </ol>	Yes	Settings Script (Page 81)  Settings Page (Page 26)

<ul style="list-style-type: none"><li>1. Default: 25</li><li>2. Range: 3-100</li><li>iii. Starting x-coordinate<ul style="list-style-type: none"><li>1. Default: 0</li><li>2. Range: 0-99 below width minus one</li></ul></li><li>iv. Start y-coordinate<ul style="list-style-type: none"><li>1. Default: 0</li><li>2. Range: 0-99 and below height minus one</li></ul></li><li>v. End x-coordinate<ul style="list-style-type: none"><li>1. Default: 24</li><li>2. Range: 0-99 and below width minus one</li></ul></li><li>vi. End y-coordinate<ul style="list-style-type: none"><li>1. Default: 24</li><li>2. Range: 0-99 and below height minus one</li></ul></li><li>d. A list of Boolean variables which the user can change that include:<ul style="list-style-type: none"><li>i. Pathfinder exit on goal reached<ul style="list-style-type: none"><li>1. Default: True</li></ul></li><li>ii. Show arrows displaying the previous node where the pathfinder traversed from<ul style="list-style-type: none"><li>1. Default: True</li></ul></li><li>iii. Allow the pathfinder to traverse in eight directions<ul style="list-style-type: none"><li>1. Default: True (False means travels in four directions)</li></ul></li><li>iv. Show the pathfinder as it traverses through the graph<ul style="list-style-type: none"><li>1. Default: True (False means just the final path is displayed)</li></ul></li></ul></li><li>e. A list of Float variables which the user can change that include<ul style="list-style-type: none"><li>i. The size of the gap between adjacent nodes (border size)<ul style="list-style-type: none"><li>1. Default: 0.1</li><li>2. Range: 0-0.5</li></ul></li><li>ii. When show iterations is true, the time interval in seconds between each step in the pathfinder<ul style="list-style-type: none"><li>1. Default: 0.01</li><li>2. Range: 0.01-5</li></ul></li></ul></li><li>f. A dynamic display that shows what will happen in the case of each Boolean variable being true or false</li></ul>		
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--	--

<p>with their descriptions being changed accordingly</p> <ul style="list-style-type: none"> <li>g. An error check for if any value entered is out of range when the confirm button is clicked</li> <li>h. An error check for the start and end nodes being the same coordinate as that distance is always zero</li> <li>i. A list of all erroneous values that have been inputted incorrectly by the user to be displayed clearly.</li> <li>j. An entry box next to each variable so the user can see the default value as well as their edited one</li> <li>k. Disable the time step entry box if the option to show iterations is set to false.</li> </ul>		
<p>4. A menu to select what type of maze type the user would like</p> <ul style="list-style-type: none"> <li>a. Display all the parameters which the user has entered clearly onto a list</li> <li>b. A button for a preset maze</li> <li>c. A button for a random maze</li> <li>d. A button for a custom maze</li> <li>e. An option to go back to the main menu</li> <li>f. A button to confirm the choice of maze type</li> <li>g. An error check to remind the user if they have not selected an option</li> </ul>	Yes	<p>Maze Type Selection (Page 27)</p> <p>Video evidence</p>
<p>5. A preset maze menu from which the user can select premade graphs</p> <ul style="list-style-type: none"> <li>a. An accurate picture of the preset maze displayed next to the option to select the respective maze</li> <li>b. Description below each preset maze displaying its width, height, starting x coordinate, starting y coordinate, ending x-coordinate, and ending y-coordinate.</li> <li>c. Premade maze types <ul style="list-style-type: none"> <li>i. A simple perfect maze</li> <li>ii. A maze with a few small gaps awkwardly placed to test the pathfinder</li> <li>iii. A massive maze with a straight path to the goal as well as a winding path</li> </ul> </li> </ul>	Yes	<p>Preset Maze Menu (Page 28)</p> <p>Settings Script (Page 81)</p>

<ul style="list-style-type: none"> <li>iv. A maze with straight paths to the goal but with different weights on each path</li> <li>d. A button to go back to the previous maze type selection menu</li> <li>e. A button to confirm the choice of preset maze</li> <li>f. An error check to remind the user if they have not selected an option</li> <li>g. A warning that their custom variable values for the width, height, starting x coordinate, starting y coordinate, ending x-coordinate, and ending y-coordinate will not be used in the preset maze</li> <li>h. A scrollable window to select these maze presets from</li> </ul>		
<ul style="list-style-type: none"> <li>6. A random maze menu from which the user can select different types of random mazes <ul style="list-style-type: none"> <li>a. An example of the random maze displayed next to the option to select the respective maze</li> <li>b. Random maze types <ul style="list-style-type: none"> <li>i. A dense random maze (33% chance of a wall)</li> <li>ii. A sparse random maze (25% chance of a wall)</li> <li>iii. A very sparse random maze (20% chance of a wall)</li> </ul> </li> <li>c. A short description below each random maze displaying the likelihood of a node being a wall.</li> <li>d. A button to go back to the previous maze type selection menu</li> <li>e. A button to confirm the choice of random maze</li> <li>f. An error check to remind the user if they have not selected an option</li> <li>g. A scrollable window to select the random maze type from</li> <li>h. A list of parameters set by the user displayed clearly on the screen</li> </ul> </li> </ul>	Yes	Random Maze Menu (Page 28)  Settings Script (Page 81)
<ul style="list-style-type: none"> <li>7. A custom maze menu where the user can create their own customised graph <ul style="list-style-type: none"> <li>a. Instructions on how to use the custom maze editor</li> <li>b. A key displaying all the node type distances and behaviour</li> </ul> </li> </ul>	Yes	Custom Maze Menu (Page 30)  Grid Set Script (Page 50)



<ul style="list-style-type: none"> <li>c. An error check to remind the user if they have not selected an option</li> <li>d. Buttons for each pathfinder in the program that include: <ul style="list-style-type: none"> <li>i. Dijkstra's Algorithm</li> <li>ii. A* (A Star)</li> <li>iii. Breadth First Search</li> <li>iv. Greedy Best First</li> </ul> </li> <li>e. Relevant information with a short summary of each pathfinder to be displayed clearly when the mouse is hovered over the relevant pathfinder</li> </ul>	Yes	Script (Page 61)
<p>9. The screen for which the pathfinder is carried out on the maze selected by the user</p> <ul style="list-style-type: none"> <li>a. A fully modelled three-dimensional graph that the pathfinder will be carried out on with shadows and dynamic lighting from a light source</li> <li>b. Various buttons to change/view settings: <ul style="list-style-type: none"> <li>i. 'Toggle parameters' which will display the variable setting that were set by the user <ul style="list-style-type: none"> <li>1. A key that displays the meaning of the colours on the graph that include: <ul style="list-style-type: none"> <li>a. Blocked node (black) which the pathfinder cannot traverse through</li> <li>b. Open node (white) which has horizontal/vertical distance of one</li> <li>c. Explored node (light grey) which shows nodes that the pathfinder has explored</li> <li>d. Neighbour node (light blue) which shows nodes that are neighbouring the explored nodes of the pathfinder (next to be explored by the pathfinder)</li> </ul> </li> </ul> </li> </ul> </li> </ul>		Pathfinders are run (Page 31)  Camera Control Script (Page 37)  Change Camera Script (Page 41)  Demo Controller Script (Page 42)  Graph Script (Page 44)  Graph View Script (Page 46)  Map Data Script (Page 53)  Node Script (Page 62)  Node View Script (Page 63)

<ul style="list-style-type: none"> <li>e. Path nodes (orange) which show the final path determined by the pathfinding algorithm</li> <li>f. Start node (green) which shows the position in the graph where the pathfinder starts from</li> <li>g. Goal node (red) which shows the position in the graph the pathfinding algorithm ends at</li> <li>h. Old neighbour (dark blue) which shows the neighbouring node of the previous pathfinding algorithm that was run</li> <li>i. Old explored (dark grey) which shows the old nodes which the previous pathfinder had explored</li> <li>ii. 'Select another pathfinder' that will allow the user to select another pathfinder to run</li> <li>iii. 'Toggle results' will show the user the time taken, distance travelled, speed, and nodes explored for the pathfinders that have been run, (in order in which they were run) in a scrollable list <ul style="list-style-type: none"> <li>1. A button to return back to menu will appear <ul style="list-style-type: none"> <li>a. A warning displaying that the current maze</li> </ul> </li> </ul> </li> </ul>		<p>Map Data Script (Page 53)</p> <p>Move Background Script (Page 61)</p> <p>Pathfinder Script (Page 66)</p> <p>Pause Script (Page 78)</p> <p>Priority Queue Script (Page 79)</p> <p>Sort Type Script (Page 98)</p> <p>Toggle Instructions Script (Page 98)</p> <p>Toggle Settings Script (Page 100)</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



<p>data will be lost if the pathfinders have not all been run yet</p> <ol style="list-style-type: none"><li>2. A button to toggle the information on the various pathfinders<ol style="list-style-type: none"><li>a. Dynamic table with the ability to sort by distance, time, or speed where the rankings of each algorithm will be shown</li><li>b. Algorithms that have not been run will display a reminder that they have not been selected yet</li></ol></li><li>iv. A 3-d/2-d camera option that will change the view of the graph from a top-down two-dimensional view of the graph, to that of a fully modelled three-dimensional view of the graph, or vice versa<ol style="list-style-type: none"><li>1. Two-dimensional camera will be default when the program is first run for the clearest view of the graph for the user</li><li>2. Allow the user to zoom in and out the graph (using the scroll wheel), drag across the graph (clicking the scroll wheel), and reset the graph by using the right mouse button</li><li>3. Three-dimensional camera will pivot around the centre point of the graph and rotate horizontally and vertically according to the mouse's movement. Prevent the user from rotating the camera</li></ol></li></ol>		
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--	--

<p>below the horizontal x axis of the graph.</p> <p>4. The three-dimensional camera can be zoomed in and out using the scroll wheel</p> <p>v. A pause/play button that will pause the timer and the pathfinder while it is being run, whilst also maintaining the functionality of both dimensional camera types</p> <p>vi. A drop-down menu where the user can change various graphical settings that include:</p> <ol style="list-style-type: none"> <li>1. Direction of light incident on the graph which will change the appearance of shadows</li> <li>2. The colour of the background which can be changed by the user from the default grey to any RGB value</li> </ol> <p>c. A button to select another pathfinder to carry out on the maze</p>	Yes	
<p>10. The program will save all relevant information in an external text file when all pathfinding algorithms have been run</p> <ol style="list-style-type: none"> <li>a. Create external text file named 'Log.txt' when all the pathfinding algorithms have been run if a 'Log.txt' file does not already exist</li> <li>b. Record the date and time at which the set of pathfinding algorithms were run</li> <li>c. Record the variable values that were set by the user for the maze in question</li> <li>d. Record all the time taken, distance travelled, speed, and nodes explored into the text file</li> <li>e. Save/Update the text file in 'Log.txt' which will be created next to the program file</li> </ol>		<p>File Structure (Page 23)</p> <p>Pathfinder Script (Page 66)</p>

## Conclusion

Overall, I believe I have met all of the objectives I initially set out to achieve. Through iterative testing and continually making improvements to my project, as well as receiving feedback from third parties, all 10 sets of objectives were completed. I began my project by creating the main part of the program, the pathfinding algorithms, before any of the menus and customisation options. Then as the core of my program (main purpose) I began to complete the other supplementary objectives which were not as integral to the goal of my project. This allowed me to make sure that my project was consistently functional throughout the process of my project.

Some improvements that could be made to my project include an option to import an existing 'Log.txt' file that was created from a previous running of the pathfinding algorithms program, and allow the user to continue using the pathfinders on the graph that they have saved earlier. Another improvement could be to allow the user to save the results of the pathfinders even if all the pathfinding algorithms have not been completed yet. However, all these improvements are quality of life improvements for the user and do not affect the main goal of my program which was to investigate, as well as compare, the benefits and drawbacks of the more prevalent algorithms used in computer pathfinding, using Dijkstra's Algorithm, Breadth-First Search, A\*(A Star) Search Algorithm, and Greedy Best-First. The goal was to make a program that could be used as both a teaching and learning tool for beginners, as well as a problem-solving tool for people to solve real-world shortest path problems using my project.

My feedback from my third-party client was that, after reviewing my final finished pathfinding project, that I had exceeded the initial aim of my program listed in my objectives, and created a fully-functional, reliable tool for any user to utilise. From taking the feedback presented to me, both in the survey and interview, I have added multiple additional objective to my initial set, which I believe to have greatly improved the quality of my project. In conclusion, as I have used a variety of advanced data structures and computing techniques, such as priority queues, stacks, multi-dimensional arrays etc, as well as meeting and surpassing the initial goals set out in the objectives, I believe I have succeeded in making my project a valuable program to be used by many others wanting to learn more about pathfinding algorithms.