2019

Year 7 Pathfinding Program

NEA PROJECT – AQA COMPUTER SCIENCE 2018-2019

FINLAY MCCORMICK | CANDIDATE 3136

QES | Centre 42339

Contents

Contents 1

Analysis 4

Introduction 4

Current method 4

Uses of pathfinding algorithms 4

Algorithms 5

Graph traversal algorithms 5

Shortest path algorithms 5

Algorithm I will use 9

Why my program? 9

Ideas after interview 9

Hardware limitations 9

Data 10

GUI 14

Flow chart 15

Scope 15

Requirements 16

FR 1 GUI 16

FR 2 graph 16

FR 3 calculating the route 17

MSCW table 17

Design 18

Introduction 18

A* algorithm 18

Structure overview 20

aStar class 21

aStar class pseudocode 21

Map 23

Data 24

Graph 24

Coordinates 26

Rooms 27

Nodes 27

Instructions 27

- Testing A* algorithm with real data ... 27
- Startup ... 28
- GUI ... 29
- interface class ... 30
- Interface class pseudocode ... 30
- Main ... 32
- Main pseudocode ... 33

Testing ... 35

- Test strategy ... 35
  - A* algorithm ... 35
  - Data structures ... 35
  - GUI ... 36
  - Beta testing ... 36
- A* algorithm ... 36
  - Testing A* algorithm works with test graph ... 36
  - Testing A* algorithm performs cost calculations correctly ... 36
  - Testing A* algorithm finds most efficient route ... 38
  - Testing A* algorithm calculates time taken correctly ... 40
- Data structures ... 42
  - Testing createRooms function ... 42
  - Testing createInstructions function ... 45
  - Testing createGraph function ... 46
  - Testing createCoordinates function ... 47
  - Testing createNodes function ... 49
  - Testing what happens when file is missing ... 50
  - Testing what happens when file is corrupted ... 51
- GUI ... 53
  - Testing the start screen ... 53
  - Testing the main screen ... 54
  - Testing the dropdown lists ... 55
  - Testing the calculate route button ... 57
  - Testing route is cleared when new option selected ... 63

Evaluation ... 66

Bibliography ... 67

Appendices ... 68

- One: Interview questions ... 68

Two: Site map 69

Analysis

Introduction

QES is a comprehensive secondary school located in Kirkby Lonsdale, which gets a large number of new students every year. One of the biggest problems these new students face is finding their way around the school, since it is quite large and confusing. Whilst the school does have an activity on the first day to try to help the new students to find their way around, I think that a program would be more useful. This program would allow them to enter the room they are in and the room they want to go to, before displaying a route on a map of the school with accompanying instructions. The main group my program will aim to help though will be year seven students, meaning that it will have to be easy to understand and not throw too many instructions at them.

My client for this project is Mr Reid, who is the head of the school's year seven population and an English teacher. He is in charge of the year sevens wellbeing and helping them to get comfortable at QES. I chose him to be my client because he has the most experience working with year sevens and best understands what they would need from a program like the one I am developing.

Current method

The method used now to help year sevens to find their way around the school is an activity on the first day. The context is that someone has kidnapped the school's head teacher and that they need to find out who did it. This activity has the year sevens following instructions in order to find clues hidden around the school. Upon completion they would use the clues they had collected to find out who did it.

According to Mr Reid, this activity was a great success, however one of the biggest problems I can see with it is that it only happens once at the start of the school year. Whilst it does help to get students used to the school and where things are, if someone doesn't know where to go at any point later in the year the only way to get help is by finding a teacher or older student and asking them. This could be quite scary for new students who don't know anyone.

Uses of pathfinding algorithms

One of the most common uses of pathfinding algorithms are in satellite navigation systems. Most satnavs use a vector map instead of a tiled map such as the one used in Google Maps. These vectors effectively represent straight road segments, with a beginning and end that is defined by latitude and longitude. They will also have information such as the road name and what type of road it is, for example, a motorway. For calculating the route, the most common algorithm is the A* algorithm, rather than something like Dijkstra's. This is because of how many roads there are on any sat nav map and the fact that for the algorithm to be efficient it should ignore most of the roads on the map, since they will either be in the wrong direction or on a longer route. I talk about how the A* algorithm works in the algorithm section of the analysis. When the route has been calculated the system will need to generate and display the map and route using the vectors and constantly update it to keep up with how the vehicle is moving. (PC Plus, n.d.)

Algorithms

Graph traversal algorithms

Breadth first algorithm

Breadth first algorithms work by traversing through every child node, then every grandchild node, etcetera to find if a value is present in a graph. (Joshi, 2017)

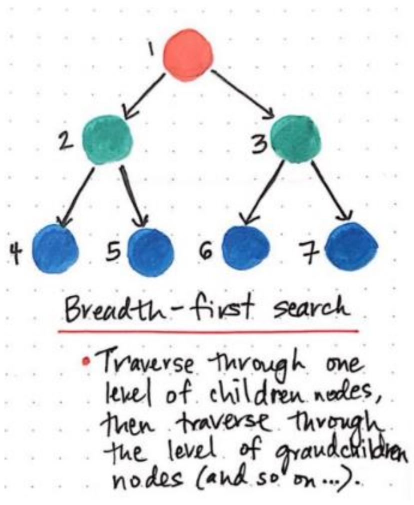

1
2
3
4
5
6
7
Breadth-first search
• Traverse through one level of children nodes, then traverse through the level of grandchildren nodes (and so on...).

Figure 1: Taken from Medium.com

Depth first algorithm

Depth first algorithms work by traversing a graph from the parent node through the children until it reaches a dead end. When it reaches a dead end (there are no more nodes to visit in its path) it backtracks to a point where it can choose another path with unvisited nodes. It repeats this process until all nodes have been visited. (Ridder, 2018)

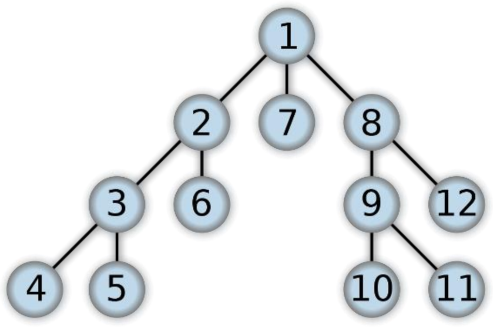


Figure 2: Taken from Wikipedia.org

Shortest path algorithms

Dijsktra's algorithm

Dijkstra's algorithm works by creating a tree of shortest paths from the start node to all other points in the graph. It does this by creating three variables: dist (an array of the distances from the start node to each node in the graph), Q (a queue of all nodes in the graph) and S (a set that contains every node the algorithm has visited). The algorithm starts by popping node v, a node not in S and with the smallest dist. The start node will always be popped first since it has a dist value of 0. Next it adds v to S to show it has been visited, before calculating new dist values for all adjacent nodes to v. This process continues until dist contains the shortest distances from the start node to every node in the graph. (Thaddeus Abiy, n.d.)

A* algorithm

Another algorithm used for pathfinding is the A* algorithm. The way this algorithm works is that it gives every node a g cost, an h cost and an f cost. The g cost is the distance from the start node and is calculated by traversing through already calculated nodes and adding the different weights together. The h cost is the distance from the end node and is calculated by effectively drawing a straight line from the current node to the end node and using Pythagoras to calculate the distance. The f cost is the sum of the g cost and the h cost. The algorithm starts by calculating the three costs of all the nodes surrounding the start node, whichever one of these has the lowest f cost then has its surrounding nodes calculated (I will call a node that has had its neighbours calculated an activated node). By repeating this process the program will eventually find the quickest route to the end node. (Rachit Belwariar, n.d.)

Below are some examples of an A* algorithm working on a simple 7 x 7 grid where every cell is a node. When a node is activated it calculates the values of the nodes directly next to it vertically, horizontally and diagonally. When the destination has been reached the route is generated by traversing up through the parents of the activated nodes leading to the end node. The cost values I use for going horizontally or vertically are 10 and the cost value for going diagonally is 14. This is because if we assume each square is 1 apart from each other, using Pythagoras the value for a diagonal movement is approximately 1.4. I multiply these values by 10 to make the costs easier to calculate and read.

Key:
The numbers in the nodes are g cost/h cost/f cost.

Start node	Calculated node	Activated node	End node	Route
🟥	🟩	🟨	🟦	🟧

Example 1

This example is a simple graph with no obstacles between the start and end node.

14/51/65	10/41/51	14/32/46				
10/50/60	00/40/40	10/30/40			??/00/??	
14/51/65	10/41/51	14/32/46				

When the algorithm starts it calculates the g, h and f cost for all of the nodes surrounding the start node.

14/51/65	10/41/51	14/32/46	24/22/46			
10/50/60	00/40/40	10/30/40	20/20/40		??/00/??	
14/51/65	10/41/51	14/32/46	24/22/46			

The node with the lowest f cost is the node directly to the right of the starting node, so it is activated and calculates the costs of all of the nodes surrounding it. Three of the recently activated nodes neighbours have already had their costs calculated, however if the newly activated node had a lower g cost than the neighbours current parents, the recently activated node would become their new parent and their costs would be updated.

14/51/65	10/41/51	14/32/46	24/22/46	34/14/48	44/10/54	
10/50/60	00/40/40	10/30/40	20/20/40	30/10/40	40/00/40	
14/51/65	10/41/51	14/32/46	24/22/46	34/14/48	44/10/54	

This process continues since the lowest possible f cost is 40 in this scenario and the only nodes with that value are the nodes directly between the start and end nodes.

14/51/65	10/41/51	14/32/46	24/22/46	34/14/48	44/10/54	
10/50/60	00/40/40	10/30/40	20/20/40	30/10/40	40/00/40	
14/51/65	10/41/51	14/32/46	24/22/46	34/14/48	44/10/54	

When the end node is reached the route is generated by traversing up through the parents of the activated nodes.

<u>Example 2</u>
This example has an obstacle (represented by the black squares). These cells are effectively just not nodes, meaning that the algorithm will have to find a way around them.

14/51/65	10/41/51	14/32/46	■			
10/50/60	00/40/40	10/30/40	■		??/00/??	
14/51/65	10/41/51	14/32/46	■			
			■			

The costs of the nodes surrounding the start node are calculated.

14/51/65	10/41/51	14/32/46	■			
10/50/60	00/40/40	10/30/40	■		??/00/??	
14/51/65	10/41/51	14/32/46	■			
			■			

The node with the lowest f cost is activated, but the obstacle means no new nodes have their values calculated.

	28/45/73	24/36/60	28/28/56			
14/51/65	10/41/51	14/32/46	⬛			
10/50/60	00/40/40	10/30/40	⬛		??/00/??	
14/51/65	10/41/51	14/32/46	⬛			
	28/45/73	24/36/60	⬛			

The next lowest f cost present is 46 and belongs to the nodes directly above and below the first activated node.

24/54/78	20/45/65	24/36/60	28/28/56			
14/51/65	10/41/51	14/32/46	⬛			
10/50/60	00/40/40	10/30/40	⬛		??/00/??	
14/51/65	10/41/51	14/32/46	⬛			
24/54/78	20/45/65	24/36/60	⬛			

The algorithm will then activate the two nodes with f cost 51, since these are the non-activated nodes with the lowest f costs. The node two cells above the start node has already had its g cost calculated, however due to the newly activated node having a lower g cost than its current parent, the newly activated node becomes its parent and its g and f costs are updated.

		34/42/76	38/36/74	42/32/74		
24/54/78	20/45/65	24/36/60	28/28/56	38/22/60		
14/51/65	10/41/51	14/32/46	⬛	42/14/56		
10/50/60	00/40/40	10/30/40	⬛		??/00/??	
14/51/65	10/41/51	14/32/46	⬛			
24/54/78	20/45/65	24/36/60	⬛			

At this point, the node with the lowest f cost is the node directly above the obstacle, so it is activated.

		34/42/76	38/36/74	42/32/74		
24/54/78	20/45/65	24/36/60	28/28/56	38/22/60	56/20/76	
14/51/65	10/41/51	14/32/46	⬛	42/14/56	52/10/62	
10/50/60	00/40/40	10/30/40	⬛	52/10/62	56/00/56	
14/51/65	10/41/51	14/32/46	⬛			
24/54/78	20/45/65	24/36/60	⬛			

The lowest f cost is now 48, so the node with that value is activated.

		34/42/76	38/36/74	42/32/74		
24/54/78	20/45/65	24/36/60	28/28/56	38/22/60	56/20/76	
14/51/65	10/41/51	14/32/46	⬛	42/14/56	52/10/62	
10/50/60	00/40/40	10/30/40	⬛	52/10/62	56/00/56	
14/51/65	10/41/51	14/32/46	⬛			
24/54/78	20/45/65	24/36/60	⬛			

The end node has been reached, so the quickest route is generated by traversing up through the parents of each node which leads to the end node.

Algorithm I will use

The algorithm that I will use for my program is the A* algorithm. I chose this algorithm over breadth first depth first because these algorithms are only for traversing graphs and not calculating the quickest routes, and I chose it over Dijkstra's because it calculates the quickest route to every node on the graph, whilst I am only interested in finding the shortest route to the destination node. The A* algorithm is one of the best pathfinding algorithms for quickly calculating the route from one point on a graph to another because it bases its calculations off of distance from the start node and distance to the end node, meaning that it is unlikely to waste time going down a path that does not lead to the destination. Speed is important for this program because the users will most likely be in a rush, meaning that Dijsktra's algorithm, which could take longer, is inappropriate when compared to the quicker A* algorithm.

Why my program?

There are many different pathfinding programs already available; however, my program will be designed to work on a much smaller scale than others, which will be more appropriate for a relatively small area like QES. My program will also be designed around speed and simplicity of use, which is important for a younger student who may be in a rush to get to a lesson. The program I am making will also be free, unlike alternatives.

Ideas after interview

After my interview with my client Mr Reid, shown in appendix one, I came up with some ideas for my project. When the user has entered their starting location and their destination, the program will not only display the route but it will also display how long it will take on average to complete the route. The program could also allow the user to select different buildings on the map to view a more detailed map of the building's interior. My client also said that it may be a good idea to have the school separated into four different sectors, each with a checkpoint where users can go to use the program. Instead of simply showing the user the quickest route to their destination, it could instead point them to the checkpoint of the sector that their destination is in, where they can then use the program again at that checkpoint to go from there to the destination.

Hardware limitations

My program should not have any hardware limitations because it requires very little processing and memory to run. The only potential issues that could be caused by hardware are having a weak CPU, which may lead to the program running slightly slower, or not having enough memory to handle all of the data, which would require the computer to use virtual memory. However, these potential issues will not be a problem since all the computers at QES have good enough hardware to run this program.

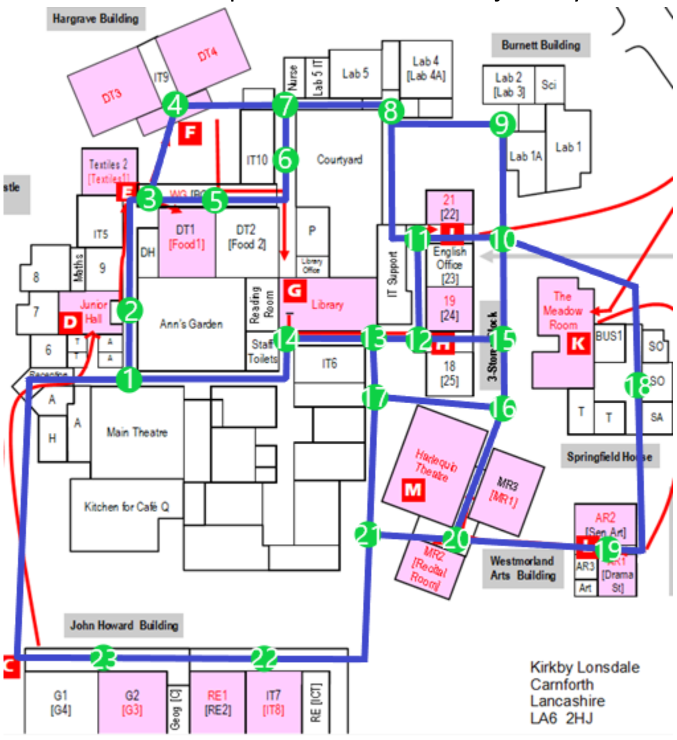
Data

For my nodes I am going to use an adjacency list, this is a dictionary that will store all the nodes, which nodes are connected to it and the weight of the edge between the nodes. Altogether, there are 24 nodes and around 30 edges, meaning that an adjacency list, rather than an adjacency matrix, would be most efficient since there are a lot of nodes and only a few more edges.

My adjacency list would look something like this, however there are no weights in this table because I have not been able to calculate them yet.

Nodes	Adjacent nodes
1	2, 14, 23
2	1, 3
3	2, 4, 5
4	3, 7
5	3, 6
6	5, 7
7	4, 6, 8
8	7, 9, 11
9	8, 10
10	9, 11, 15, 18
11	8, 10, 12
12	11, 13, 15
13	12, 14, 17
14	1, 13
15	10, 12, 16
16	15, 17, 20
17	13, 16, 21
18	10, 19
19	18, 20
20	16, 19, 21
21	17, 20, 22
22	21, 23
23	1, 22

Below is a visual representation of the adjacency list using the map from appendix two.



The graph will be used in the main A* algorithm class that calculates the quickest route. It will be imported as a parameter.

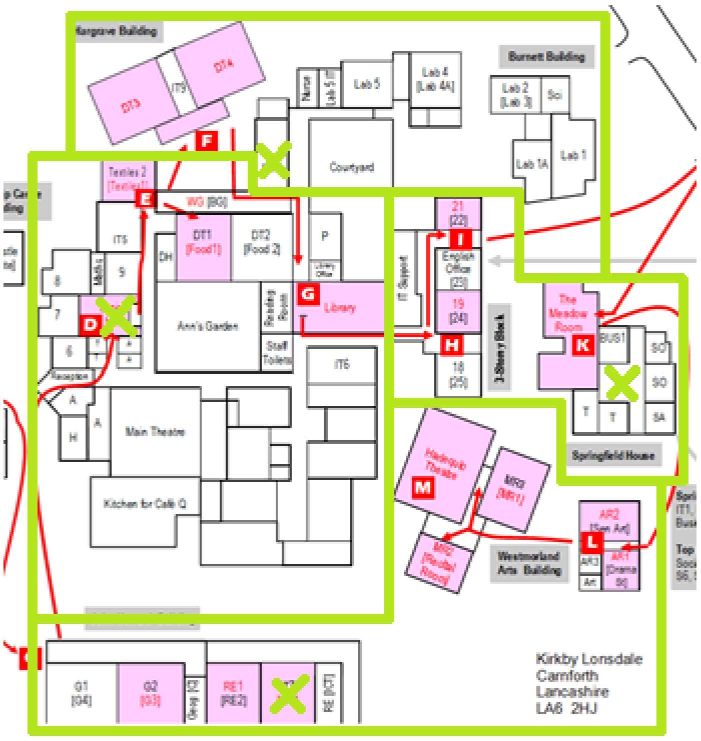
On my graph, one node could be used to represent multiple rooms. I will use a dictionary containing every room present on the map alongside the node representing it. Having this is important because it means that the user can enter rooms into the program instead of the node that the room is represented by. This is what part of the dictionary would look like.

Rooms	Nodes
Main Theatre	1
Junior Hall	2
IT5	3
DT3	4
DT1	5
IT10	6
Nurse	7
Lab 5	8

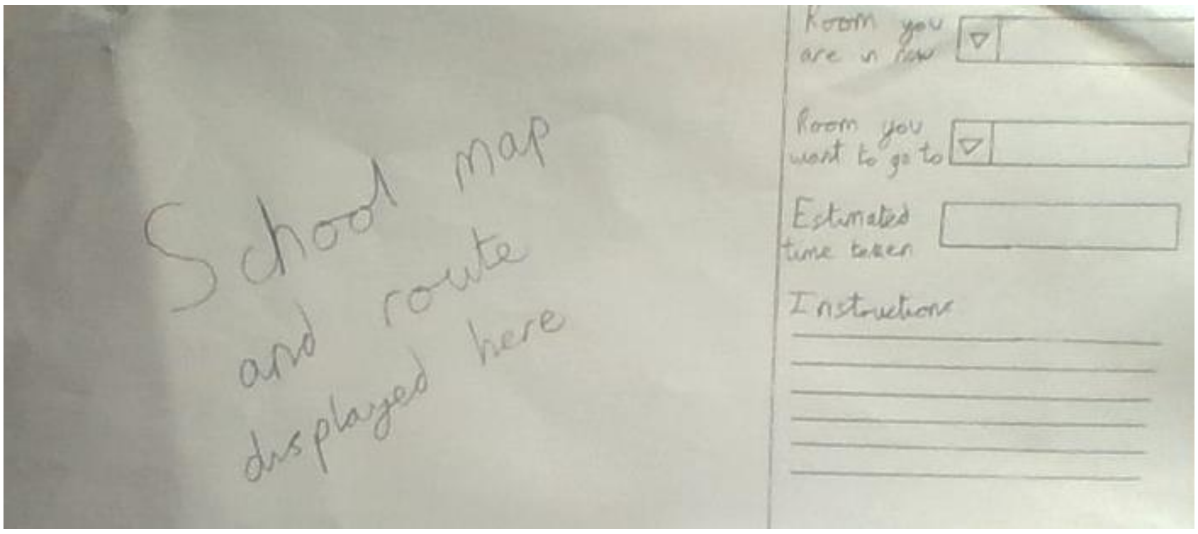
This dictionary will be used in the main part of the program when the user has selected their starting and destination rooms.

All the above data will be stored in a separate text file and imported into my program, rather than being stored in the program itself. This is to make it easier for people to edit the map and rooms without having to touch the code itself.

Though I am no longer sure if I will use this idea, below is the outline of the sectors and where the checkpoints would be located (represented by a green x). I don't think I will use this idea because the graph I have created does not have the complexity to necessitate the route being broken into multiple, potentially longer routes. This is the opposite of what this program needs to be, which is fast and efficient.



GUI

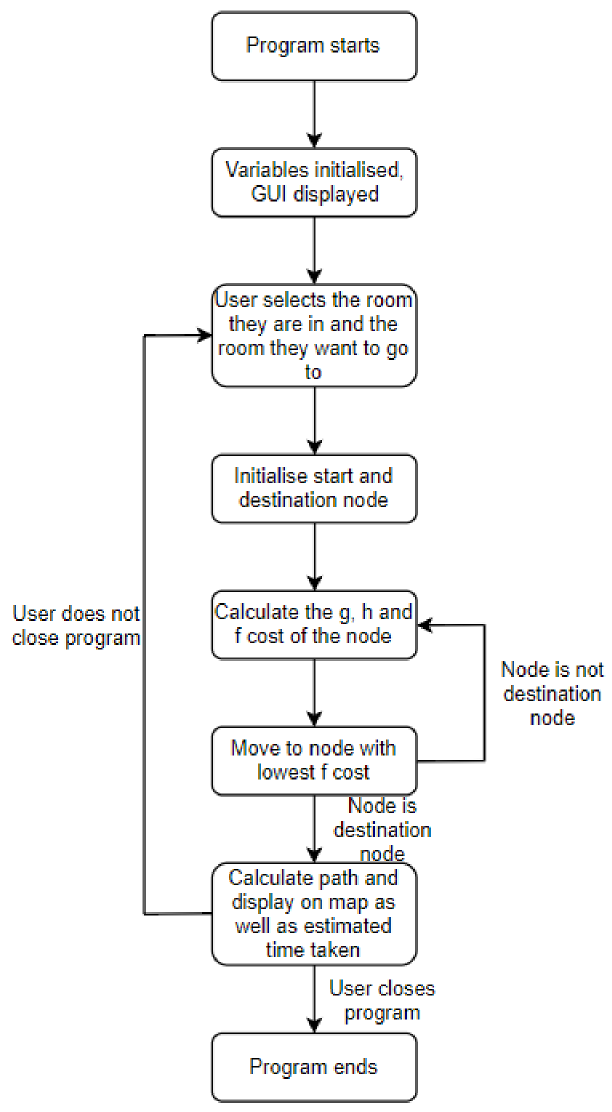


Above is a basic representation of what my programs GUI will look like. I asked my client and some potential end users what they thought about it and they said that it looked good but should have the estimated time taken both walking and running. The estimated times will be calculated by taking the final distance of the route and dividing it by an average walking and running speed to get the time. The instructions are not instructions of how to follow the route but rather on how to work the program, so that it will be easy for anyone to use.

Below is a picture of me talking about the GUI with some potential end users.

Flow chart

This is a flow chart showing how the program will run.


Program starts
Variables initialised, GUI displayed
User selects the room they are in and the room they want to go to
Initialise start and destination node
Calculate the g, h and f cost of the node
Move to node with lowest f cost
Node is not destination node
Node is destination node
Calculate path and display on map as well as estimated time taken
User does not close program
User closes program
Program ends

Scope

For this project, I intend to make a pathfinding project which will help students to find their way around the school, meaning that I need a pathfinding algorithm. For my project I am going to use the A* algorithm.

I also intend to have the program calculate and display the time a user will need to follow the route to their destination, assuming they are walking at a defined average walking speed.

I do not intend to represent the entire school in my graph as shown in appendix two. This is because the central parts of the school are complex enough to properly test my program and it should be able to adapt to new nodes added to the graph anyway.

I also decided to not implement any additional instructions such as turn left or turn right, since the program will not be run in real time, so the user will have to memorise the instructions and the route, which a year seven could struggle with.

Requirements

FR 1 GUI

This section is about the requirements of the graphical user interface.

FR 1.1:

The user must be able to see an interactive GUI which shows the school map.

FR 1.1.1:

The user could be able to select buildings on the map and get a more detailed view of the interior of the building.

FR 1.2:

There must be dropdown lists where the user can select the room they are in and the room they want to go to.

FR 1.3:

The GUI should show a set of basic instructions on how to work the program.

FR 1.4:

When the route has been calculated the GUI must display the route.

FR 1.4.1:

The route must be displayed as a red line displayed on top of the school map.

FR 1.4.2:

The estimated time taken when walking and running could also be calculated, based off a constant walking and running speed value, and displayed.

FR 2 graph

This section is about the requirements of the graph representing the school.

FR 2.1:

There must be a graph with nodes and edges representing the layout of the school.

FR 2.1.1:

The edges must have a weight representing the distance between the nodes it is between.

FR 2.1.1.1:

Some of the edges weights could change based on the time of day.

FR 2.1.2:

Most of the nodes will represent an area or set of rooms within the school.

FR 2.2:

On the graph one node can be used to represent multiple rooms so there must be a dictionary which contains every room on the map and which node represents it.

FR 2.3:

The graph would be separated into four different sections with a central node in each section which acts as a checkpoint.

FR 3 calculating the route

This section is about the requirements of the algorithms being used to calculate the route.

FR 3.1:

The program must use the A* algorithm to calculate the quickest route from the room the user is in to the user's destination.

FR 3.2:

The program should use a constant walking and running speed to calculate an estimated time taken to complete the route when running and walking.

MSCW table

Functional requirement	Must have	Should have	Could have	Would have
GUI with map of the school	X			
GUI with interactive map of the school that allows user to select buildings and see detailed view of interior			X	
GUI with dropdown lists that allow user to select their start and destination rooms	X			
GUI with instructions showing how to use program		X		
GUI that displays route when calculated as a red line on the map	X			
GUI that displays estimated time taken when walking and running		X		
Graph that represents school as nodes and edges	X			
Graph with weighted edges representing the distance between the nodes	X			
Graph with weighted edges that change at different times of day to represent congestion			X	
Graph with nodes representing different areas and rooms around school	X			
Dictionary which stores every node and the rooms they represent	X			
Graph which is separated into different sections with a checkpoint in each section				X
Program which uses A* algorithm used to calculate quickest route between two nodes	X			
Program which calculates estimated time taken when running and walking		X		

Design

Introduction

My project intends to provide an appropriate tool for year sevens to use to navigate the school easier and see the fastest route from where they are to where they need to go. Due to the fact that my program is meant to be used by younger children who may not be comfortable using computers, I will be designing it to be as clear and easy to use as possible, with few items they can interact with, such as buttons or menu boxes, and without showing too much information.

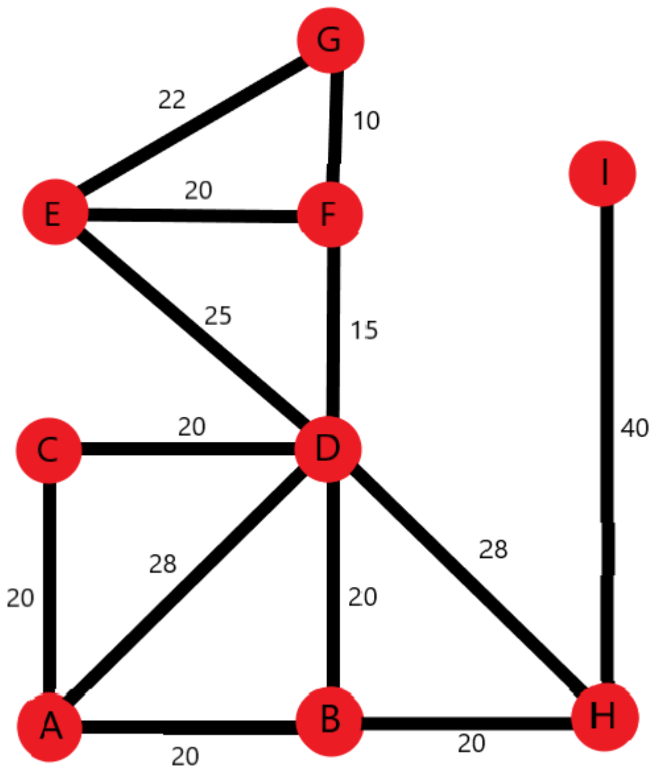
In my design I will first focus on developing my A* algorithm, since it is the core processing of my program. Next, I will figure out how I intend to structure my program and develop the data structures that I will use. Then, I will design my GUI and the code needed to generate and display it, before finally designing the module that will import all of the data and start the program.

A* algorithm

The first part of my project I decided to tackle was the A* algorithm, since it is the core functionality of my program. I started this process by finding pseudocode online for an A* algorithm. (Imms, 2016)

It was at this point that I decided to have my A* algorithm be a class because this would make it self contained and allow for variables to be easily passed throughout methods in the class as attributes. The pseudocode was already separated into three subroutines, however I decided that my class would use five methods: one to calculate g cost, one to calculate h cost, one to calculate all of a nodes neighbours, one to build the final quickest path and one to do all of the main processing.

My next step was to prototype this class in Python, which I did, however in order to test it I needed a graph. The real graph that I am using is too large and complex to have tested this prototype, so I created a much smaller and simpler graph which I used to make an adjacency list and coordinates, which I stored in dictionaries. Below is a visual representation of the graph. I also found the average running and walking speeds for a person, which I needed for calculating the time taken. These were 2.5m/s and 1.4m/s respectively. (Wikipedia, 2019) (OnAverage, n.d.)



Test number	Start location	End location	Route
1	A	E	E, D, B, A
2	A	G	G, F, D, A
3	A	I	I, H, B, D, A
4	E	I	I, H, B, D, F, E

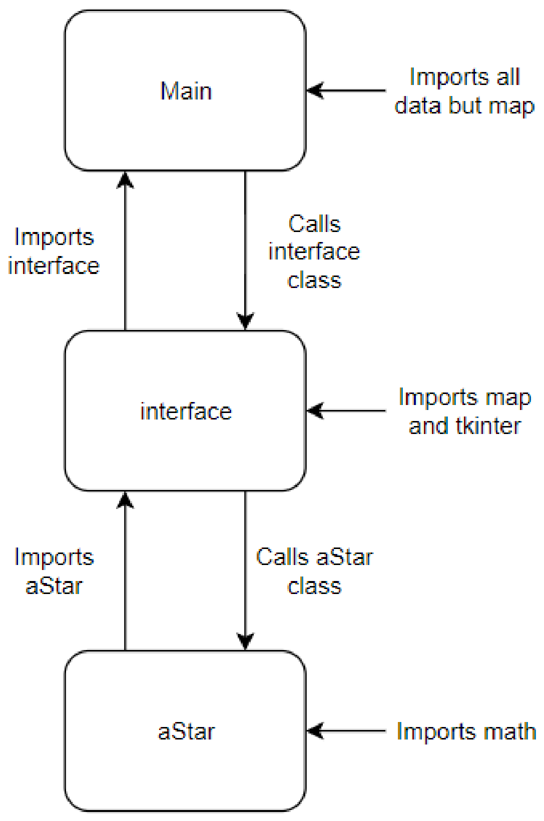
Early on when testing the program I found two things that caught my interest. The first was that it output the final route backwards (going from destination to start location) however I realised that this wouldn't matter when the route is displayed, since it is just a red line. The second was that sometimes the calculated route would not be the most efficient, as shown with tests 1, 3 and 4. I found that this was because once a neighbour is calculated its parent would never update, even if the most efficient route goes to it from a node that isn't its parent. I fixed this by adding code that updated the parent to whichever neighbouring node had the lowest g cost. After this bug was fixed I did some more testing which showed the error was gone.

Test number	Start location	End location	Route
1	A	E	E, D, A
2	A	G	G, F, D, A
3	A	I	I, H, B, A
4	E	I	I, H, D, E

Structure overview

After completing my first A* algorithm I decided to think about how I would structure the program as a whole. My program will be separated into three separate modules, Main, interface_class (which stores the interface class) and aStar_class (which stores the aStar class). Main will import all the data (except the map, this is imported later) and then create the GUI object from interface_class, which is imported into Main. The data will be passed as a parameter into GUI, which will then load the start screen. Main doesn't do anything after this. Upon initialisation, the interface class will assign all of the parameters to local variables and create the PathFinder object, passing the graph and coordinates in as parameters, using aStar_class, which is imported into interface_class. Once the user has pressed the start button, the main screen will be displayed with the map (which is imported when the main screen is generated) and a control panel where the user can run the program. When the user has entered their start and destination, they can press the calculate route button which will calculate the quickest route using the PathFinder object (the aStar class).

Below is a diagram show how the different files interact, as well as where data and modules are imported.



aStar class

Next, I decided to design my proper A* algorithm class.

aStar
- startNode: string - destinationNode: string - graph: dictionary - coordinates: dictionary - path: list
+ importValues (importStartNode, importDestinationNode): procedure + calculateRoute (): function - calculateg (node): function - calculateh (node): function - findNeighbours (node): function - buildPath (node): procedure + calculateTimeTaken (): function

The aStar class has five attributes: startNode, which will store the node the path needs to start from, destinationNode, which will store the node the path needs to end at, path, which will store the final path generated by the program, and graph and coordinates which are discussed in the data section. The aStar class also has seven methods: importValues, which allows the user to set their startNode and destinationNode, calculateRoute, which will use the A* algorithm to calculate the route and return it, calculateg and calculateh, which are called from calculateRoute and calculate the g and h costs respectively, findNeighbours, which is called from calculateRoute and returns a list of all of a nodes neighbours, buildPath, which will also be called from calculateRoute to create the final route once it has been fully calculated, and calculateTimeTaken, which will calculate and return the estimated time taken when walking and running.

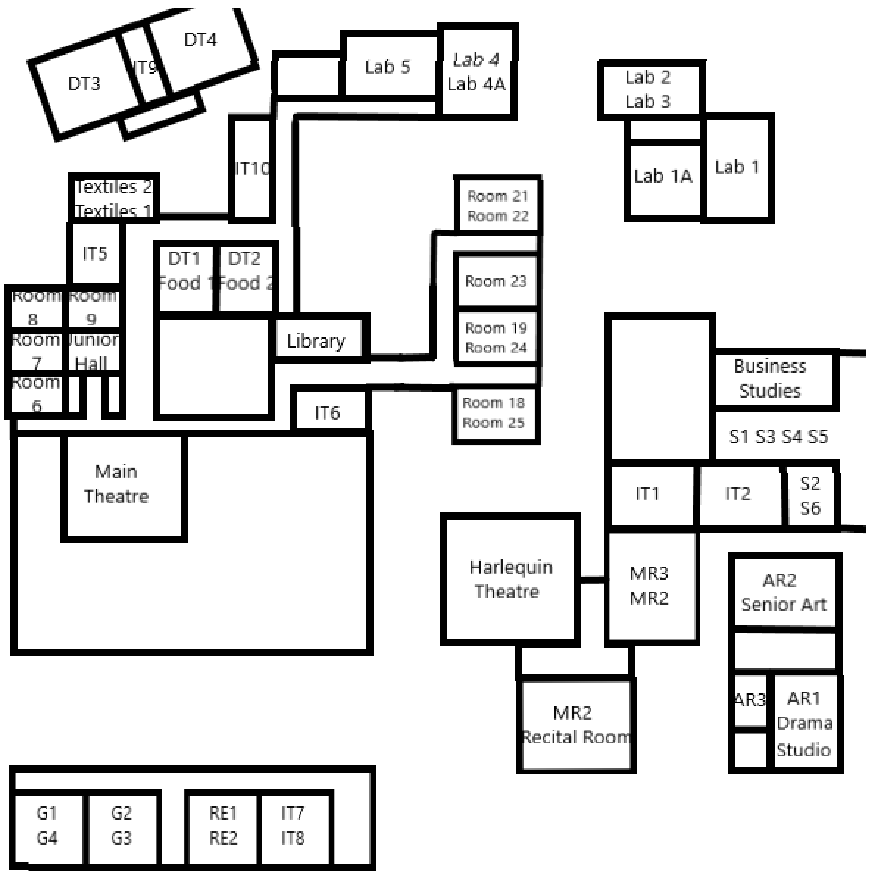
aStar class pseudocode

```
Procedure initialise (importGraph, importCoordinates):
       graph = importGraph
       coordinates = importCoordinates
       startNode = empty string
       destinationNode = empty string
       path = empty list
       walkSpeed = 1.4
       runSpeed = 2.5
       pixelsToMetres = 3.846

Procedure importValues (importStartNode, importDestinationNode):
       startNode = importStartNode
       destinationNode = importDestinationNode
```

```
Function calculateRoute ():
    openList = list containing startNode
    closedList = empty list
    startNode [g] = 0
    startNode [h] = calculateh (startNode)
    startNode [f] = startNode [g] + startNode [h]
    While openList is not empty:
        currentNode = openList [0]
        For items in openList:
            If item [f] is less than currentNode [f]:
                currentNode = item
        If currentNode is destinationNode:
            buildPath (currentNode)
            Return path
        Remove currentNode from openList
        Append currentNode to closedList
        For neighbour in findNeighbours (currentNode):
            If neighbour not in closedList:
                neighbour [parent] = currentNode
                neighbour [g] = calculateg (neighbour)
                neighbour [h] = calculateh (neighbour)
                neighbour [f] = neighbour [g] + neighbour [h]
                If neighbour not in openList     :
                    Append neighbour to openList
            Else:
                currentg = calculateg (currentNode)
                currentParent = currentNode [parent]
                currentNode [parent] = neighbour
                altg = calculateg (currentNode)
                if altg > currentg:
                    currentNode [parent] = currentParent

Function calculateg (node):
    g = 0
    While node [parent] is not startNode:
        g = g + graph [node] [parent]
        node = node [parent]
    Return g

Function calculateh (node):
    xDist = (coordinates [destinationNode] [x] - coordinates [node] [x])$^2$
    yDist = (coordinates [destinationNode] [y] - coordinates [node] [y])$^2$
    h = √(xDist + yDist)
    Return h

Function findNeighbours (node):
    neighbours = empty list
    For neighbour in graph [node]:
        Append neighbour to neighbours
    Return neighbours

Procedure buildPath (node):
    path = list containing node
    While node [parent] exists:
        node = node [parent]
        Append node to path

Function calculateTimeTaken ():
    routeLength = 0
    For index in range of path length - 2:
        Add graph [path [index]] [path [index + 1]] to routeLength
    walkTime = routeLength / (walkSpeed * pixelsToMetres)
    runtime = routeLength / (runSpeed * pixelsToMetres)
    Return walkTime and runtime
```

Map

After designing the aStar class I decided to create my map of the school, which is displayed below. It will be stored as a 600 by 600 pixel image and imported into the interface class. I created the map based off of the map shown in appendix two using the inbuilt Windows program paint 3D. On my map I decided to only have it display the layout of one floor because I thought that having multiple floors would make the map far more complicated and overwhelming for a young student. I also chose to make the map more linear than the one given to me by abstracting it in such a way that someone can clearly tell what an area or building represents but that the routes will only ever consist of multiple straight and horizontal lines, making it easier to visualise and for the user to memorise. It also means that the graph data structure is smaller.



Data

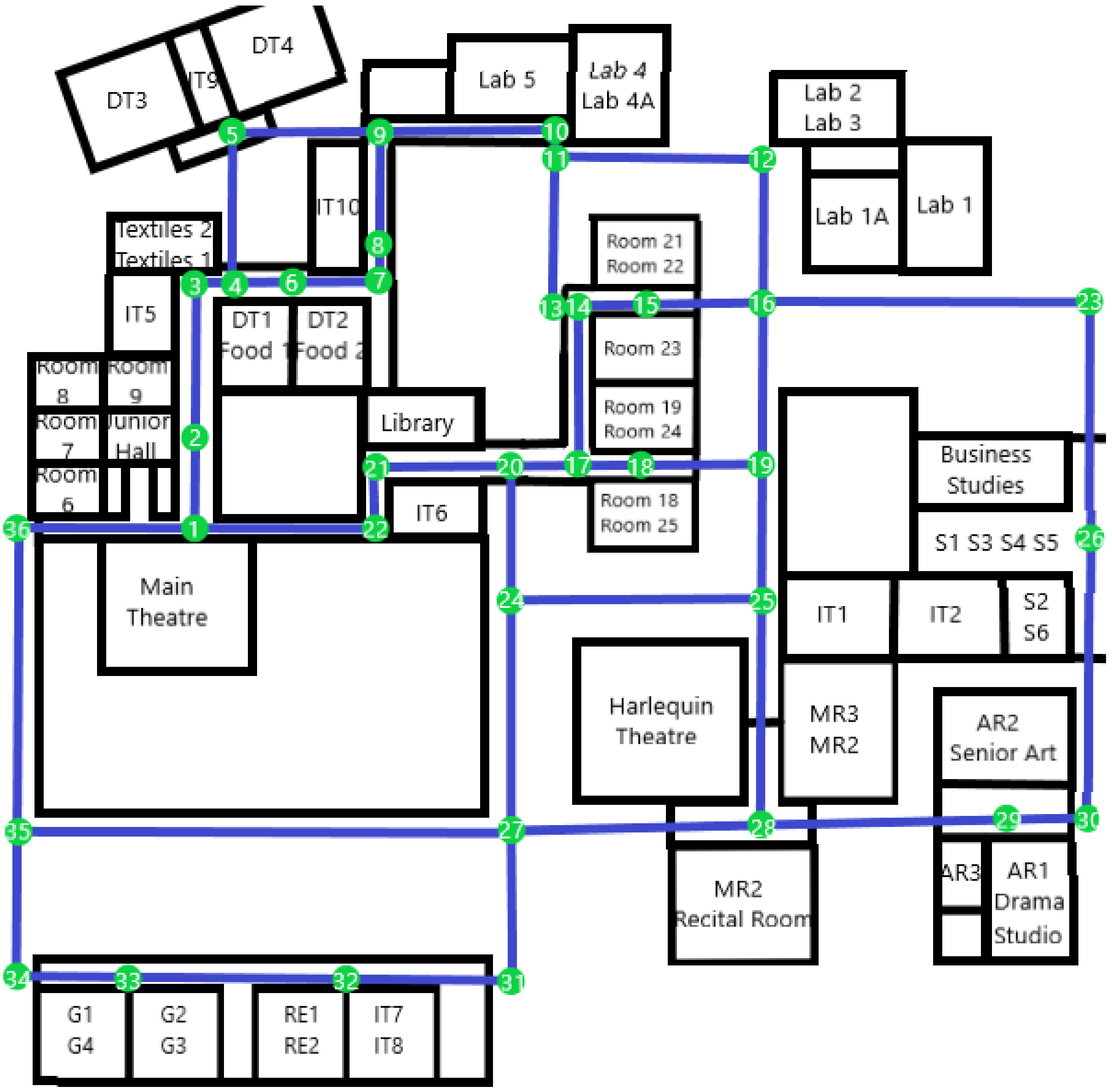
Next, I thought about the data I would use. All the data structures in my program will be stored in separate text documents and imported into the program.

Graph

The graph will represent the school's layout and it will be an adjacency list. It is weighted so I will use the dictionary data type with the format graph (current node (adjacent node: weight of edge)). The weights are the number of pixels between two nodes on the map.

Nodes	Dictionary representations
1	graph (1 (2: 50, 22: 95, 36: 100))
2	graph (2 (1: 50, 3: 85))
3	graph (3 (2: 85, 4: 20))
4	graph (4 (3: 20, 5: 80, 6: 35))
5	graph (5 (4: 80, 9: 85))
6	graph (6 (4: 35, 7: 45))
7	graph (7 (6: 45, 8: 25))
8	graph (8 (7: 25, 9: 60))
9	graph (9 (5: 85, 8: 60, 10: 100))
10	graph (10 (9: 100, 11: 15))
11	graph (11 (10: 15, 12: 110, 13: 80))
12	graph (12 (11: 110, 16: 80))
13	graph (13 (11: 115, 14: 10))
14	graph (14 (13: 10, 15: 35, 17: 90))
15	graph (15 (14: 35, 16: 60))
16	graph (16 (12: 110, 15: 60, 19: 90, 23: 175))
17	graph (17 (14: 90, 18: 35, 20: 45))
18	graph (18 (17: 35, 19: 60))
19	graph (19 (16: 90, 18: 60, 25: 75))
20	graph (20 (17: 45, 21: 70, 24: 75))
21	graph (21 (20: 70, 22: 35))
22	graph (22 (1: 95, 21: 35))
23	graph (23 (16: 175, 26: 130))
24	graph (24 (20: 75, 25: 140, 27: 120))
25	graph (25 (19: 75, 24: 140, 28: 120))
26	graph (26 (23: 130, 30: 150))
27	graph (27 (24: 120, 28: 140, 31: 80, 35: 265))
28	graph (28 (25: 120, 27: 140, 29: 130))
29	graph (29 (28: 130, 30: 50))
30	graph (30 (26: 150, 29: 50))
31	graph (31 (27: 80, 32: 90))
32	graph (32 (31: 90, 33: 120))
33	graph (33 (32: 120, 34: 60))
34	graph (34 (33: 60, 35: 80))
35	graph (35 (27: 265, 34: 80, 36: 165))
36	graph (36 (1: 100, 35: 165))

The dictionary shown above is larger than the one that appears in my analysis because I decided to use the canvas function in tkinter to draw the route when it is calculated. This function allows you to draw a straight line between two coordinates on the canvas, meaning that I will need to have a node for every corner on the graph, otherwise the route will not be drawn correctly and may pass through buildings on the map. Below is a visual representation of the new graph.



I chose to have nodes represent groups of rooms, rather than have a node for each room, because it greatly simplified the graph meaning that it would take less time for the program to calculate the quickest route and because the school is already well signposted so as long as a student is in the area they can find the room.

Coordinates

I will have another dictionary that stores coordinates for each node on the graph. This will be used to calculate the h cost, which is necessary for the A* algorithm to work. This dictionary will be in the form coordinates (node: (x position, y position). The x and y positions are also in pixels.

Nodes	Dictionary representations
1	coordinates (1: (110, 320))
2	coordinates (2: (110, 370))
3	coordinates (3: (110, 455))
4	coordinates (4: (130, 455))
5	coordinates (5: (130, 535))
6	coordinates (6: (165, 455))
7	coordinates (7: (210, 455))
8	coordinates (8: (210, 480))
9	coordinates (9: (210, 535))
10	coordinates (10: (310, 535))
11	coordinates (11: (310, 520))
12	coordinates (12: (420, 520))
13	coordinates (13: (310, 405))
14	coordinates (14: (320, 405))
15	coordinates (15: (355, 405))
16	coordinates (16: (420, 405))
17	coordinates (17: (320, 315))
18	coordinates (18: (355, 315))
19	coordinates (19: (420, 315))
20	coordinates (20: (275, 315))
21	coordinates (21: (205, 315))
22	coordinates (22: (205, 280))
23	coordinates (23: (595, 405))
24	coordinates (24: (275, 240))
25	coordinates (25: (420, 240))
26	coordinates (26: (595, 275))
27	coordinates (27: (275, 120))
28	coordinates (28: (420, 120))
29	coordinates (29: (545, 120))
30	coordinates (30: (595, 120))
31	coordinates (31: (275, 75))
32	coordinates (32: (185, 75))
33	coordinates (33: (65, 75))
34	coordinates (34: (10, 75))
35	coordinates (35: (10, 155))
36	coordinates (36: (10, 320))

The values used in graph and coordinates are in pixels, rather than metres. This is because the easiest way to draw a route using the canvas function in tkinter is by drawing straight lines between different coordinates on the canvas. These coordinates are in pixels, meaning less processing is required if the coordinate dictionary uses pixels rather than metres, since it doesn't have to convert. However, because the coordinates dictionary is also used for calculating the route, I had to use pixels for the weights of the graph.

The only difference made by using pixels instead of metres is that the values for estimated time taken were incorrect. In order to fix this, I measured the real life distance between nodes one and two and found they were 13 metres apart. Using this information, I calculated that to convert pixels to metres I would have to divide by 3.846. This means that I simply need to implement this conversion into my calculations for time taken and the times will be accurate again.

Rooms
I will store every room represented by the graph in the rooms list. This list will be used for the dropdown lists in the menu when a user is selecting their starting point and destination.

Nodes
Most nodes in the graph represent multiple rooms. These nodes and their corresponding rooms will be stored in the nodes dictionary. This will be in the format nodes (node: (rooms)).

Nodes	Dictionary representations
1	nodes (1: (‘Main Theatre’))
2	nodes (2: (‘Junior Hall’, ‘Room 6’, ‘Room 7’, ‘Room 8’, ‘Room 9’))
3	nodes (3: (‘IT5’, ‘Textiles 1’, ‘Textiles 2’))
5	nodes (5: (‘DT3’, ‘DT4’, ‘IT9’))
6	nodes (6: (‘DT1’, ‘DT2’, ‘Food 1’, ‘Food 2’))
8	nodes (8: (‘IT10’))
10	nodes (10: (‘Lab 4’, ‘Lab 4A’, ‘Lab 5’))
12	nodes (12: (‘Lab 1’, ‘Lab 1A’, ‘Lab 2’, ‘Lab 3’))
15	nodes (15: (‘Room 21’, ‘Room 22’, ‘Room 23’))
18	nodes (18: (‘Room 18’, ‘Room 19’, ‘Room 24’, ‘Room 25’))
21	nodes (21: (‘IT6’, ‘Library’))
26	nodes (26: (‘Business Studies’, ‘IT1’, ‘IT2’, ‘S1’, ‘S2’, ‘S3’, ‘S4’, ‘S5’, ‘S6’))
28	nodes (28: (‘Harlequin Theatre’, ‘MR1’, ‘MR2’, ‘MR3’, ‘Recital Room’))
29	nodes (29: (‘AR1’, ‘AR2’, ‘AR3’, ‘Drama Studio’, ‘Senior Art’))
32	nodes (32: (‘IT7’, ‘IT8’, ‘RE1’, ‘RE2’))
33	nodes (33: (‘G1’, ‘G2’, ‘G3’, ‘G4’))

This dictionary only includes nodes that represent rooms.

Instructions
To help the users understand how to use the program, a set of instructions on how to use the program will be stored in the instructions string.

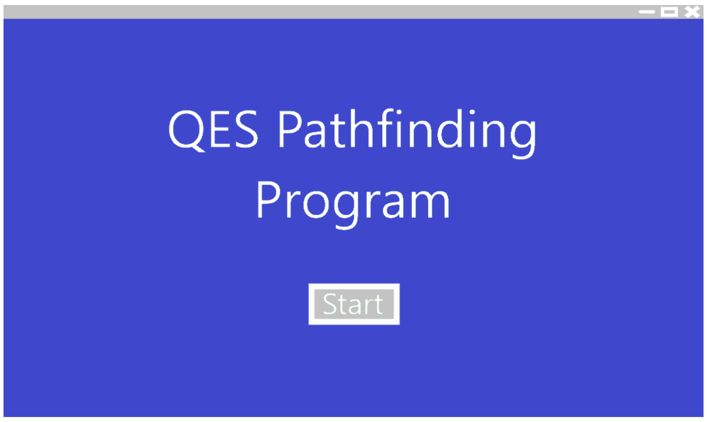
Testing A* algorithm with real data
Next, I decided to perform some brief tests to check my A* algorithm worked with real data.

Start location	End location	Route
5	29	29, 28, 25, 19, 18, 17, 14, 13, 11, 10, 9, 5
12	33	33, 32, 31, 27, 24, 25, 19, 16, 12
10	31	31, 27, 24, 20, 17, 14, 13, 11, 10
26	36	36, 1, 22, 21, 20, 17, 18, 19, 16, 23, 26
5	34	34, 35, 36, 1, 2, 3, 4, 5

Whilst this is not thorough testing, it shows that the algorithm does work with the real data set, however I will do more systematic testing later.

Startup

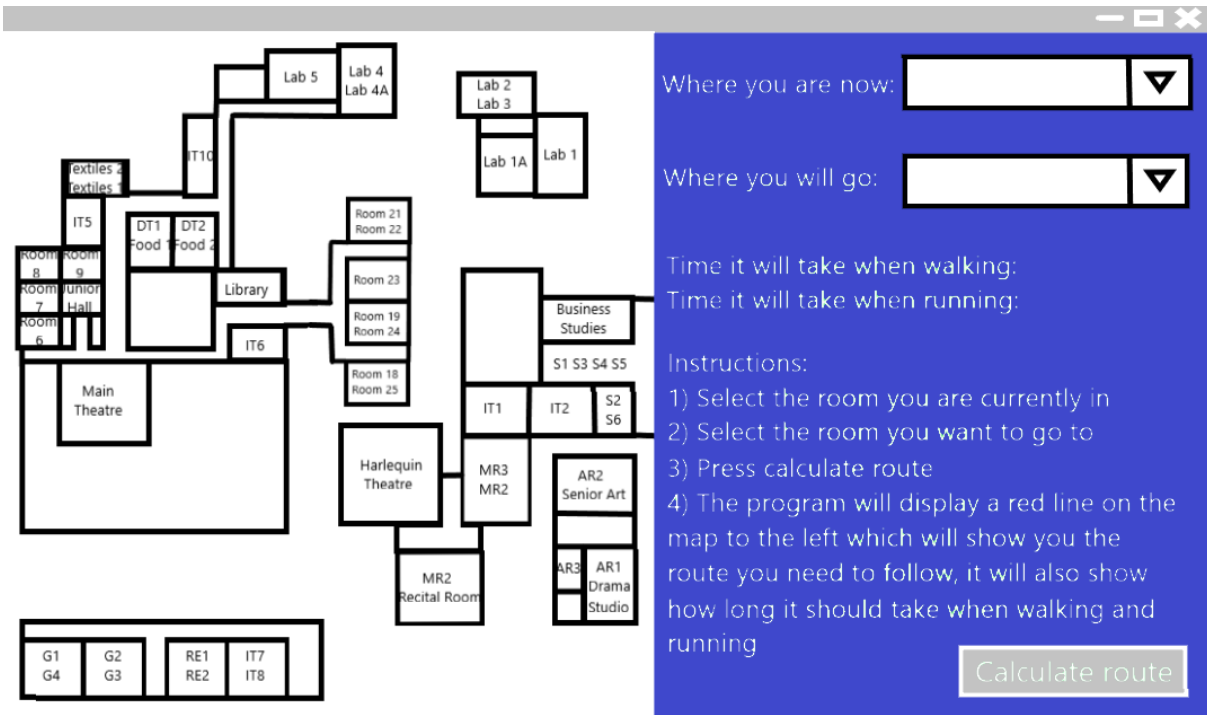
After testing with the proper data, I decided to start working on my GUI, but first I decided to think about what happens when the program first starts up. When the program starts the user will be shown a start screen like what is shown below. From this screen the user can choose to start the program.



While the start screen is being displayed, data will be being loaded from an external file in the main program. This data will be the graph dictionary, the node coordinates dictionary, the nodes dictionary and the rooms list, which are all explained in the data section below, as well as the instructions on how to run the program. After it has been loaded, it will all be imported as a parameter into the interface class, then the graph and node coordinates will be imported into the aStar class.

GUI

After the user has selected start on the start screen the main GUI will be displayed, which will look something like what is shown below.



Where you are now:

Where you will go:

Time it will take when walking:

Time it will take when running:

Instructions:

1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map to the left which will show you the route you need to follow, it will also show how long it should take when walking and running

Calculate route

This window displays a map of the school, two dropdown boxes labelled “Where you are now:” and “Where you will go:” which will allow the user to select their start and destination rooms, two lines where the estimated time taken when running and walking will be displayed, a list of instructions explaining how to work the program and a calculate route button.

interface class

interface
- root: Tk - rooms: list - instructions: string - coordinates: dictionary - nodes: dictionary - walkTimeString: string - runTimeString: string - lines: list - circles: list
+ startScreen (): procedure - nextScreen (): procedure - mainScreen (): procedure - map (): procedure - controlPanel (): procedure - startLocation (): procedure - endLocation (): procedure - deleteRoute (x): procedure - runAStarAlgorithm (): procedure - displayRoute (path): procedure

The interface class has nine attributes: root, which is a custom variable in the tkinter module that is effectively the window the GUI is shown in, walkTimeString, that stores the string to be shown displaying the estimated time when walking, runTimeString, that stores the string to be shown displaying the estimated time when running, lines, that stores all of the lines displayed on the map when the route is displayed, circles, which stores all of the circles displayed on the map when the route is displayed, and rooms, instructions, coordinates and nodes that are all discussed in the data section. The interface class also has nine methods: startScreen, which displays the programs start screen, nextScreen, which is activated when the user hits the start button and deletes the start screen, mainScreen, which is called after the start screen has been destroyed in nextScreen and displays the main screen, map, which displays the map on the main screen, controlPanel, which displays the right side of main screen where the user controls the program, startLocation and endLocation, which display their respective option boxes and text where the user enters their start location and destination, deleteRoute, which deletes the previously displayed route off of the canvas and resets walkTimeString and runTimeString, runAStarAlgorithm, which is called when the user presses the calculate route button and calls the aStar class to generate the shortest path and time taken, and displayRoute, which uses the path to display the route on the map.

Interface class pseudocode

```
Procedure initialise (importRooms, importInstructions, importGraph,
importCoordinates, importNodes):
       root = Tk()
       rooms = importRooms
       instructions = importInstructions
       coordinates = importCoordinates
       nodes = importNodes
       walkTimeString = "Time it will take when walking: "
       runTimeString = "Time it will take when running: "
       lines = empty list
       circles = empty list
       PathFinder = aStar (importGraph, importCoordinates)
```

```
Procedure startScreen ():
       startScreenFrame = frame with blue background
       titleLabel = label in startScreenFrame
       Configure titleLabel with white text, blue background and text reading “QES
       Pathfinding Program”
       startButton = button in startScreenFrame
       Configure startButton with text reading “Start” and command being nextScreen
       Pack startScreenFrame
       Pack titleLabel to top
       Pack startButton to bottom

Procedure nextScreen ():
       Destroy startScreenFrame
       mainScreen ()

Procedure mainScreen ():
       mainScreenFrame = frame with blue background
       map ()
       controlPanel ()
       Pack mapFrame to left
       Pack controlPanelFrame to right

Procedure map ():
       mapFrame = frame in mainScreenFrame
       mapCanvas = canvas in mapFrame
       Configure mapCanvas with width of 600 pixels and height of 600 pixels
       mapImage = photo image map.png
       Create mapImage at coordinates 300, 300 in mapCanvas
       Pack mapCanvas

Procedure controlPanel ():
       controlPanelFrame = frame in mainScreenFrame with blue background
       startLocation ()
       endLocation ()
       walkTimeLabel = label in controlPanelFrame
       Configure walkTimeLabel with white text, blue background and walkTimeString
       as text
       runTimeLabel = label in controlPanelFrame
       Configure runTimeLabel with white text, blue background and runTimeString as
       text
       instructionsLabel = label in controlPanelFrame
       Configure instructionsLabel with white text, blue background and
       instructions as text
       calculateRouteButton = button in controlPanelFrame
       Configure calculateRouteButton with text reading “Calculate Route” and
       command being runAStarAlgorithm
       Pack startLocationFrame to top
       Pack endLocationFrame to top
       Pack walkTimeLabel to top
       Pack runTimeLabel to top
       Pack instructionsLabel to top
       Pack calculateRouteButton to bottom

Procedure startLocation ():
       startLocationFrame = frame in controlPanelFrame with blue background
       startlocationLabel = label in startLocationFrame
       Configure startLocationLabel with white text, blue background and text
       reading “Where you are now:”
       selectedStartLocation = string
       startLocationMenu = option menu in startLocationFrame
       Configure startLocationMenu with options being rooms list and selection
       being selectedStartLocation
       Pack startLocationLabel to left
       Pack startLocationMenu to right
```

```
Procedure endLocation ():
    endLocationFrame = frame in controlPanelFrame with blue background
    endlocationLabel = label in endLocationFrame
    Configure endLocationLabel with white text, blue background and text reading
    "Where you are going:"
    selectedEndLocation = string
    endLocationMenu = option menu in endLocationFrame
    Configure endLocationMenu with options being rooms list and selection being
    selectedEndLocation
    Pack endLocationLabel to left
    Pack endLocationMenu to right

Procedure deleteRoute (x):
    For line in lines:
        Delete line from mapCanvas
    For circle in circles:
        Delete circle from mapCanvas
    walkTimeString = "Time it will take when walking: "
    runTimeString = "Time it will take when running: "

Procedure runAStarAlgorithm ():
    If selectedStartLocation is not empty and selectedEndLocation is not empty
    and selectedStartLocation is not selectedEndLocation:
        For node in nodes:
            If selectedStartLocation is in nodes [node]:
                startNode = node
            If selectedEndLocation is in nodes [node]:
                destinationNode = node
        From PathFinder call importValues (startNode, endNode)
        path = from PathFinder call calculateRoute ()
        displayRoute (path)
        walkTime, runTime = from PathFinder call calculateTimeTaken ()
        Set walkTimeString text to "Time it will take when walking: " +
        walkTime as a string
        Set runTimeString text to "Time it will take when running: " + runTime
        as a string

Procedure displayRoute (path):
    For index in range of path length - 1:
        Append new line on mapCanvas between coordinates of path [index] and
        path [index + 1] with red fill and width of 5 pixels to lines
        Append new oval on mapCanvas at coordinates of path [index] with red
        fill and width of 2 pixels to circles
    Append new oval on mapCanvas at coordinates of path [0] with red fill and
    width of 4 pixels to circles
    Append new oval on mapCanvas at coordinates of path [path length - 1] with
    red fill and width of 4 pixels to circles
```

Main

Finally, I decided to design the Main module. Main is where the program is run from and where all of the values for the data structures are imported from the text files. Main will have five subroutines: createRooms, createInstructions, createGraph, createCoordinates and createNodes. All of these subroutines take a loaded file as a parameter and use that file to create their respective data structures before returning it. When Main is run it simply opens the different text files and calls the subroutines before creating the GUI object from the interface class, with the data structures as parameters, then calling the startScreen procedure from GUI.

Main pseudocode

```
Function createRooms (file):
    rooms = empty list
    For line in file:
        Strip line
        Append line to rooms
    Return rooms

Function createInstructions (file):
    instructions = empty string
    For line in file:
        Add line to instructions
    Return instructions

Function createGraph (file):
    graph = empty dictionary
    tempString = empty string
    lineNo = 1
    For line in file:
        graph [lineNo as a string] = empty dictionary
        For character in line:
            If character is ',':
                graph [lineNo as a string] [key] = tempString as an
                integer
                tempString = empty string
            Else if character is ':':
                key = tempString
                tempString = empty string
            Else:
                Add character to tempString
        graph [lineNo as a string] [key] = tempString as an integer
        tempString = empty string
        Increment lineNo by 1
    Return graph

Function createCoordinates (file):
    coordinates = empty dictionary
    tempString = empty string
    lineNo = 1
    For line in file:
        coordinates [lineNo as a string] = empty list
        For character in line:
            If character is ',':
                Append tempString as an integer to coordinates [lineNo
                as a string]
                tempString = empty string
            Else:
                Add character to tempString
        Append tempString as an integer to coordinates [lineNo as a string]
        tempString = empty string
        Increment lineNo by 1
    Return coordinates
```

```
Function createNodes (file):
    nodes = empty dictionary
    tempString = empty string
    For line in file:
        For character in line:
            If character is ',':
                Append tempString to nodes [key]
                tempString = empty string
            Else if character is ':':
                key = tempString
                nodes [key] = empty list
                tempString = empty string
            Else:
                Add character to tempString
        Strip '\n' from tempString
        Append tempString to nodes [key]
        tempString = empty string
    Return nodes

Open rooms.txt as file
rooms = createRooms (file)
Close rooms.txt

Open instructions.txt as file
instructions = createInstructions (file)
Close instructions.txt

Open graph.txt as file
graph = createGraph (file)
Close graph.txt

Open coordinates.txt as file
coordinates = createCoordinates (file)
Close coordinates.txt

Open nodes.txt as file
nodes = createNodes (file)
Close nodes.txt

GUI = interface (rooms, instructions, graph, coordinates, nodes)
From GUI call startScreen ()
```

Testing

Test strategy

A* algorithm

Test	Testing type	Purpose
Testing A* algorithm works with test graph		To ensure that the A* algorithm finds the most efficient route at a basic level with a small graph
Testing A* algorithm performs cost calculations correctly		To ensure that correct values for a nodes g, h and f cost are properly calculated
Testing A* algorithm finds most efficient route		To ensure that the A* algorithm finds the most efficient route between two points on the graph
Testing A* algorithm calculates time taken correctly		To ensure that the estimated times taken are properly calculated and are sensible values

Data structures

Test	Testing type	Purpose
Testing createRooms function		To ensure that every value imported from the file is stored in the rooms list
Testing createInstructions function		To ensure that the text imported from the file is properly formatted and stored in the instructions string
Testing createGraph function		To ensure that every value imported from the file is stored in the graph dictionary and can be properly accessed
Testing createCoordinates function		To ensure that every value imported from the file is stored in the coordinates dictionary and can be properly accessed
Testing createNodes function		To ensure that every value imported from the file is stored in the nodes dictionary and can be properly accessed
Testing what happens when file is missing		To ensure that if any of the files are missing the program doesn't crash
Testing what happens when file is corrupted		To ensure that if any of the files are corrupted or edited to not have the correct format the program doesn't crash

GUI

Test	Testing type	Purpose
Testing the start screen		To ensure that the start screen is generated and displayed as expected
Testing the main screen		To ensure that the main screen is generated and displayed as expected
Testing the dropdown lists		To ensure that the dropdown lists allow users to select options and display said option in the box
Testing the calculate route button		To ensure that the calculate route button works and is not called when two rooms on the same node are selected
Testing route is cleared when new option selected		To ensure that the displayed route is removed from the canvas when a new option is selected

Beta testing

A* algorithm

Testing A* algorithm works with test graph

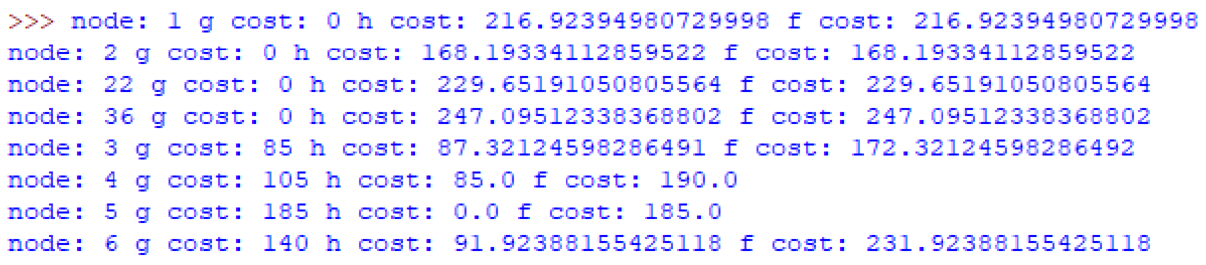
I already performed this test in the A* algorithm section of my design (page 18). I was confident following this proof of concept that the core functionality was working.

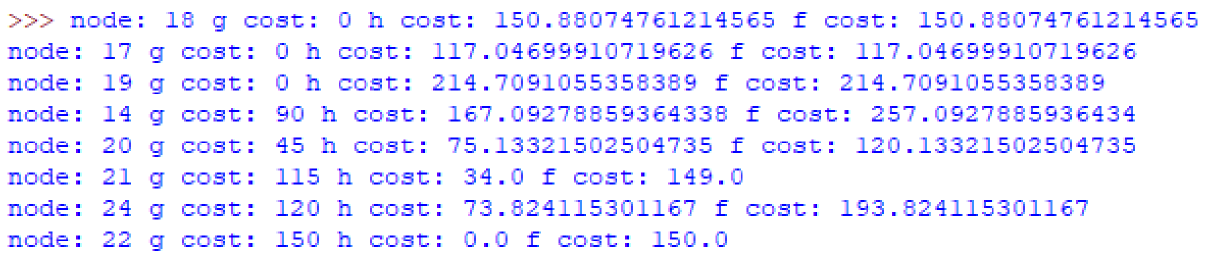
Testing A* algorithm performs cost calculations correctly

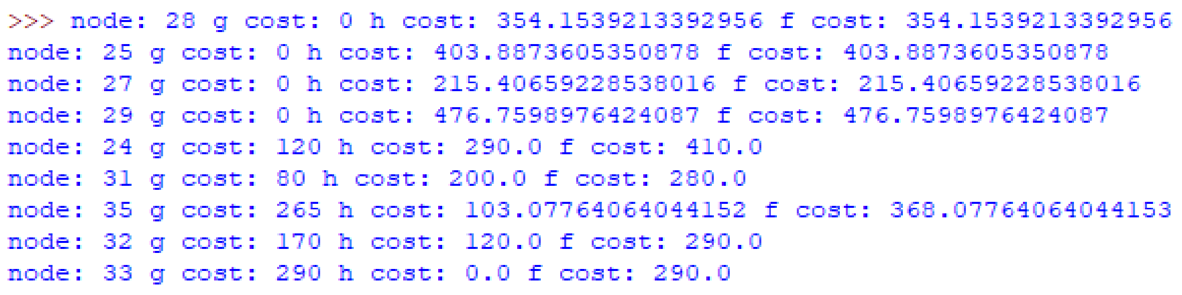
For the following tests I will only use routes that go between nodes that are linked to rooms by the nodes dictionary because in the real program you cannot go to or from a node which does not represent one or more rooms. For this test I will use short routes because I only need a few nodes to ensure that the cost calculations are being performed properly. I will use print statements to take interim values of the current neighbour being calculated and the g, h and f costs as they are calculated. I will also print the node and the costs of the start node first.

Route	Predicted node	Predicted g cost	Predicted h cost	Predicted f cost
1 – 5	1, 2, 22, 36, 3, 4, 5, 6	0, 50, 95, 100, 135, 155, 235, 190	216.9, 168.2, 229.7, 247.1, 87.3, 85.0, 0.0, 91.9	216.9, 218.2, 324.7, 347.1, 222.3, 240.0, 235.0, 281.9
18 – 22	18, 17, 19, 14, 20, 21, 24, 22	0, 35, 60, 125, 80, 150, 155, 185	150.9, 117.0, 214.7, 167.1, 75.1, 34.0, 73.8, 0.0	150.9, 152.0, 274.7, 292.1, 155.1, 184.0, 228.8, 185.0
28 – 33	28, 25, 27, 29, 24, 31, 35, 32, 33	0, 120, 140, 130, 260, 220, 405, 310, 430	354.2, 403.9, 215.4, 476.8, 290.0, 200.0, 103.1, 120.0, 0.0	354.2, 523.9, 355.4, 606.8, 550.0, 420.0, 508.1, 430.0, 430.0

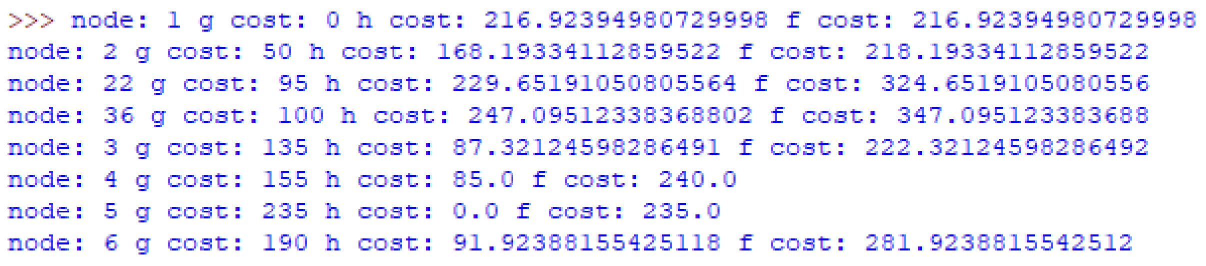
Below is an image of my tests. Whilst the h costs are correct and the f costs are correct for the g and h costs, the g costs are smaller than they should be and different to my predicted values.

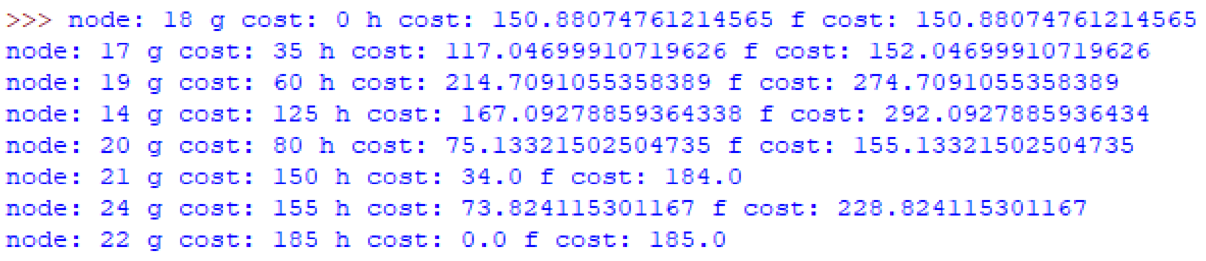

```
>>> node: 1 g cost: 0 h cost: 216.92394980729998 f cost: 216.92394980729998
node: 2 g cost: 0 h cost: 168.19334112859522 f cost: 168.19334112859522
node: 22 g cost: 0 h cost: 229.65191050805564 f cost: 229.65191050805564
node: 36 g cost: 0 h cost: 247.09512338368802 f cost: 247.09512338368802
node: 3 g cost: 85 h cost: 87.32124598286491 f cost: 172.32124598286492
node: 4 g cost: 105 h cost: 85.0 f cost: 190.0
node: 5 g cost: 185 h cost: 0.0 f cost: 185.0
node: 6 g cost: 140 h cost: 91.92388155425118 f cost: 231.92388155425118
```

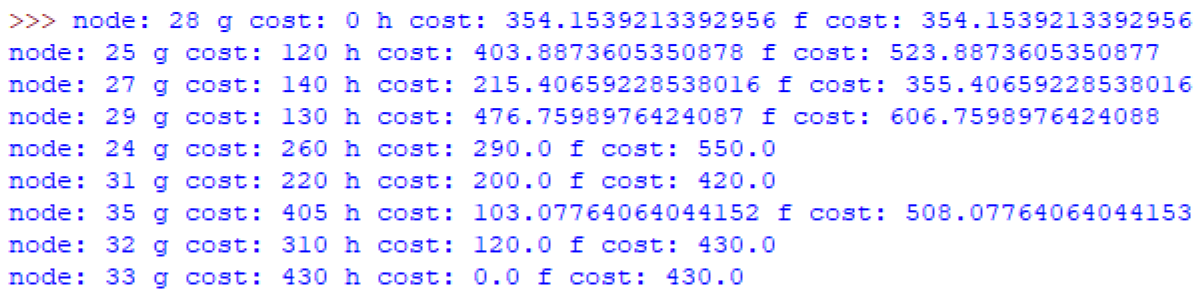

```
>>> node: 18 g cost: 0 h cost: 150.88074761214565 f cost: 150.88074761214565
node: 17 g cost: 0 h cost: 117.04699910719626 f cost: 117.04699910719626
node: 19 g cost: 0 h cost: 214.7091055358389 f cost: 214.7091055358389
node: 14 g cost: 90 h cost: 167.09278859364338 f cost: 257.0927885936434
node: 20 g cost: 45 h cost: 75.13321502504735 f cost: 120.13321502504735
node: 21 g cost: 115 h cost: 34.0 f cost: 149.0
node: 24 g cost: 120 h cost: 73.824115301167 f cost: 193.824115301167
node: 22 g cost: 150 h cost: 0.0 f cost: 150.0
```


```
>>> node: 28 g cost: 0 h cost: 354.1539213392956 f cost: 354.1539213392956
node: 25 g cost: 0 h cost: 403.8873605350878 f cost: 403.8873605350878
node: 27 g cost: 0 h cost: 215.40659228538016 f cost: 215.40659228538016
node: 29 g cost: 0 h cost: 476.7598976424087 f cost: 476.7598976424087
node: 24 g cost: 120 h cost: 290.0 f cost: 410.0
node: 31 g cost: 80 h cost: 200.0 f cost: 280.0
node: 35 g cost: 265 h cost: 103.07764064044152 f cost: 368.07764064044153
node: 32 g cost: 170 h cost: 120.0 f cost: 290.0
node: 33 g cost: 290 h cost: 0.0 f cost: 290.0
```

I realised that this was because my g cost calculation did not include the start node, so I fixed this and tested again. Below are the same tests but this time working as expected.


```
>>> node: 1 g cost: 0 h cost: 216.92394980729998 f cost: 216.92394980729998
node: 2 g cost: 50 h cost: 168.19334112859522 f cost: 218.19334112859522
node: 22 g cost: 95 h cost: 229.65191050805564 f cost: 324.6519105080556
node: 36 g cost: 100 h cost: 247.09512338368802 f cost: 347.095123383688
node: 3 g cost: 135 h cost: 87.32124598286491 f cost: 222.32124598286492
node: 4 g cost: 155 h cost: 85.0 f cost: 240.0
node: 5 g cost: 235 h cost: 0.0 f cost: 235.0
node: 6 g cost: 190 h cost: 91.92388155425118 f cost: 281.9238815542512
```


```
>>> node: 18 g cost: 0 h cost: 150.88074761214565 f cost: 150.88074761214565
node: 17 g cost: 35 h cost: 117.04699910719626 f cost: 152.04699910719626
node: 19 g cost: 60 h cost: 214.7091055358389 f cost: 274.7091055358389
node: 14 g cost: 125 h cost: 167.09278859364338 f cost: 292.0927885936434
node: 20 g cost: 80 h cost: 75.13321502504735 f cost: 155.13321502504735
node: 21 g cost: 150 h cost: 34.0 f cost: 184.0
node: 24 g cost: 155 h cost: 73.824115301167 f cost: 228.824115301167
node: 22 g cost: 185 h cost: 0.0 f cost: 185.0
```



```
>>> node: 28 g cost: 0 h cost: 354.1539213392956 f cost: 354.1539213392956
node: 25 g cost: 120 h cost: 403.8873605350878 f cost: 523.8873605350877
node: 27 g cost: 140 h cost: 215.40659228538016 f cost: 355.40659228538016
node: 29 g cost: 130 h cost: 476.7598976424087 f cost: 606.7598976424088
node: 24 g cost: 260 h cost: 290.0 f cost: 550.0
node: 31 g cost: 220 h cost: 200.0 f cost: 420.0
node: 35 g cost: 405 h cost: 103.07764064044152 f cost: 508.07764064044153
node: 32 g cost: 310 h cost: 120.0 f cost: 430.0
node: 33 g cost: 430 h cost: 0.0 f cost: 430.0
```

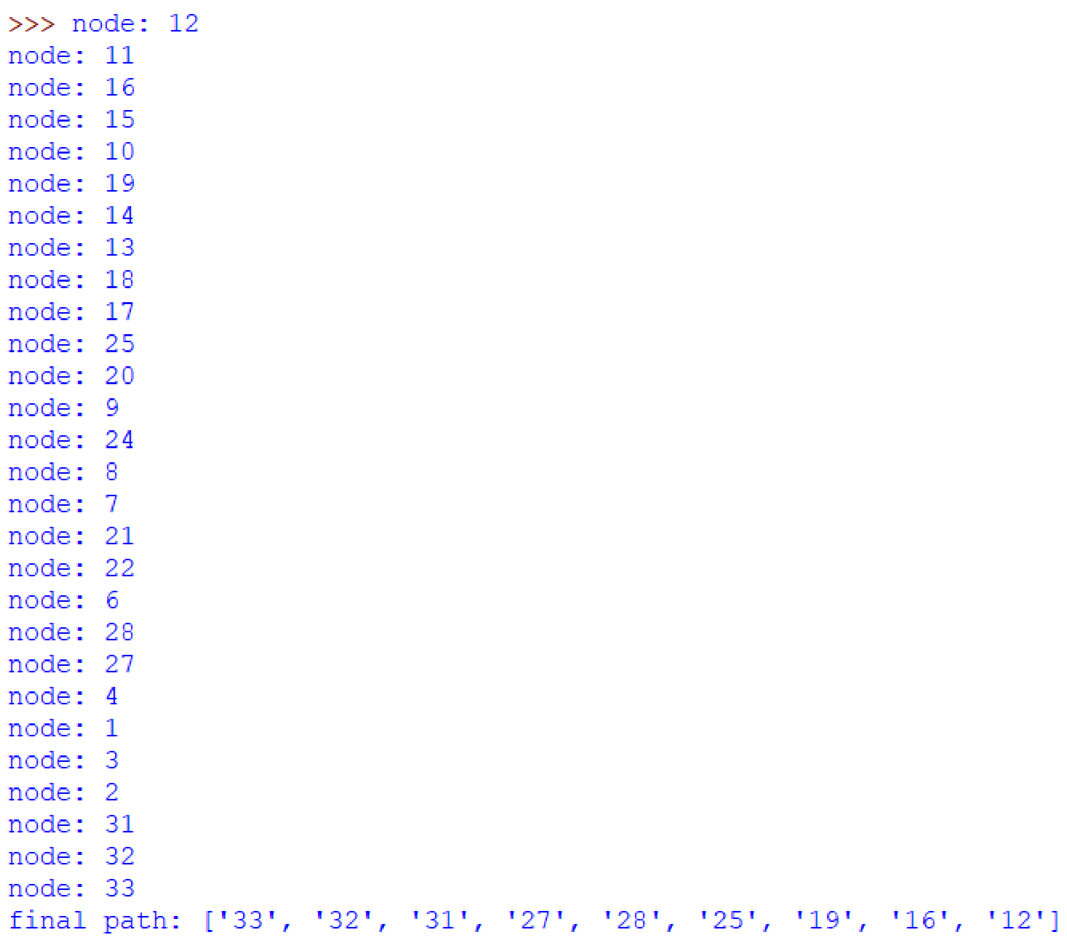
Testing A* algorithm finds most efficient route

I already performed this test in the testing A* algorithm with real data section of my design (page 26), however, due to the importance of this test and the fact that I did not take interim values I am redoing it.

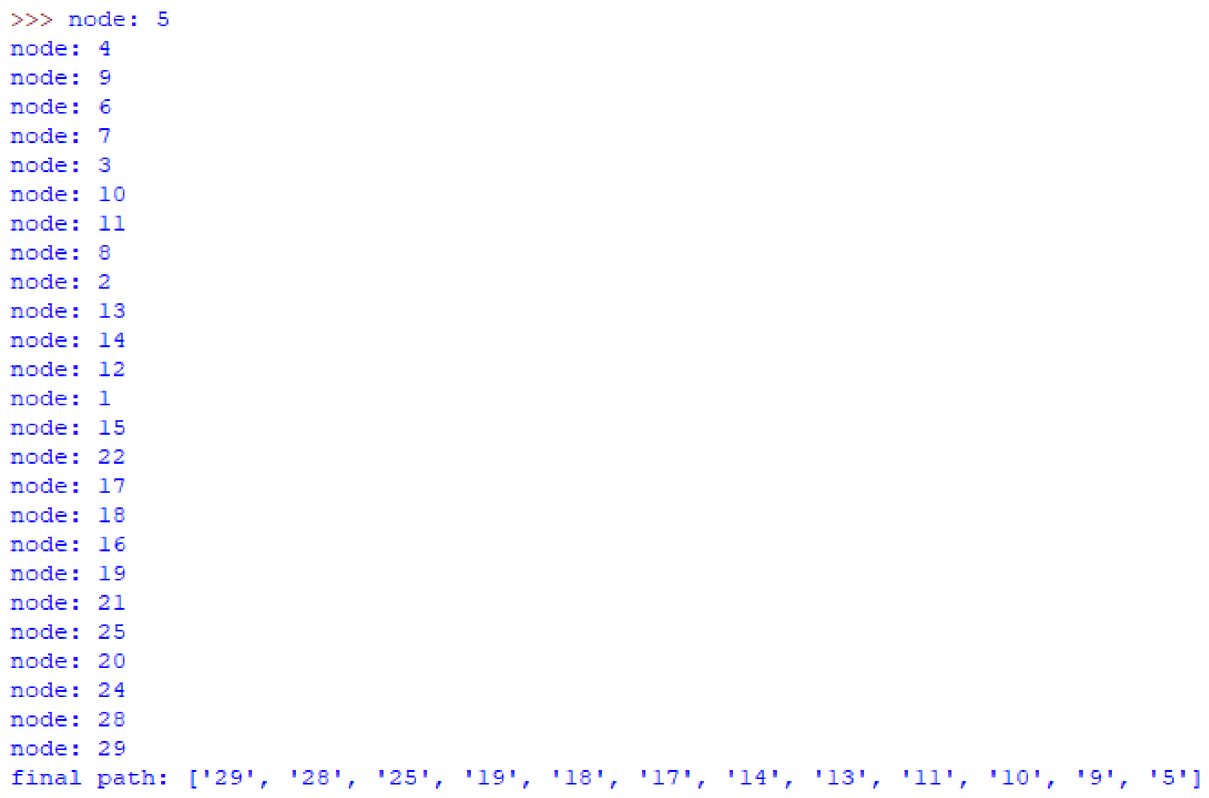
For this test I will test the longest possible routes going across the graph. I will use print statements to get interim values for every node that is activated before printing the final route at the end.

Route	Predicted node	Predicted final path
12 – 33	12, 11, 16, 15, 10, 19, 14, 13, 18, 17, 25, 20, 9, 24, 8, 7, 21, 22, 6, 28, 27, 4, 1, 3, 2, 31, 32, 33	33, 32, 31, 27, 28, 25, 19, 16, 12
5 – 29	5, 4, 9, 6, 7, 3, 10, 11, 8, 2, 13, 14, 12, 1, 15, 22, 17, 18, 16, 19, 21, 25, 20, 24, 28, 29	29, 28, 25, 19, 18, 17, 14, 13, 11, 10, 9, 5
1 – 26	1, 22, 21, 20, 17, 18, 19, 2, 25, 3, 14, 4, 6, 15, 7, 16, 13, 8, 36, 23, 26	26, 23, 16, 19, 18, 17, 20, 21, 22, 1

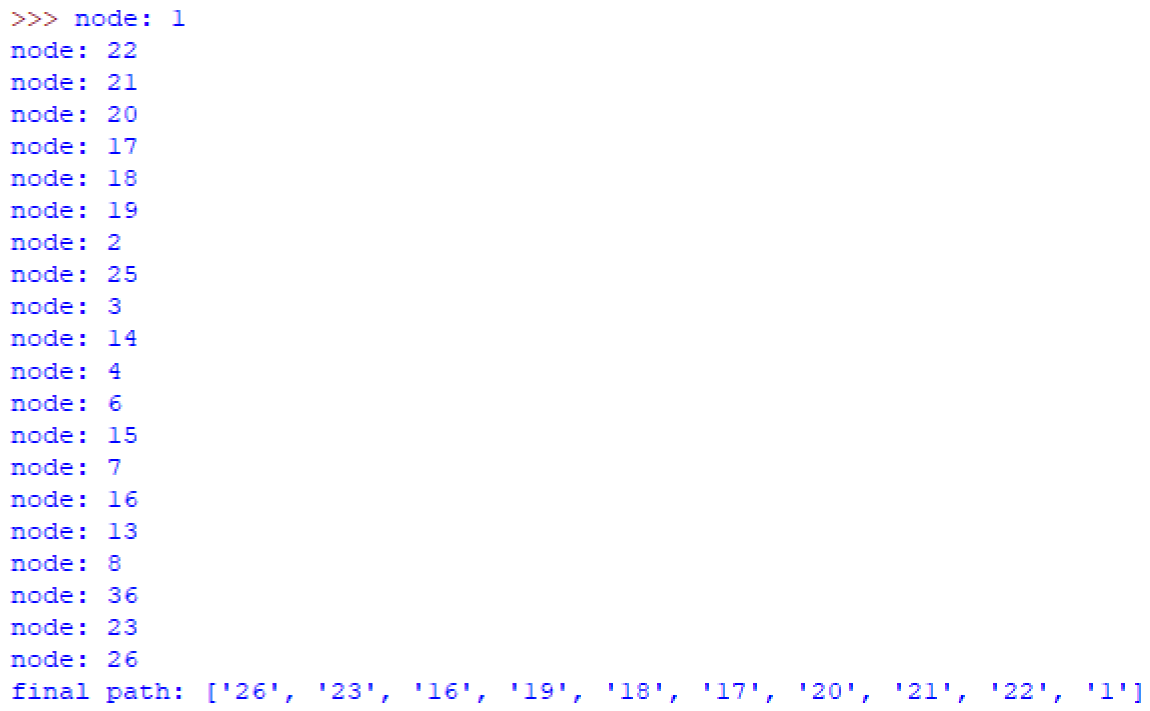
Below are the images of my tests. They match my predictions showing me that the program is working as expected.



```
>>> node: 12
node: 11
node: 16
node: 15
node: 10
node: 19
node: 14
node: 13
node: 18
node: 17
node: 25
node: 20
node: 9
node: 24
node: 8
node: 7
node: 21
node: 22
node: 6
node: 28
node: 27
node: 4
node: 1
node: 3
node: 2
node: 31
node: 32
node: 33
final path: ['33', '32', '31', '27', '28', '25', '19', '16', '12']
```



```
>>> node: 5
node: 4
node: 9
node: 6
node: 7
node: 3
node: 10
node: 11
node: 8
node: 2
node: 13
node: 14
node: 12
node: 1
node: 15
node: 22
node: 17
node: 18
node: 16
node: 19
node: 21
node: 25
node: 20
node: 24
node: 28
node: 29
final path: ['29', '28', '25', '19', '18', '17', '14', '13', '11', '10', '9', '5']
```



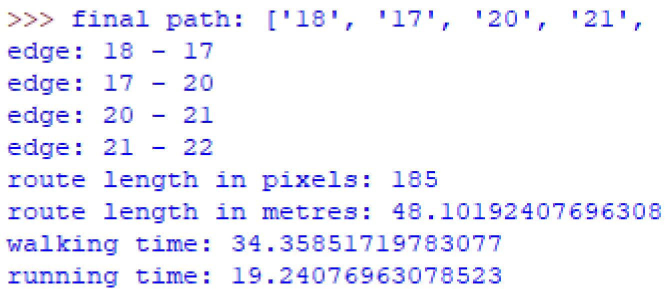
```
>>> node: 1
node: 22
node: 21
node: 20
node: 17
node: 18
node: 19
node: 2
node: 25
node: 3
node: 14
node: 4
node: 6
node: 15
node: 7
node: 16
node: 13
node: 8
node: 36
node: 23
node: 26
final path: ['26', '23', '16', '19', '18', '17', '20', '21', '22', '1']
```

<u>Testing A* algorithm calculates time taken correctly</u>

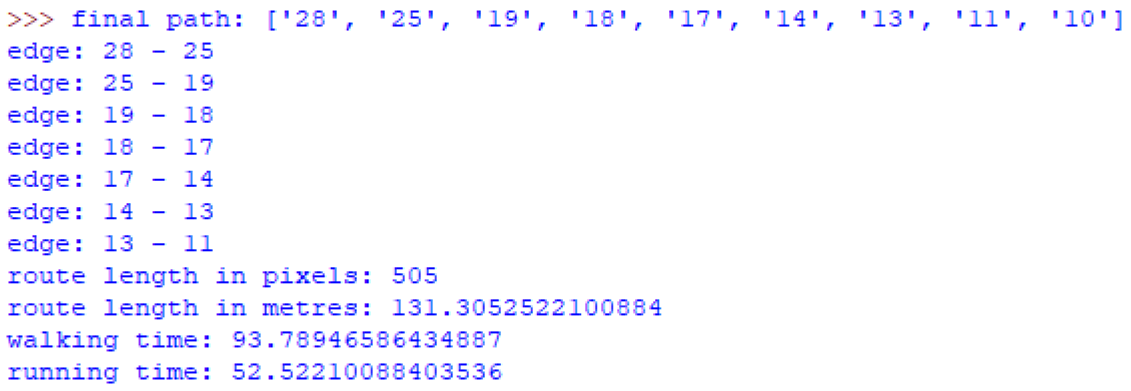
For this test I will use a variety of different length routes in order to get a large variety of time calculations. I will use print statements to get the final path, each edge that has its weight added to the route length, the final calculated length of the route in pixels and metres and the estimated walking and running times.

Route	Predicted final path	Predicted edge	Predicted route length in pixels	Predicted route length in metres	Predicted walking time	Predicted running time
1 – 18	18, 17, 20, 21, 22, 1	18 – 17, 17 – 20, 20 – 21, 21 – 22, 22 – 1	280	72.8	52.0	29.1
10 – 28	28, 25, 19, 18, 17, 14, 13, 11, 10	28 – 25, 25 – 19, 19 – 18, 18 – 17, 17 – 14, 14 – 13, 13 – 11, 11 – 10	520	135.2	96.6	54.1
5 – 33	33, 34, 35, 36, 1, 2, 3, 4, 5	33 – 34, 34 – 35, 35 – 36, 36 – 1, 1 – 2, 2 – 3, 3 – 4, 4 – 5	640	166.4	118.9	66.6

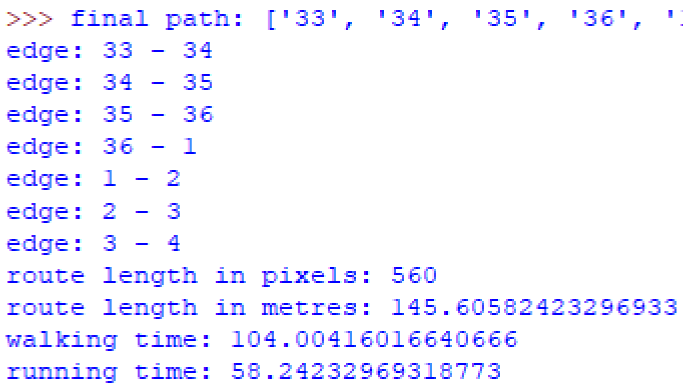
Below are the images of my tests. Whilst the final paths are correct, not all of the edges are visited meaning that the route lengths and times taken are lower than expected.



```
>>> final path: ['18', '17', '20', '21', '22', '1']
edge: 18 - 17
edge: 17 - 20
edge: 20 - 21
edge: 21 - 22
route length in pixels: 185
route length in metres: 48.10192407696308
walking time: 34.35851719783077
running time: 19.24076963078523
```

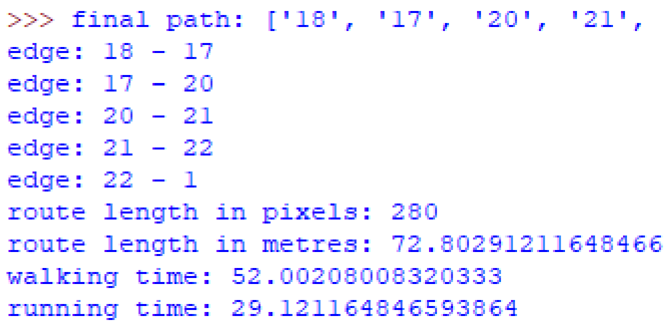


```
>>> final path: ['28', '25', '19', '18', '17', '14', '13', '11', '10']
edge: 28 - 25
edge: 25 - 19
edge: 19 - 18
edge: 18 - 17
edge: 17 - 14
edge: 14 - 13
edge: 13 - 11
route length in pixels: 505
route length in metres: 131.3052522100884
walking time: 93.78946586434887
running time: 52.52210088403536
```

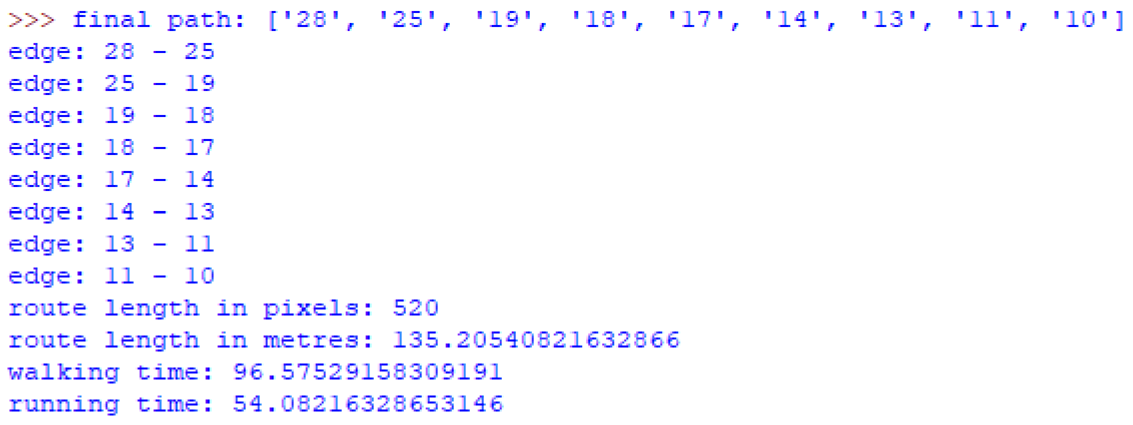


```
>>> final path: ['33', '34', '35', '36', '1', '2', '3', '4', '5']
edge: 33 - 34
edge: 34 - 35
edge: 35 - 36
edge: 36 - 1
edge: 1 - 2
edge: 2 - 3
edge: 3 - 4
route length in pixels: 560
route length in metres: 145.60582423296933
walking time: 104.00416016640666
running time: 58.24232969318773
```

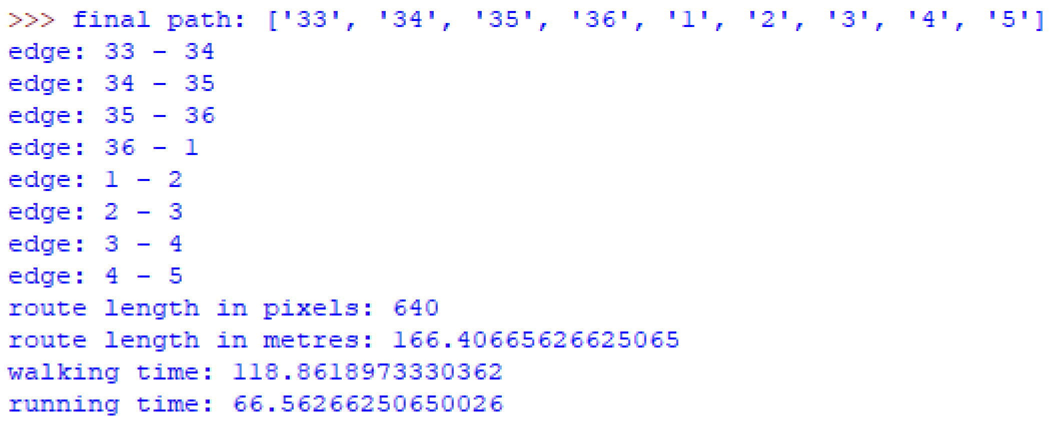
I fixed my program by making the loop that went through all the edges include the final edge and then tested again, below are the results showing it is working.



```
>>> final path: ['18', '17', '20', '21', '22', '1']
edge: 18 - 17
edge: 17 - 20
edge: 20 - 21
edge: 21 - 22
edge: 22 - 1
route length in pixels: 280
route length in metres: 72.80291211648466
walking time: 52.00208008320333
running time: 29.121164846593864
```



```
>>> final path: ['28', '25', '19', '18', '17', '14', '13', '11', '10']
edge: 28 - 25
edge: 25 - 19
edge: 19 - 18
edge: 18 - 17
edge: 17 - 14
edge: 14 - 13
edge: 13 - 11
edge: 11 - 10
route length in pixels: 520
route length in metres: 135.20540821632866
walking time: 96.57529158309191
running time: 54.08216328653146
```



```
>>> final path: ['33', '34', '35', '36', '1', '2', '3', '4', '5']
edge: 33 - 34
edge: 34 - 35
edge: 35 - 36
edge: 36 - 1
edge: 1 - 2
edge: 2 - 3
edge: 3 - 4
edge: 4 - 5
route length in pixels: 640
route length in metres: 166.40665626625065
walking time: 118.8618973330362
running time: 66.56266250650026
```

Data structures

Testing createRooms function

For this test I will use print statements to take interim values for each line in the file and check that the new line character (\n) is removed by adding an exclamation mark after the room name, if the new line character has been removed it will be on the same line.

Predicted room	Predicted new line check
AR1	AR1!
AR2	AR2!
AR3	AR3!
Business Studies	Business Studies!
Drama Studio	Drama Studio!
DT1	DT1!
DT2	DT2!
DT3	DT3!
DT4	DT4!
Food 1	Food 1!
Food 2	Food 2!
G1	G1!
G2	G2!
G3	G3!
G4	G4!
Harlequin Theatre	Harlequin Theatre!
IT1	IT1!
IT2	IT2!

IT5	IT5!
IT6	IT6!
IT7	IT7!
IT8	IT8!
IT9	IT9!
IT10	IT10!
Junior Hall	Junior Hall!
Lab 1	Lab 1!
Lab 1A	Lab 1A!
Lab 2	Lab 2!
Lab 3	Lab 3!
Lab 4	Lab 4!
Lab 4A	Lab 4A!
Lab 5	Lab 5!
Library	Library!
Main Theatre	Main Theatre!
MR1	MR1!
MR2	MR2!
MR3	MR3!
RE1	RE1!
RE2	RE2!
Recital Room	Recital Room!
Room 6	Room 6!
Room 7	Room 7!
Room 8	Room 8!
Room 9	Room 9!
Room 18	Room 18!
Room 19	Room 19!
Room 21	Room 21!
Room 22	Room 22!
Room 23	Room 23!
Room 24	Room 24!
Room 25	Room 25!
S1	S1!
S2	S2!
S3	S3!
S4	S4!
S5	S5!
S6	S6!
Senior Art	Senior Art!
Textiles 1	Textiles 1!
Textiles 2	Textiles 2!

Below is an image of the test. It only shows the first lines of my testing, however all the tests after this one showed that the new line character was not being removed.

```
room: AR1
 new line check: AR1
!
room: AR2
 new line check: AR2
!
room: AR3
 new line check: AR3
!
room: Business Studies
 new line check: Business Studies
!
room: Drama Studio
 new line check: Drama Studio
!
room: DT1
 new line check: DT1
!
room: DT2
 new line check: DT2
!
room: DT3
 new line check: DT3
!
room: DT4
 new line check: DT4
!
room: Food 1
 new line check: Food 1
!
room: Food 2
 new line check: Food 2
!
```

I fixed my program by using the inbuilt python strip function to remove the new line character from the line and store it in a new variable called room, before appending room to the rooms list. Below are the first lines of the test showing it is working.

```
room: AR1 new line check: AR1!
room: AR2 new line check: AR2!
room: AR3 new line check: AR3!
room: Business Studies new line check: Business Studies!
room: Drama Studio new line check: Drama Studio!
room: DT1 new line check: DT1!
room: DT2 new line check: DT2!
room: DT3 new line check: DT3!
room: DT4 new line check: DT4!
room: Food 1 new line check: Food 1!
room: Food 2 new line check: Food 2!
room: G1 new line check: G1!
room: G2 new line check: G2!
room: G3 new line check: G3!
room: G4 new line check: G4!
room: Harlequin Theatre new line check: Harlequin Theatre!
room: IT1 new line check: IT1!
room: IT2 new line check: IT2!
room: IT5 new line check: IT5!
room: IT6 new line check: IT6!
room: IT7 new line check: IT7!
room: IT8 new line check: IT8!
room: IT9 new line check: IT9!
room: IT10 new line check: IT10!
room: Junior Hall new line check: Junior Hall!
room: Lab 1 new line check: Lab 1!
room: Lab 1A new line check: Lab 1A!
room: Lab 2 new line check: Lab 2!
room: Lab 3 new line check: Lab 3!
room: Lab 4 new line check: Lab 4!
room: Lab 4A new line check: Lab 4A!
room: Lab 5 new line check: Lab 5!
room: Library new line check: Library!
room: Main Theatre new line check: Main Theatre!
room: MR1 new line check: MR1!
room: MR2 new line check: MR2!
room: MR3 new line check: MR3!
room: RE1 new line check: RE1!
room: RE2 new line check: RE2!
room: Recital Room new line check: Recital Room!
```

Testing createInstructions function

For this test I will use print statements to take interim values for each line in the file and then the instructions as they will be displayed on the GUI.

Predicted line	Predicted instructions
1) Select the room you are currently in	1) Select the room you are currently in 2) Select the room you want to go to 3) Press calculate route 4) The program will display a red line on the map which will show you the route you need to follow
2) Select the room you want to go to	
3) Press calculate route	
4) The program will display a red line on	
the map which will show you the route you	

need to follow	

Below is the image of my test, showing it is working.

```
line: 1) Select the room you are currently in

line: 2) Select the room you want to go to

line: 3) Press calculate route

line: 4) The program will display a red line on

line: the map which will show you the route you

line: need to follow
instructions: 1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on
the map which will show you the route you
need to follow
```

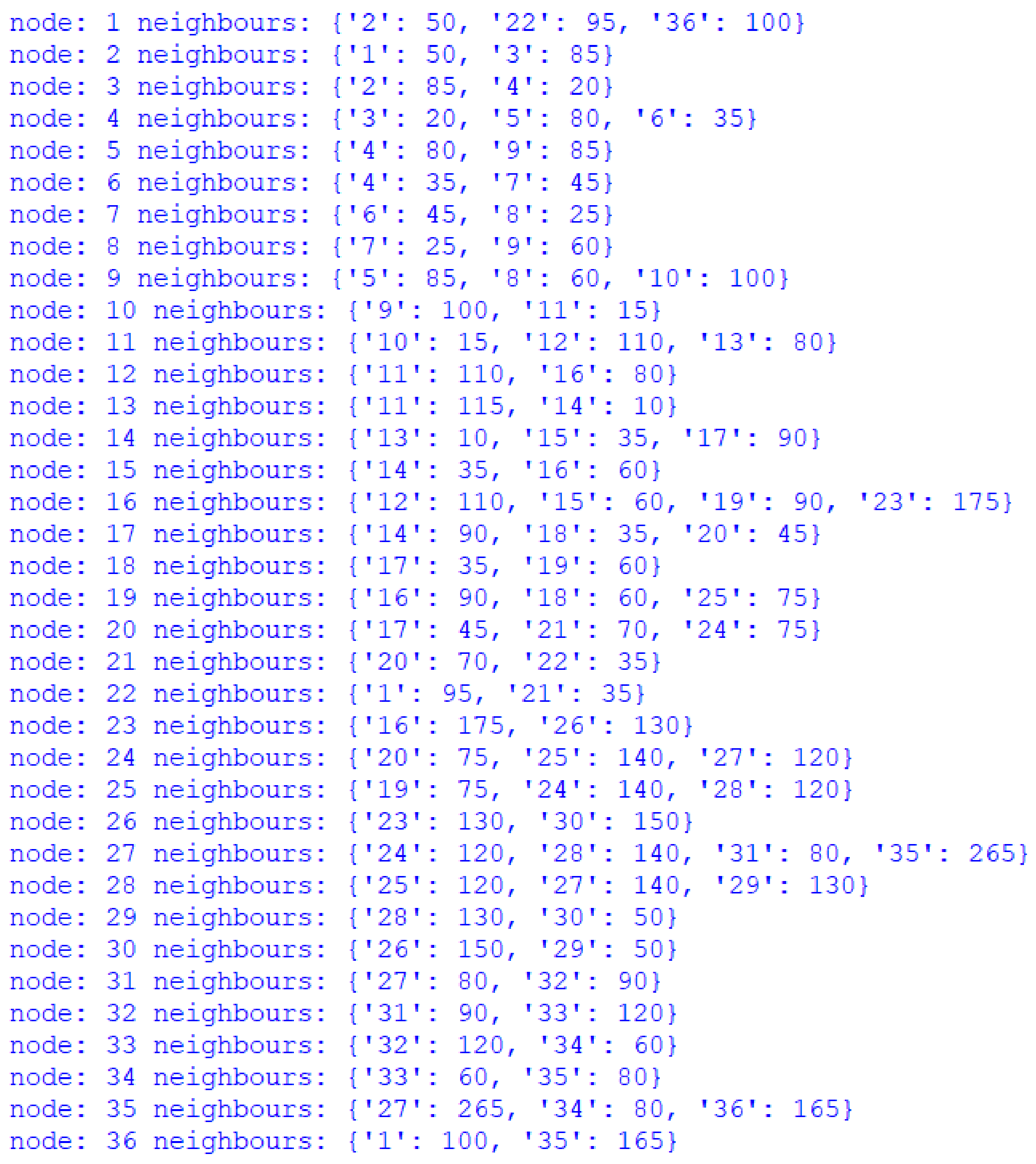
<u>Testing createGraph function</u>

For this test I will use print statements to get interim values for each node in the graph, their neighbours and the weights.

Predicted node	Predicted neighbours
1	2: 50, 22: 95, 36: 100
2	1: 50, 3: 85
3	2: 85, 4: 20
4	3: 20, 5: 80, 6: 35
5	4: 80, 9: 85
6	4: 35, 7: 45
7	6: 45, 8: 25
8	7: 25, 9: 60
9	5: 85, 8: 60, 10: 100
10	9: 100, 11: 15
11	10: 15, 12: 110, 13: 80
12	11: 110, 16: 80
13	11: 115, 14: 10
14	13: 10, 15: 35, 17: 90
15	14: 35, 16: 60
16	12: 110, 15: 60, 19: 90, 23: 175
17	14: 90, 18: 35, 20: 45
18	17: 35, 19: 60
19	16: 90, 18: 60, 25: 75
20	17: 45, 21: 70, 24: 75
21	20: 70, 22: 35
22	1: 95, 21: 35
23	16: 175, 26: 130
24	20: 75, 25: 140, 27: 120
25	19: 75, 24: 140, 28: 120
26	23: 130, 30: 150
27	24: 120, 28: 140, 31: 80, 35: 265

28	25: 120, 27: 140, 29: 130
29	28: 130, 30: 50
30	26: 150, 29: 50
31	27: 80, 32: 90
32	31: 90, 33: 120
33	32: 120, 34: 60
34	33: 60, 35: 80
35	27: 265, 34: 80, 36: 165
36	1: 100, 35: 165

Below is an image of the test showing it is working.



```
node: 1 neighbours: {'2': 50, '22': 95, '36': 100}
node: 2 neighbours: {'1': 50, '3': 85}
node: 3 neighbours: {'2': 85, '4': 20}
node: 4 neighbours: {'3': 20, '5': 80, '6': 35}
node: 5 neighbours: {'4': 80, '9': 85}
node: 6 neighbours: {'4': 35, '7': 45}
node: 7 neighbours: {'6': 45, '8': 25}
node: 8 neighbours: {'7': 25, '9': 60}
node: 9 neighbours: {'5': 85, '8': 60, '10': 100}
node: 10 neighbours: {'9': 100, '11': 15}
node: 11 neighbours: {'10': 15, '12': 110, '13': 80}
node: 12 neighbours: {'11': 110, '16': 80}
node: 13 neighbours: {'11': 115, '14': 10}
node: 14 neighbours: {'13': 10, '15': 35, '17': 90}
node: 15 neighbours: {'14': 35, '16': 60}
node: 16 neighbours: {'12': 110, '15': 60, '19': 90, '23': 175}
node: 17 neighbours: {'14': 90, '18': 35, '20': 45}
node: 18 neighbours: {'17': 35, '19': 60}
node: 19 neighbours: {'16': 90, '18': 60, '25': 75}
node: 20 neighbours: {'17': 45, '21': 70, '24': 75}
node: 21 neighbours: {'20': 70, '22': 35}
node: 22 neighbours: {'1': 95, '21': 35}
node: 23 neighbours: {'16': 175, '26': 130}
node: 24 neighbours: {'20': 75, '25': 140, '27': 120}
node: 25 neighbours: {'19': 75, '24': 140, '28': 120}
node: 26 neighbours: {'23': 130, '30': 150}
node: 27 neighbours: {'24': 120, '28': 140, '31': 80, '35': 265}
node: 28 neighbours: {'25': 120, '27': 140, '29': 130}
node: 29 neighbours: {'28': 130, '30': 50}
node: 30 neighbours: {'26': 150, '29': 50}
node: 31 neighbours: {'27': 80, '32': 90}
node: 32 neighbours: {'31': 90, '33': 120}
node: 33 neighbours: {'32': 120, '34': 60}
node: 34 neighbours: {'33': 60, '35': 80}
node: 35 neighbours: {'27': 265, '34': 80, '36': 165}
node: 36 neighbours: {'1': 100, '35': 165}
```

Testing createCoordinates function

For this test I will use interim print statements to get each node and its coordinates.

Predicted node	Predicted x coordinate	Predicted y coordinate
1	105	284
2	105	235
3	105	153
4	125	153

5	125	68
6	160	153
7	204	153
8	204	125
9	204	68
10	302	68
11	302	85
12	415	85
13	302	160
14	315	160
15	350	160
16	415	160
17	315	250
18	350	250
19	415	250
20	270	250
21	203	250
22	203	284
23	590	160
24	270	315
25	415	315
26	590	290
27	270	445
28	415	445
29	540	445
30	590	445
31	270	525
32	190	525
33	70	525
34	5	525
35	5	445
36	5	284

Below is an image of my test showing it is working.

```
node: 1 x coordinate: 105 y coordinate: 284
node: 2 x coordinate: 105 y coordinate: 235
node: 3 x coordinate: 105 y coordinate: 153
node: 4 x coordinate: 125 y coordinate: 153
node: 5 x coordinate: 125 y coordinate: 68
node: 6 x coordinate: 160 y coordinate: 153
node: 7 x coordinate: 204 y coordinate: 153
node: 8 x coordinate: 204 y coordinate: 125
node: 9 x coordinate: 204 y coordinate: 68
node: 10 x coordinate: 302 y coordinate: 68
node: 11 x coordinate: 302 y coordinate: 85
node: 12 x coordinate: 415 y coordinate: 85
node: 13 x coordinate: 302 y coordinate: 160
node: 14 x coordinate: 315 y coordinate: 160
node: 15 x coordinate: 350 y coordinate: 160
node: 16 x coordinate: 415 y coordinate: 160
node: 17 x coordinate: 315 y coordinate: 250
node: 18 x coordinate: 350 y coordinate: 250
node: 19 x coordinate: 415 y coordinate: 250
node: 20 x coordinate: 270 y coordinate: 250
node: 21 x coordinate: 203 y coordinate: 250
node: 22 x coordinate: 203 y coordinate: 284
node: 23 x coordinate: 590 y coordinate: 160
node: 24 x coordinate: 270 y coordinate: 315
node: 25 x coordinate: 415 y coordinate: 315
node: 26 x coordinate: 590 y coordinate: 290
node: 27 x coordinate: 270 y coordinate: 445
node: 28 x coordinate: 415 y coordinate: 445
node: 29 x coordinate: 540 y coordinate: 445
node: 30 x coordinate: 590 y coordinate: 445
node: 31 x coordinate: 270 y coordinate: 525
node: 32 x coordinate: 190 y coordinate: 525
node: 33 x coordinate: 70 y coordinate: 525
node: 34 x coordinate: 5 y coordinate: 525
node: 35 x coordinate: 5 y coordinate: 445
node: 36 x coordinate: 5 y coordinate: 284
```

Testing createNodes function

For this test I will use print statements to get the values for each node in the dictionary and the rooms linked to each node.

Predicted node	Predicted rooms
1	Main Theatre
2	Junior Hall, Room 6, Room 7, Room 8, Room 9
3	IT5, Textiles 1, Textiles 2
5	DT3, DT4, IT9
6	DT1, DT2, Food 1, Food 2
8	IT10
10	Lab 4, Lab 4A, Lab 5
12	Lab 1, Lab 1A, Lab 2, Lab 3
15	Room 21, Room 22, Room 23
18	Room 18, Room 19, Room 24, Room 25
21	Library
22	IT6
26	Business Studies, IT1, IT2, S1, S2, S3, S4, S5, S6

28	Harlequin Theatre, Recital Room, MR1, MR2, MR3
29	Drama Studio, Senior Art, AR1, AR2, AR3
32	IT7, IT8, RE1, RE2
33	G1, G2, G3, G4

Below is the test showing it is working.

```
node: 1 rooms: ['Main Theatre']
node: 2 rooms: ['Junior Hall', 'Room 6', 'Room 7', 'Room 8', 'Room 9']
node: 3 rooms: ['IT5', 'Textiles 1', 'Textiles 2']
node: 5 rooms: ['DT3', 'DT4', 'IT9']
node: 6 rooms: ['DT1', 'DT2', 'Food 1', 'Food 2']
node: 8 rooms: ['IT10']
node: 10 rooms: ['Lab 4', 'Lab 4A', 'Lab 5']
node: 12 rooms: ['Lab 1', 'Lab 1A', 'Lab 2', 'Lab 3']
node: 15 rooms: ['Room 21', 'Room 22', 'Room 23']
node: 18 rooms: ['Room 18', 'Room 19', 'Room 24', 'Room 25']
node: 21 rooms: ['Library']
node: 22 rooms: ['IT6']
node: 26 rooms: ['Business Studies', 'IT1', 'IT2', 'S1', 'S2', 'S3', 'S4', 'S5', 'S6']
node: 28 rooms: ['Harlequin Theatre', 'Recital Room', 'MR1', 'MR2', 'MR3']
node: 29 rooms: ['Drama Studio', 'Senior Art', 'AR1', 'AR2', 'AR3']
node: 32 rooms: ['IT7', 'IT8', 'RE1', 'RE2']
node: 33 rooms: ['G1', 'G2', 'G3', 'G4']
```

Testing what happens when file is missing

For this test I will see what happens when I run the program with a file missing. The file I am choosing is the one containing the graph. I predict that there will be an error because I haven't added any error handling for this part yet.

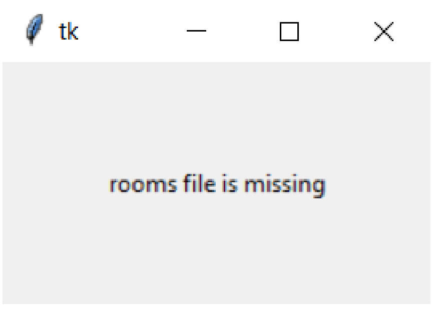
Below is the image showing the error, matching my predictions.

```
Traceback (most recent call last):
  File "\\webdav.queenelizabeth.cumbria.sch.uk@SSL\DavWWWRoot\2012\12finlay.mcco
rmick\Sixth Form\Computing\Year 13\NEA\Program for removing file\Main.py", line
90, in <module>
    with open('graph.txt', 'r') as file:
FileNotFoundError: [Errno 2] No such file or directory: 'graph.txt'
```

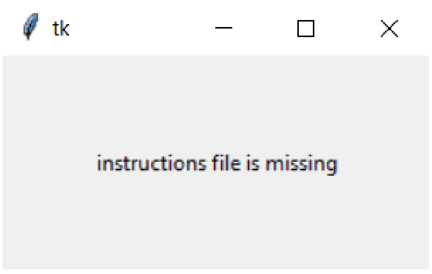
To fix this I added try and except statements to my program, with the code importing the file in the try and a call to a new subroutine in the except. This subroutine is an error handling procedure called displayError, with the file name as the parameter, that displays a window telling the user which file is missing and stops the program.

Missing file	Predicted error message
rooms.txt	rooms file is missing
instructions.txt	instructions file is missing
graph.txt	graph file is missing
coordinates.txt	coordinates file is missing
nodes.txt	nodes file is missing

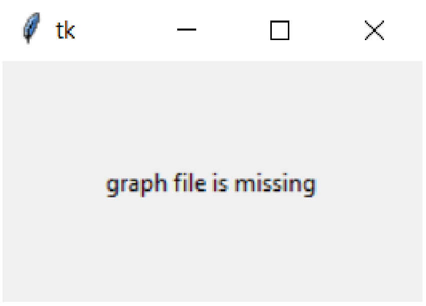
Below are images of the error message displayed in the window.



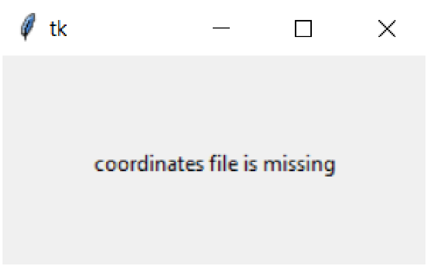
rooms file is missing



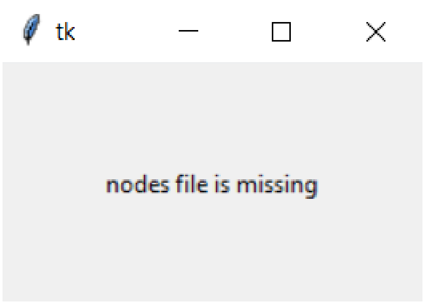
instructions file is missing



graph file is missing



coordinates file is missing

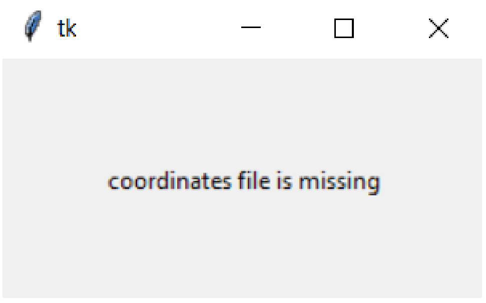


nodes file is missing

Testing what happens when file is corrupted

For this test I will see what happens if any of the data files are either missing data or using the wrong format. I expect that any errors will be caught by the error handling I added for the previous test. I decided to start by replacing one of the commas with a full stop in the coordinates file.

Below is the image of the window, showing I was correct.


tk

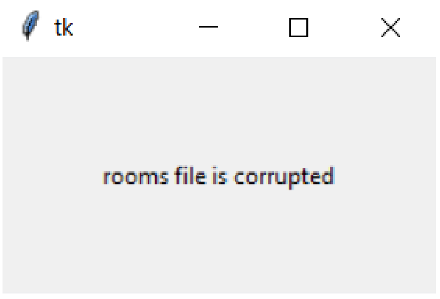
```
coordinates file is missing
```

Whilst it is doing what I expect, the error message is incorrect for the problem being caused, so I added another parameter to displayError which allowed it to deal with both missing and corrupted files. I also added try and except statements to each of my data structure creating functions, with the code in the try and a call to displayError in the except. To cause these errors I will remove all text from the files.

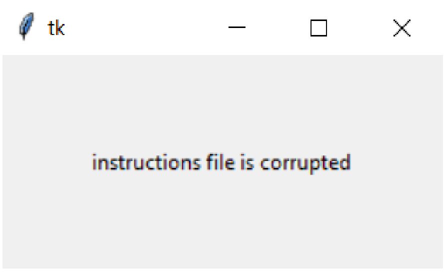
Corrupted file	Predicted error message
rooms.txt	rooms file is corrupted
instructions.txt	instructions file is corrupted
graph.txt	graph file is corrupted
coordinates.txt	coordinates file is corrupted
nodes.txt	nodes file is corrupted

However, due to how my code works, this type of error is not picked up, so I added additional code to ensure that these errors would be picked up. This code added a Boolean variable called dataPresent that is initialised as false and will only become true if there is text on a line in the file. If dataPresent is not true then the displayError procedure will be called.

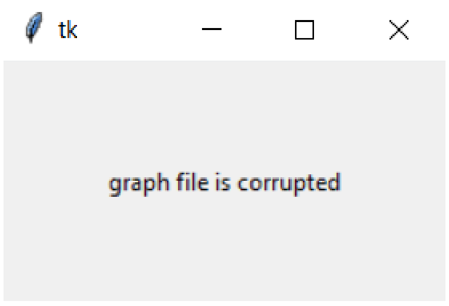
Below are images of the windows showing my program is working.

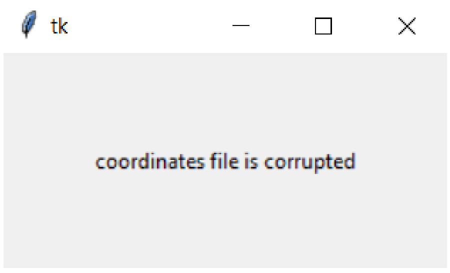

tk

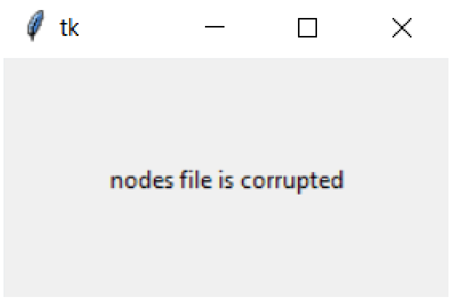
```
rooms file is corrupted
```


tk

```
instructions file is corrupted
```


```
tk
graph file is corrupted
```

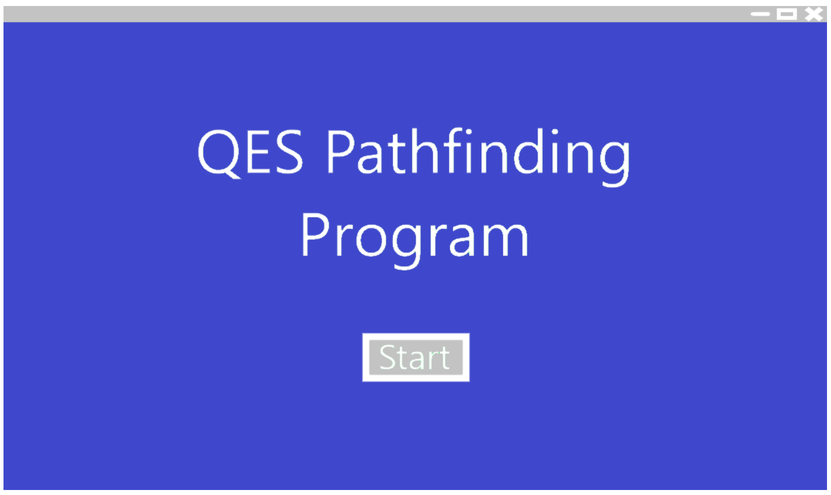

```
tk
coordinates file is corrupted
```


```
tk
nodes file is corrupted
```

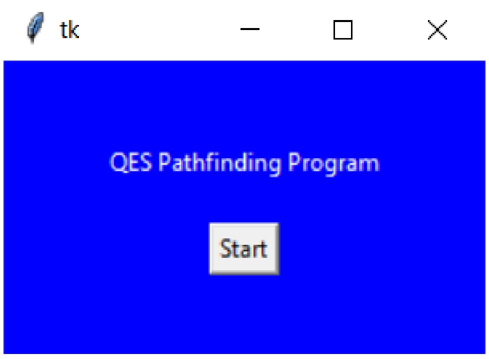
GUI

Testing the start screen

For this test I will compare the start screen generated by my GUI to the one I designed in my design section. Below is the picture of how I expect my start screen to look.

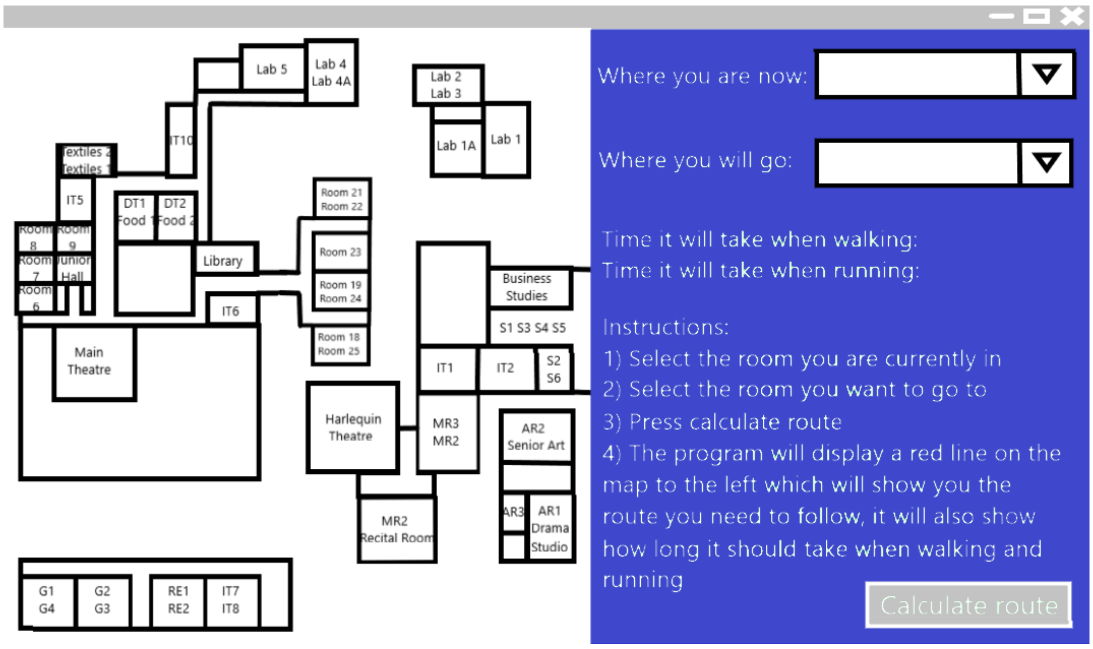


Below is the picture of the start screen generated by the GUI, whilst they are not identical, for example the title is only one line, it is practically the same so satisfies me.



Testing the main screen

For this test I will compare the main screen generated by my GUI to the one I designed in my design section. Below is the picture of how I expect my main screen to look.



Where you are now:

Where you will go:

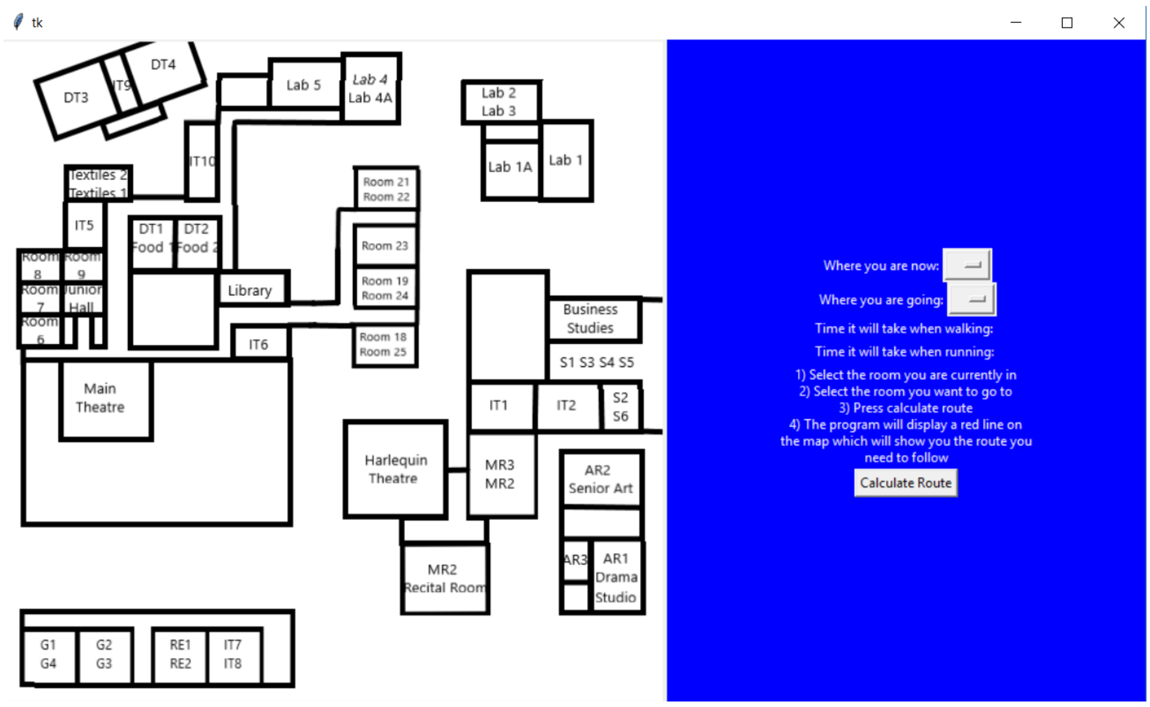
Time it will take when walking:

Time it will take when running:

Instructions:

1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map to the left which will show you the route you need to follow, it will also show how long it should take when walking and running

Similarly to with the start screen, there are some minor differences from the main screen I designed, however they are similar enough.



Where you are now:

Where you are going:

Time it will take when walking:

Time it will take when running:

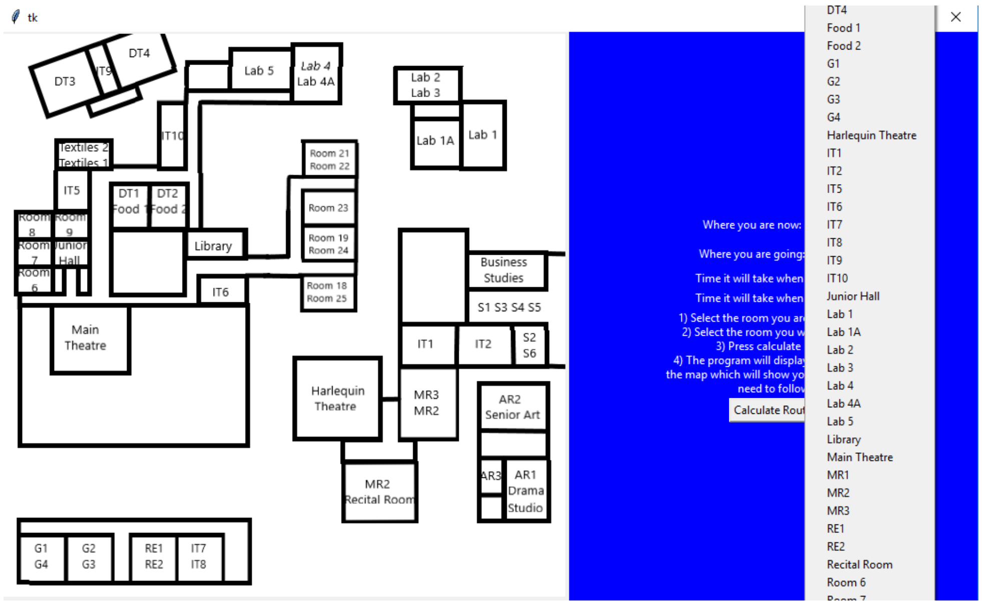
1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow

Calculate Route

Testing the dropdown lists

For this test I will take screenshots of my GUI showing the rooms being displayed in the dropdown lists and the selected room being displayed in the dropdown list boxes.

Below are the images of the rooms being shown in the dropdown lists.



Where you are now:

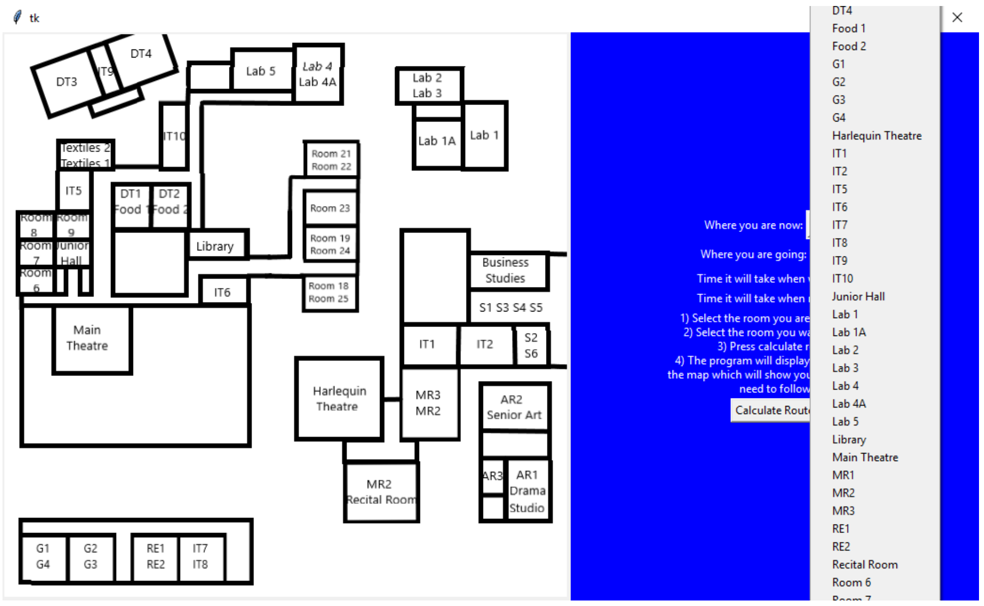
Where you are going:

Time it will take when

Time it will take when

1) Select the room you ar
2) Select the room you w
3) Press calculate
4) The program will displa
the map which will show yo
need to follo

Calculate Rou



Where you are now:

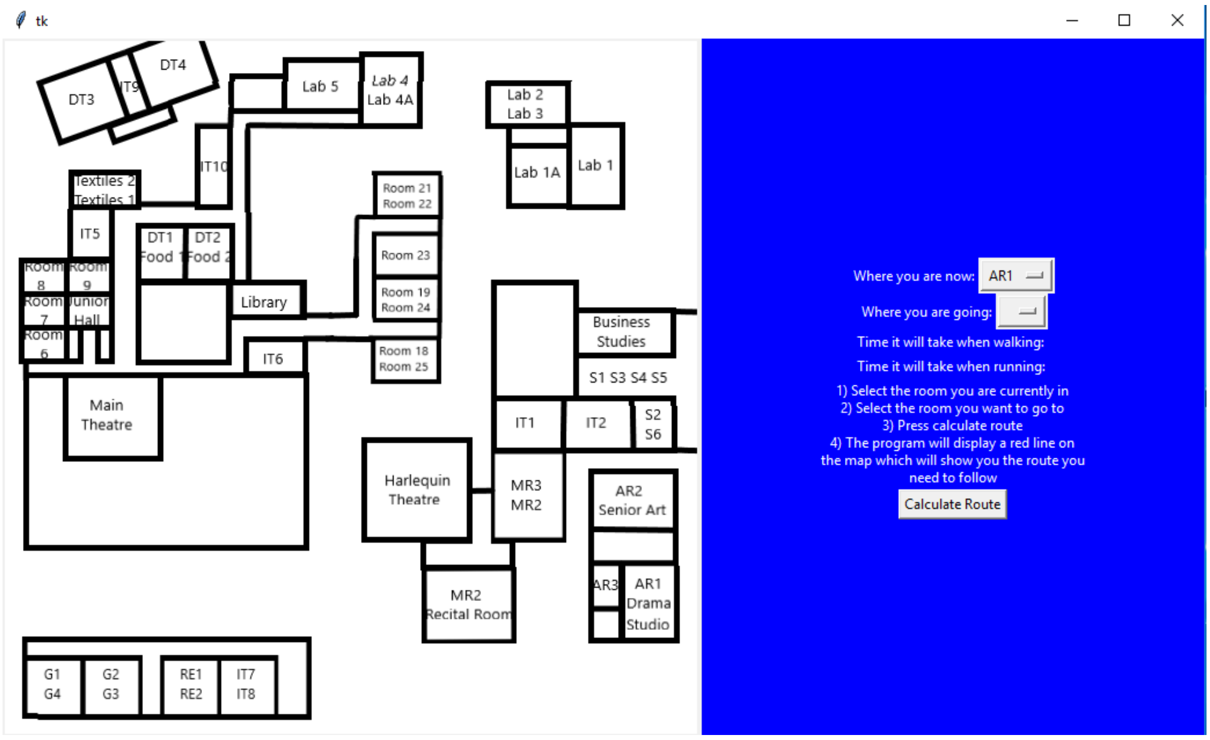
Where you are going:

Time it will take when

Time it will take when

1) Select the room you are
2) Select the room you wa
3) Press calculate r
4) The program will display
the map which will show you
need to follow

Below are images of the GUI with selected options being displayed in the dropdown list boxes.



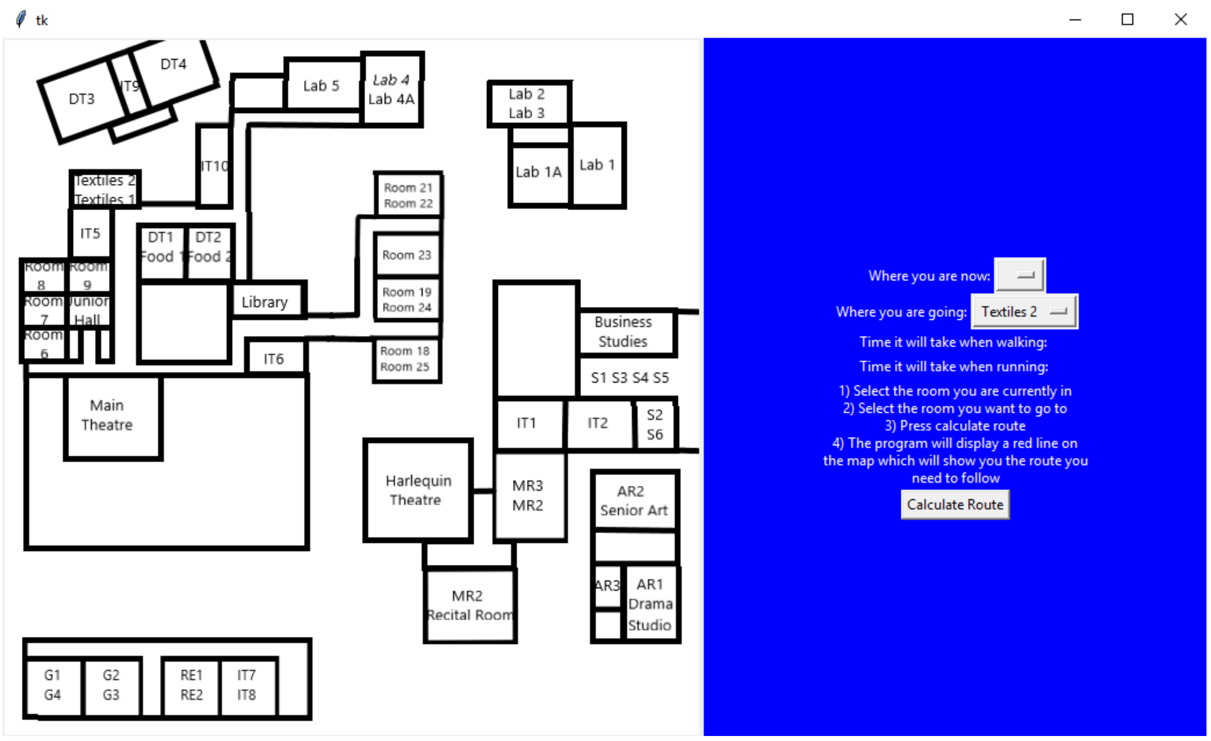
Where you are now: AR1

Where you are going:

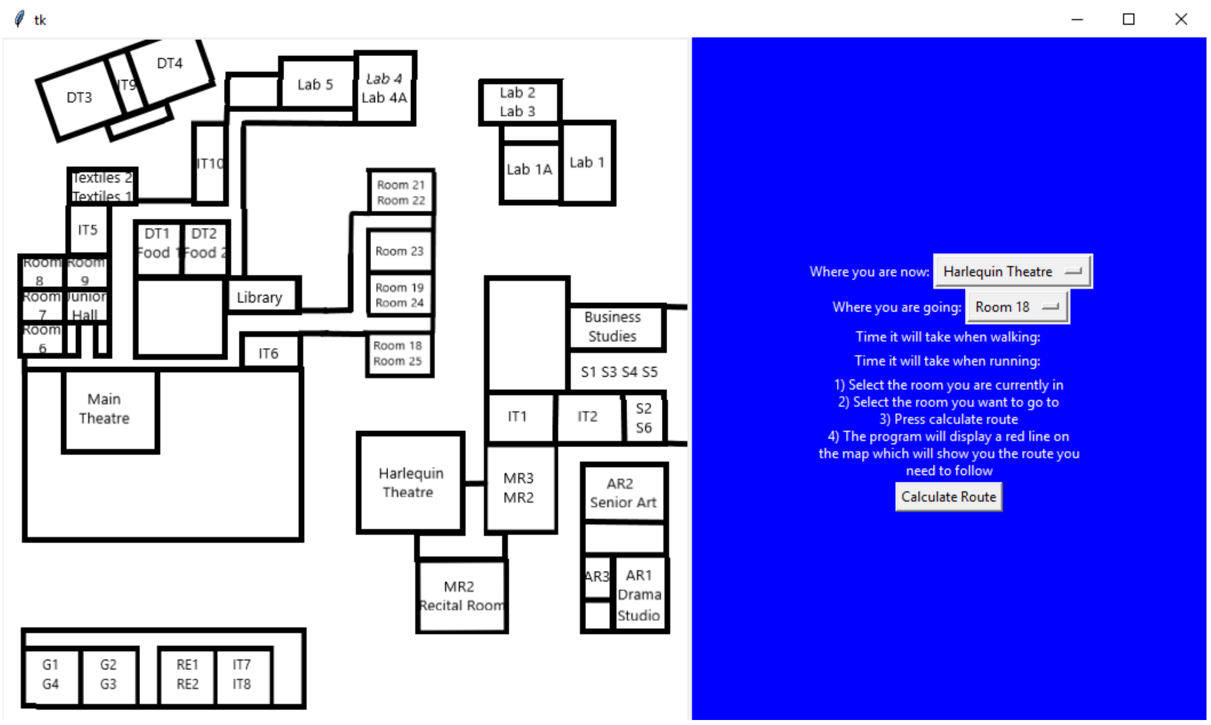
Time it will take when walking:

Time it will take when running:

1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on
the map which will show you the route you
need to follow



Where you are now:
Where you are going: Textiles 2
Time it will take when walking:
Time it will take when running:
1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow



Where you are now: Harlequin Theatre
Where you are going: Room 18
Time it will take when walking:
Time it will take when running:
1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow

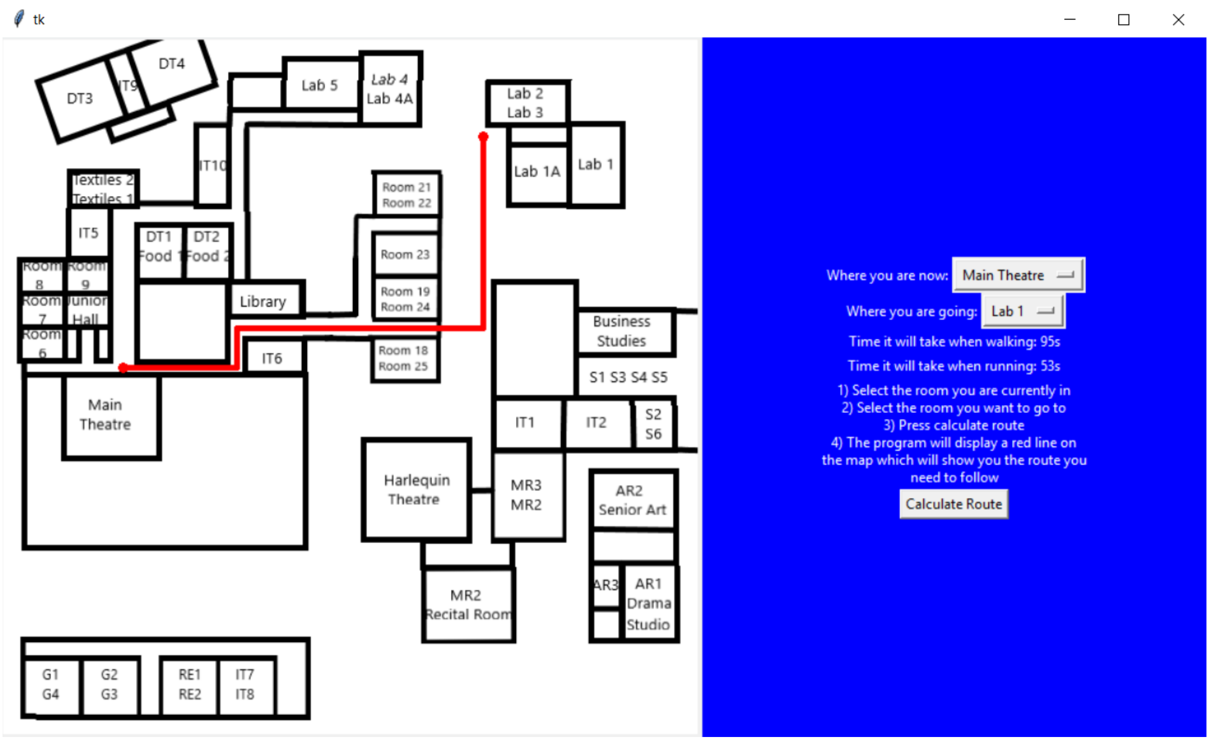
Testing the calculate route button

For this test I will enter values into the dropdown boxes in my GUI and press the calculate route button. I will not do any tests with boundary data since there is no boundary data, only normal and erroneous.

Inputs		Type	Result
Main Theatre	Lab 1	Normal	Quickest route displayed as red line between Main Theatre and Lab 1 and times displayed

G1	IT10	Normal	Quickest route displayed as red line between G1 and IT10 and times displayed
AR1	Textiles 2	Normal	Quickest route displayed as red line between AR1 and Textiles 2 and times displayed
MR1	MR1	Erroneous	Nothing should happen because they are the same room
Room 18	Room 19	Erroneous	Nothing should happen because they are on the same node

Below are the images of my tests, whilst the first four do as I expected, the fifth one causes a red dot to appear and the times to appear as zero seconds.



Where you are now: Main Theatre

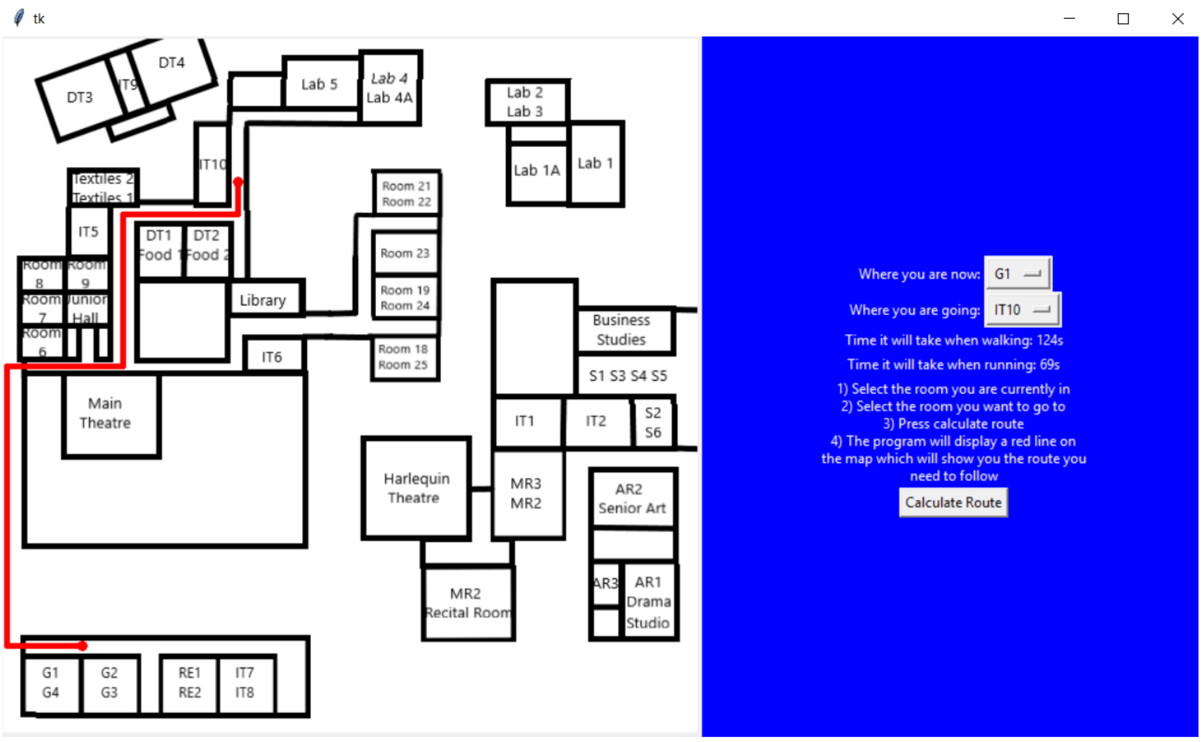
Where you are going: Lab 1

Time it will take when walking: 95s

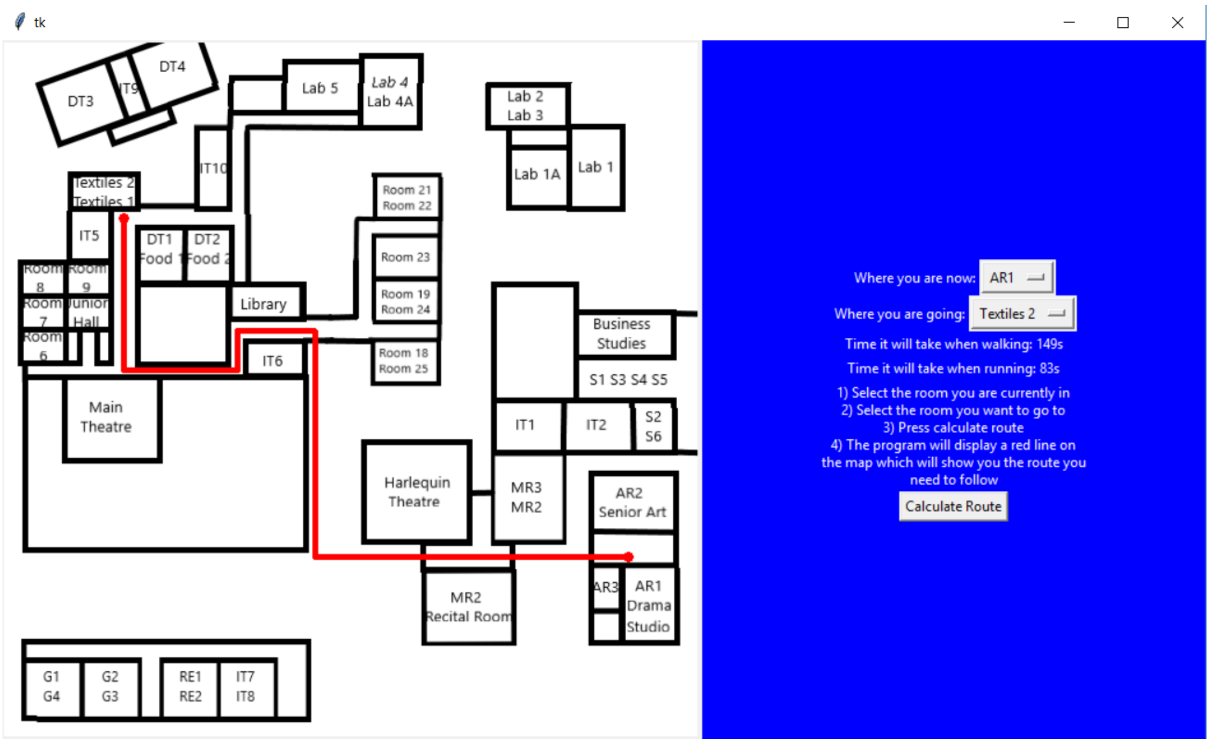
Time it will take when running: 53s

1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow

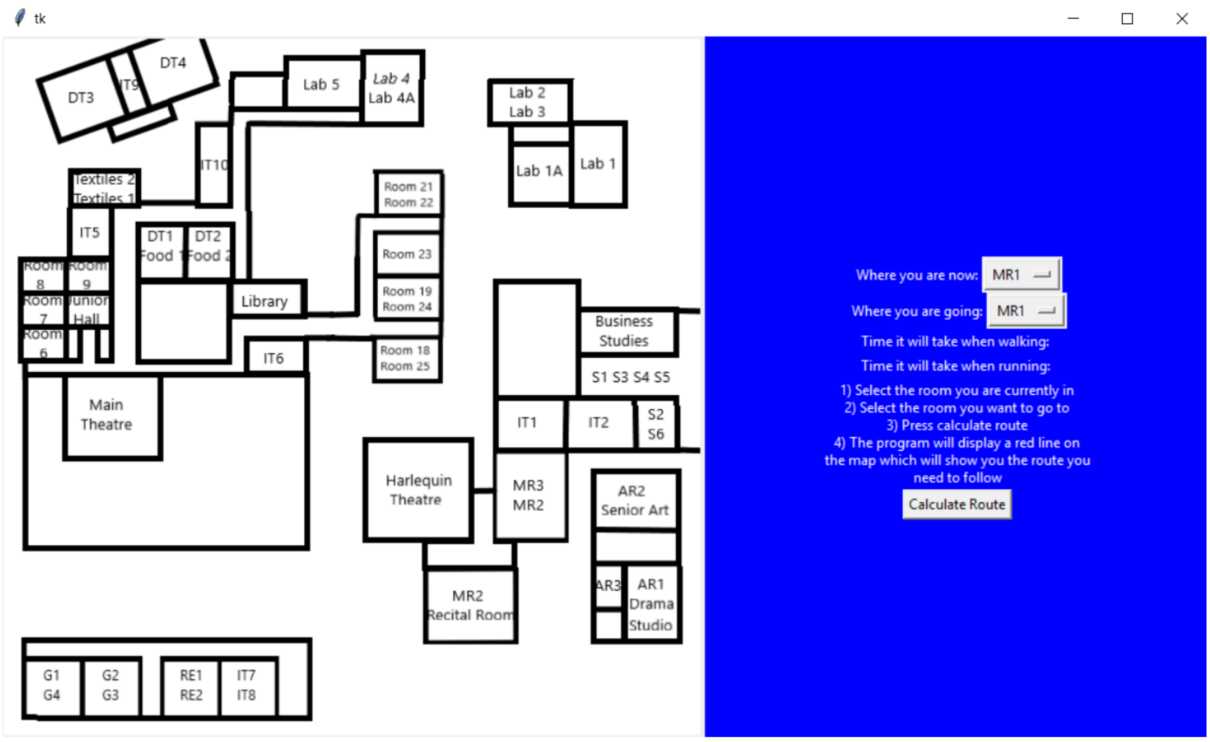
Calculate Route



Where you are now: G1
Where you are going: IT10
Time it will take when walking: 124s
Time it will take when running: 69s
1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow



Where you are now: AR1
Where you are going: Textiles 2
Time it will take when walking: 149s
Time it will take when running: 83s
1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow



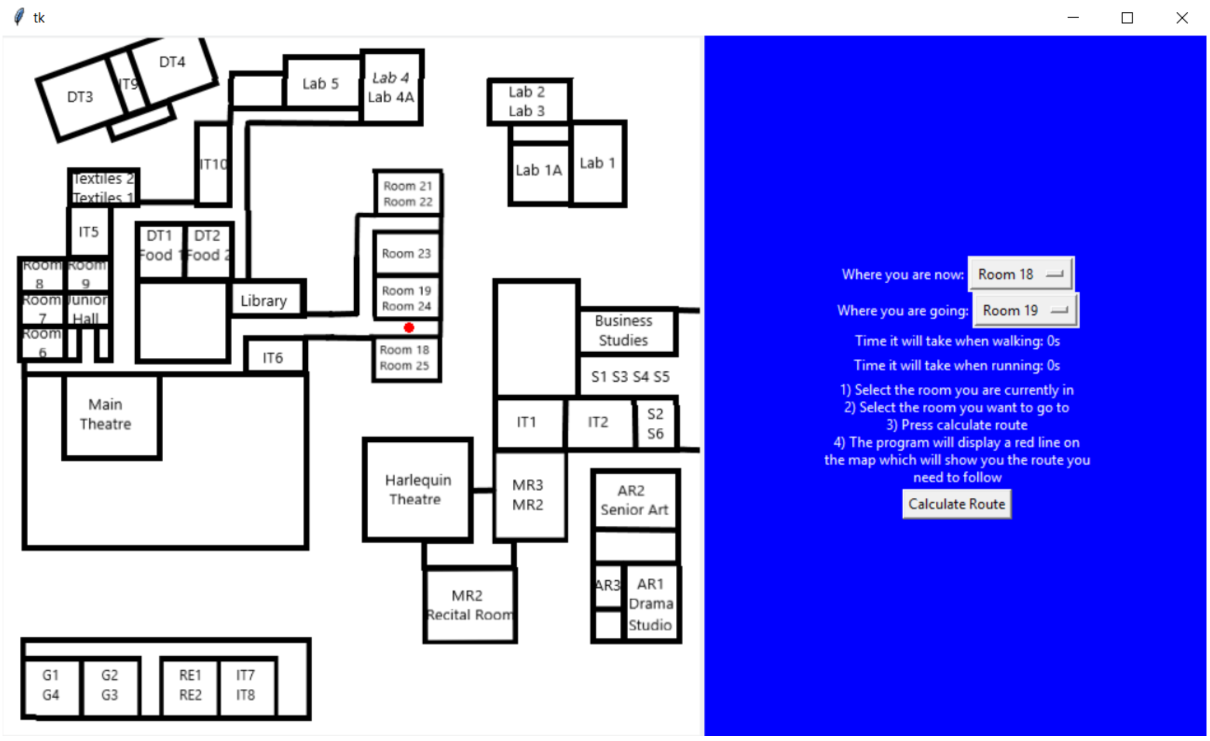
Where you are now: MR1

Where you are going: MR1

Time it will take when walking:

Time it will take when running:

1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow



Where you are now: Room 18

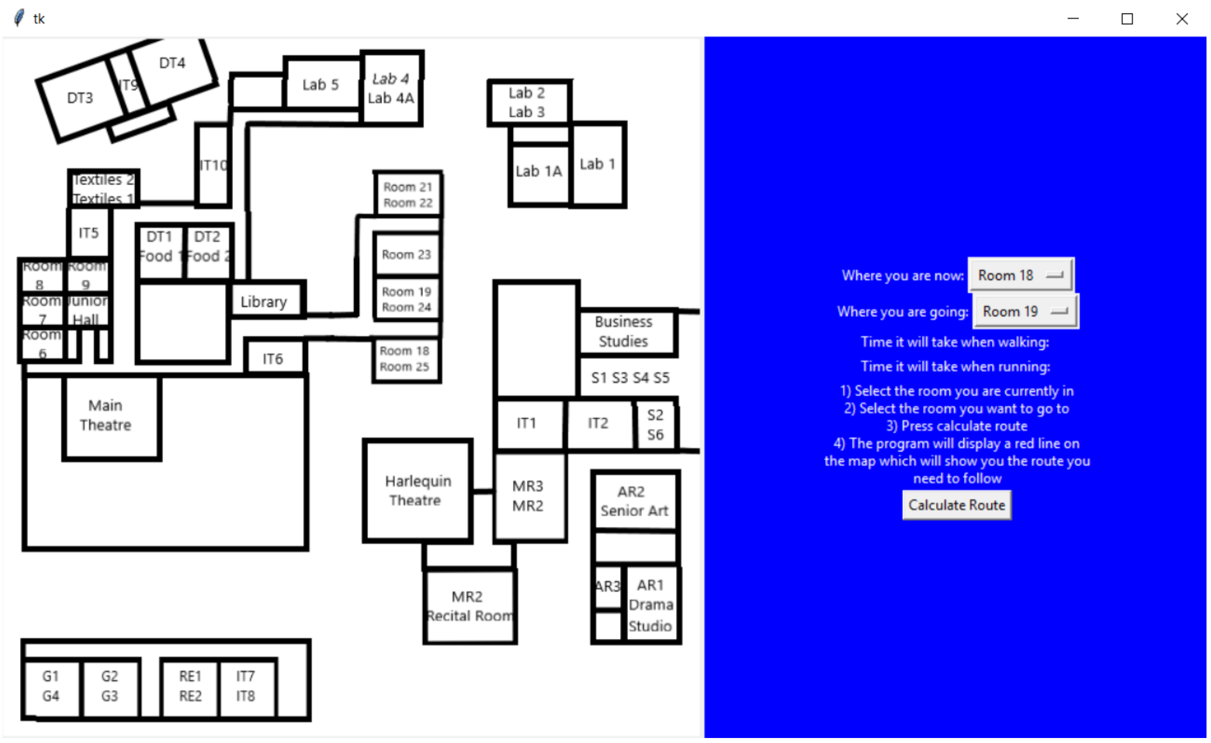
Where you are going: Room 19

Time it will take when walking: 0s

Time it will take when running: 0s

1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow

To fix this bug I added another if statement to the runAStarAlgorithm procedure in the interface class which meant that the main code would not run if startNode and destinationNode were not the same.

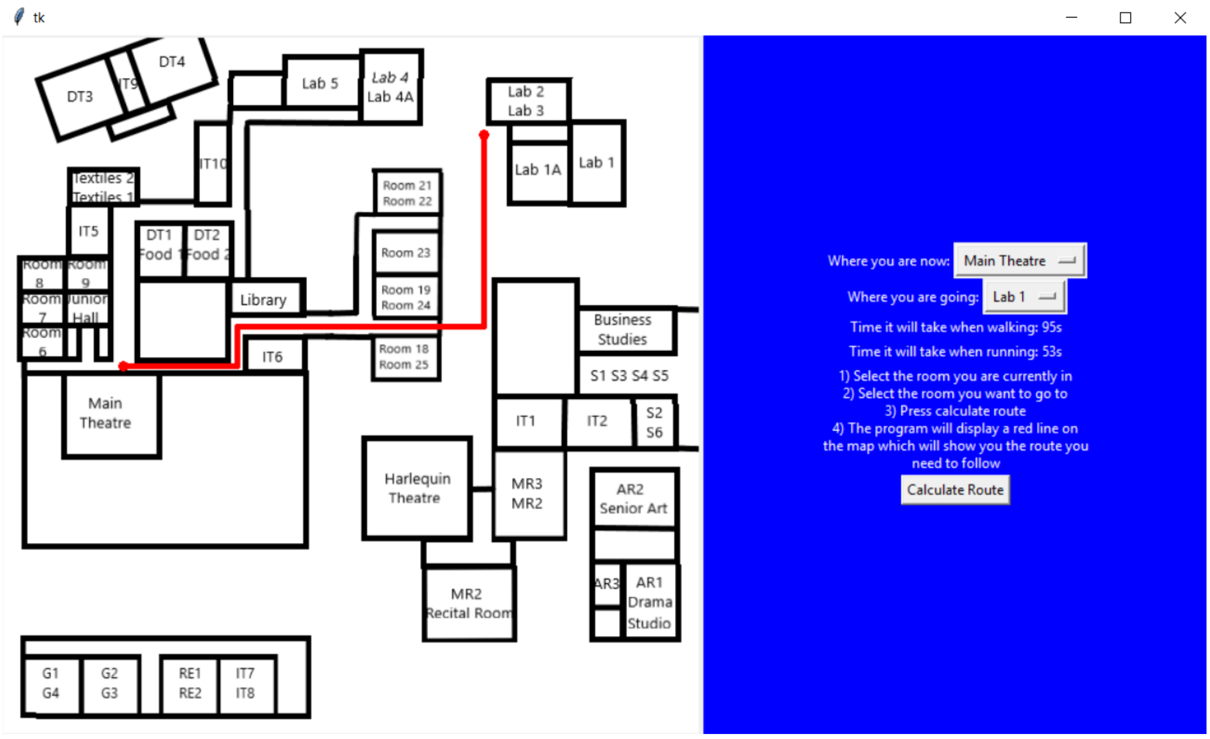


Where you are now: Room 18
Where you are going: Room 19
Time it will take when walking:
Time it will take when running:
1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow
Calculate Route

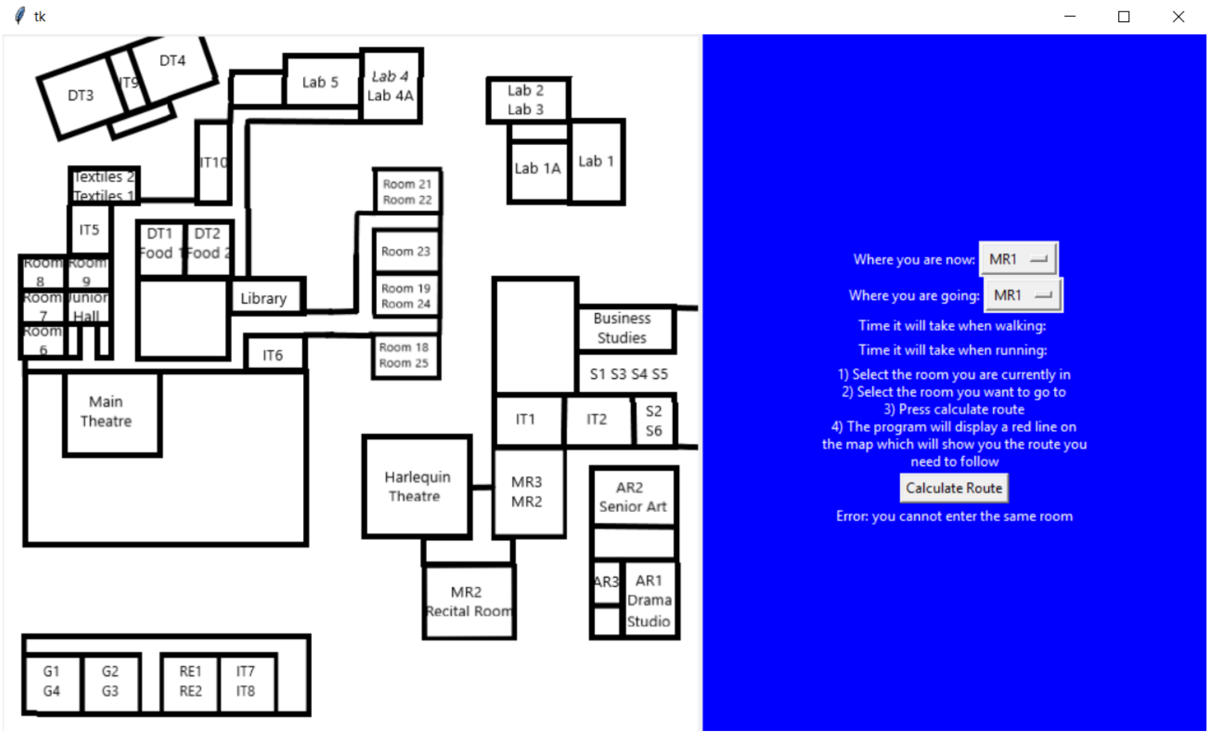
However, whilst the bug was fixed the program was still not very descriptive since it did not tell the user why it was not running when the same room or node was selected, so I added a label below the calculate route button which would display error messages depending on the situation. I also made it so that when a new option in the dropdown list is selected the error message is reset.

Inputs		Predicted message
Main Theatre	Lab 1	*nothing*
MR1	MR1	Error: you cannot enter the same room
Room 18	Room 19	Error: these rooms are too close together

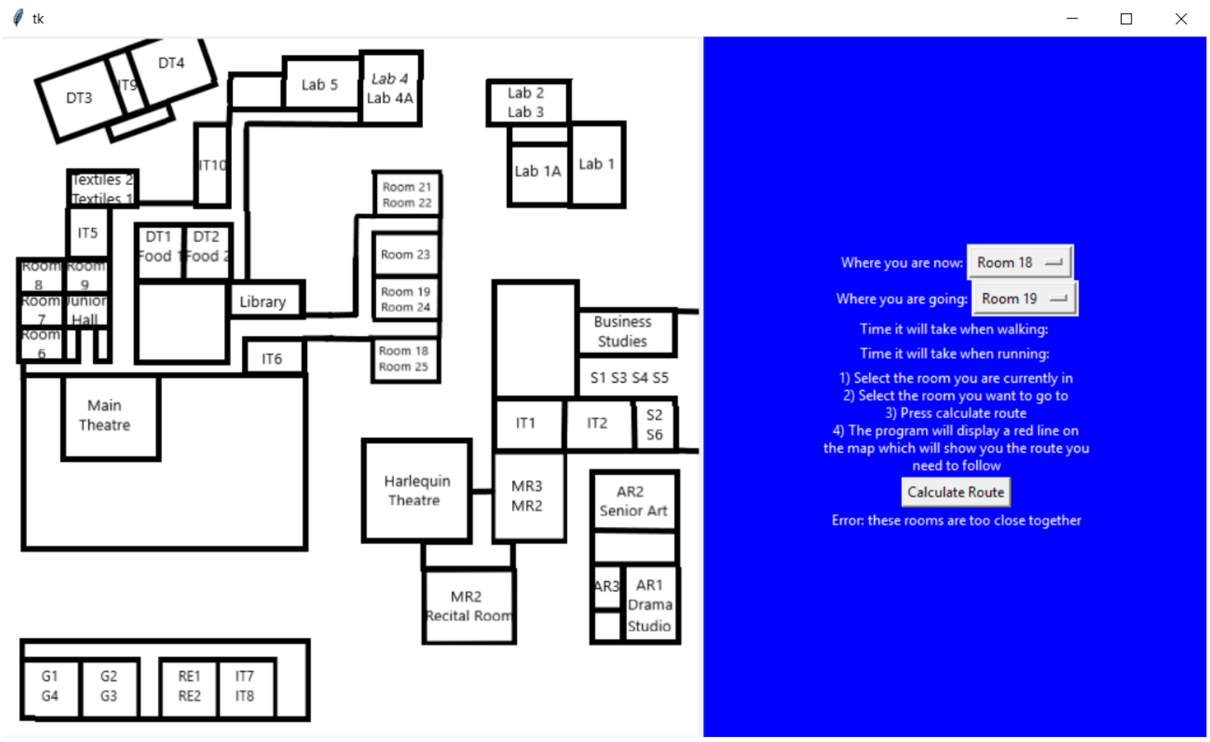
Below are images of the error messages being properly displayed.



Where you are now: Main Theatre
Where you are going: Lab 1
Time it will take when walking: 95s
Time it will take when running: 53s
1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow
Calculate Route



Where you are now: MR1
Where you are going: MR1
Time it will take when walking:
Time it will take when running:
1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow
Calculate Route
Error: you cannot enter the same room



Where you are now: Room 18

Where you are going: Room 19

Time it will take when walking:

Time it will take when running:

1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow

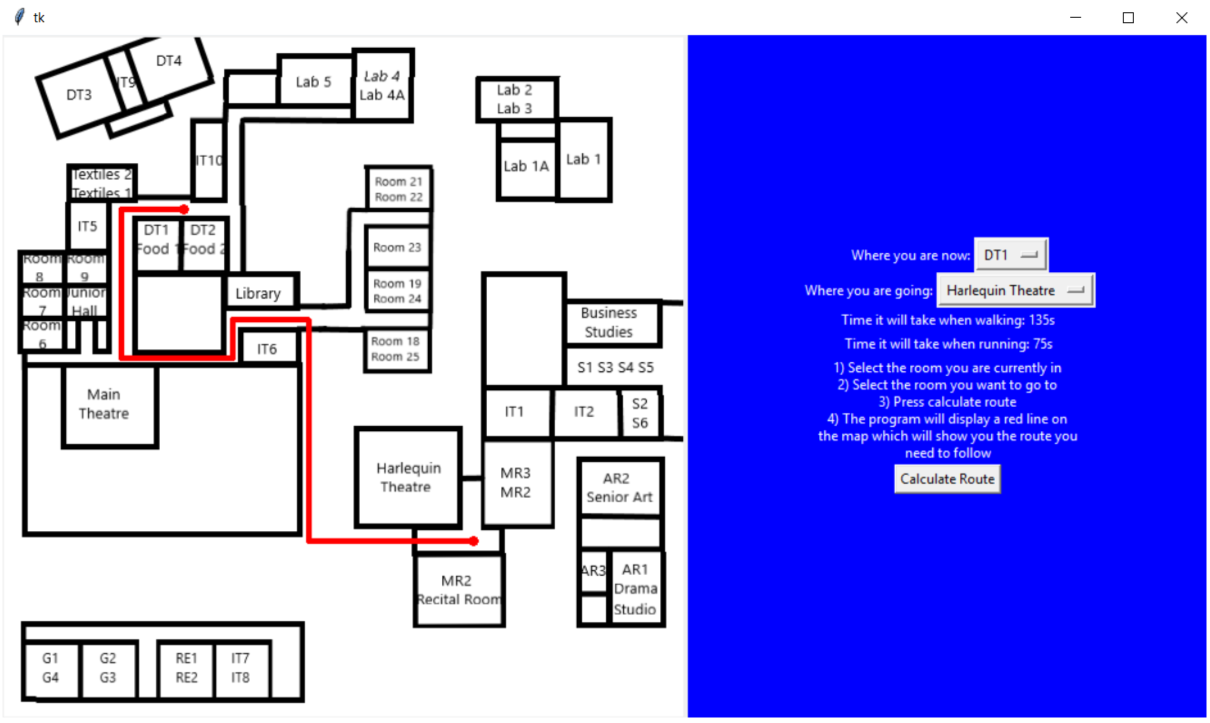
Calculate Route

Error: these rooms are too close together

Testing route is cleared when new option selected

For this test I will enter values into the dropdown boxes, calculate the route then enter a new value which should make the route disappear off the canvas and the strings showing the times to reset.

Below are images showing that it is working.



Where you are now: DT1

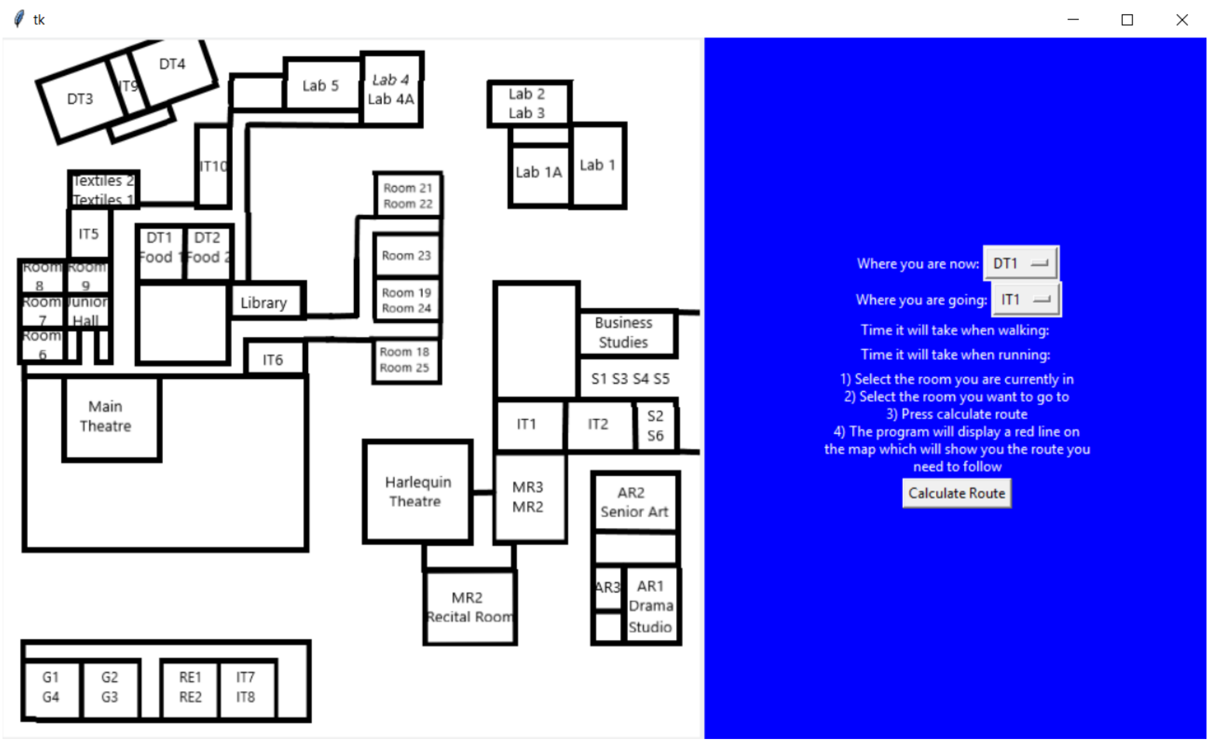
Where you are going: Harlequin Theatre

Time it will take when walking: 135s

Time it will take when running: 75s

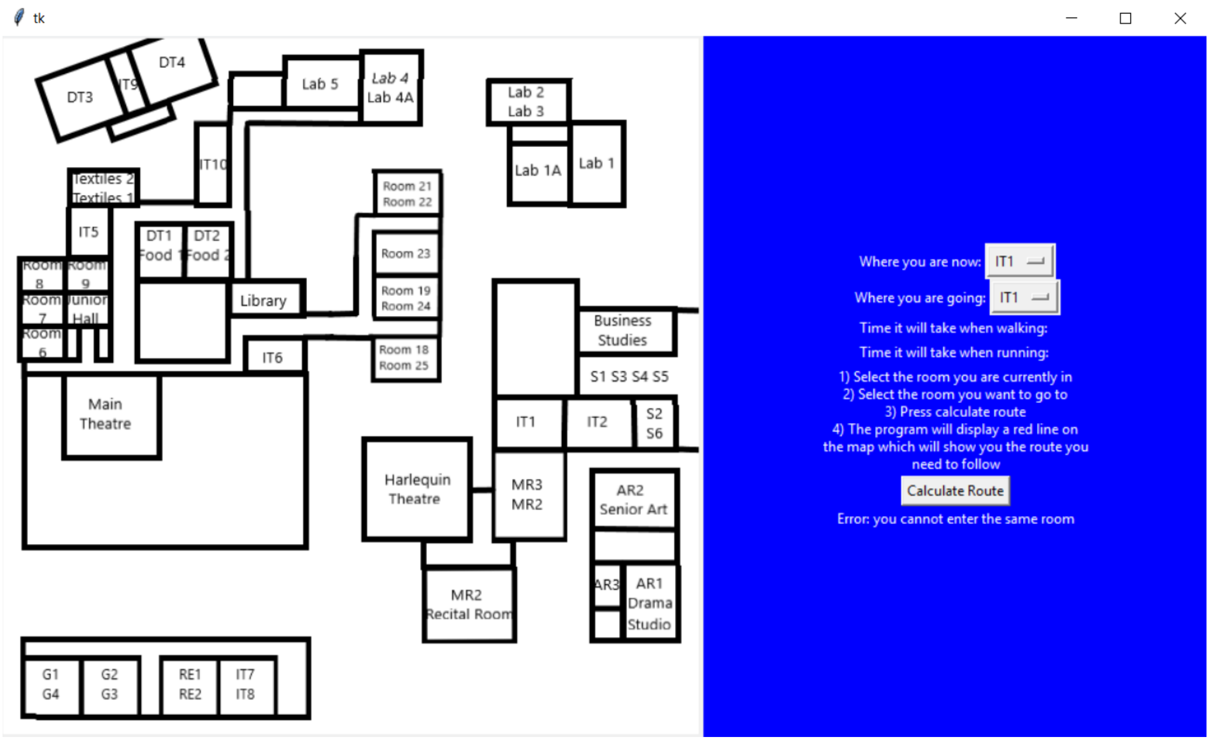
1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow

Calculate Route



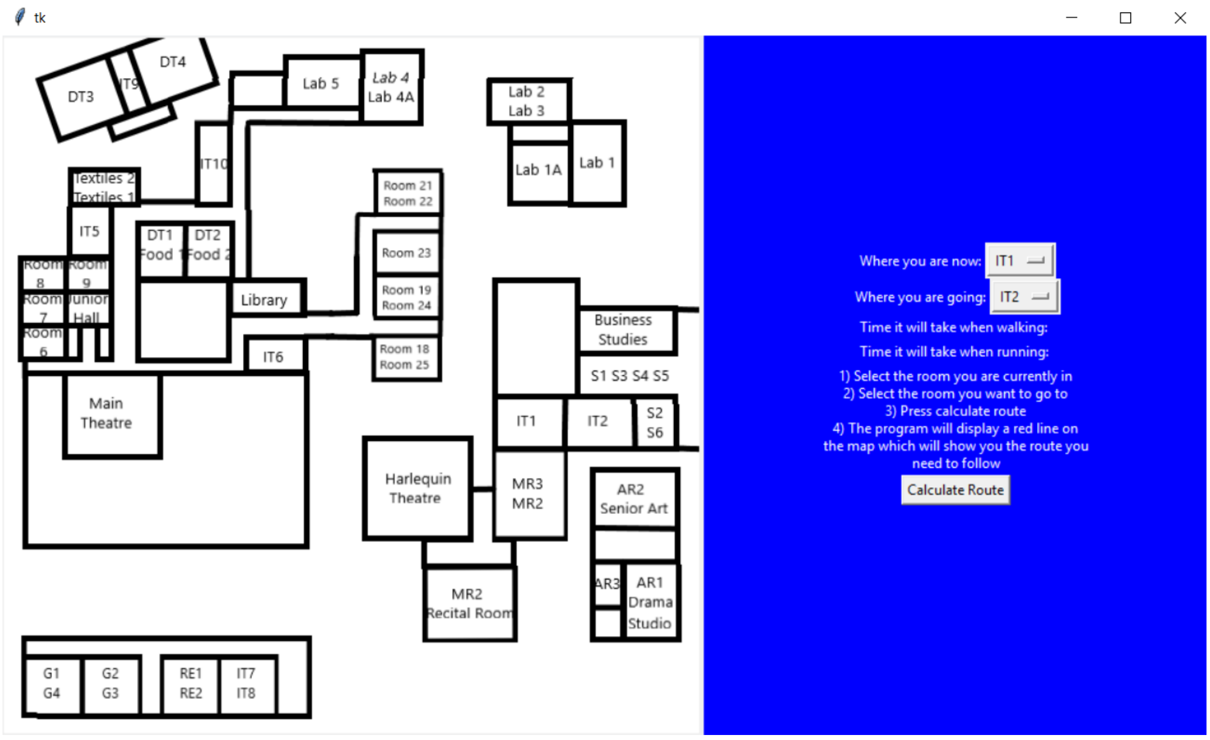
Where you are now: DT1
Where you are going: IT1
Time it will take when walking:
Time it will take when running:
1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow

I also decided to test that the error messages disappeared as well.



Where you are now: IT1
Where you are going: IT1
Time it will take when walking:
Time it will take when running:
1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow

Error: you cannot enter the same room



Where you are now: IT1

Where you are going: IT2

Time it will take when walking:

Time it will take when running:

1) Select the room you are currently in
2) Select the room you want to go to
3) Press calculate route
4) The program will display a red line on the map which will show you the route you need to follow

Calculate Route

Beta testing

Overview

Evaluation

Bibliography

Imms, D. (2016, May 28). *A* pathfinding algorithm*. Retrieved from Growing with the Web: https://www.growingwiththeweb.com/2012/06/a-pathfinding-algorithm.html

Joshi, V. (2017, April 10). *Breaking Down Breadth-First Search*. Retrieved from Medium: https://medium.com/basecs/breaking-down-breadth-first-search-cebe696709d9

OnAverage. (n.d.). *Average Running Speed*. Retrieved from OnAverage: https://www.onaverage.co.uk/speed-averages/average-running-speed

PC Plus. (n.d.). *How your sat-nav works out the best route*. Retrieved from techradar: https://www.techradar.com/uk/news/car-tech/satnav/how-your-sat-nav-works-out-the-best-route-677682/2

Rachit Belwariar, V. M. (n.d.). *A* Search Algorithm*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/a-search-algorithm/

Ridder, A. D. (2018, February 23). *Depth First Search Algorithm: What it is and How it Works*. Retrieved from Edgy Labs: https://edgylabs.com/depth-first-search-algorithm-what-it-is-and-how-it-works

Thaddeus Abiy, H. P. (n.d.). *Dijkstra's Shortest Path Algorithm*. Retrieved from Brilliant: https://brilliant.org/wiki/dijkstras-short-path-finder/

Wikipedia. (2019, January 15). *Walking*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Walking

Appendices

One: Interview questions

Questions and answers from my interview with Mr Reid, my client, on Tuesday 9th October 2018.

1) How much information should be shown to the user (e.g. average time, distance)?

Mr Reid said that the only extra information the users should be given is the time taken, since he thought that that was the only important information to a year seven student.

2.1) Should the program show a list of instructions that the user needs to follow as well as the route on the map?

Mr Reid said that instructions should only be shown if it is done in real time.

2.2) If yes, how detailed?

Mr Reid said that they should be very basic (e.g. turn left, turn right).

3) How detailed should the map of the school be?

Mr Reid said that the map should be about as detailed as the site map used by the school.

4.1) Should the map be split into different sections with a checkpoint in each section, meaning that instead of having one route from start to finish it will send you between checkpoints until you get into the same section as the room you want to go to, where it will send you from that sections checkpoint to the room? Checkpoints would have computers where students could view the next part of their route.

Mr Reid said that this was a good idea, but the checkpoints would need many computers in order to be able to handle the number of students using the system.

4.2) If yes, how many should there be?

Mr Reid said that there should be no more than four checkpoints and sectors.

5) During times like lunch and break, should the path be allowed to go through rooms, for example, room 6 and junior hall, which can be used as a shortcut from outside to the main corridor?

Mr Reid said that it should not be able to plot the route through rooms.

6) How interactive should the map be, for example, selecting a building could show you a more detailed map of the building?

Mr Reid said that it would be a good idea for students to be able to select buildings in order to see a more detailed map of the room.

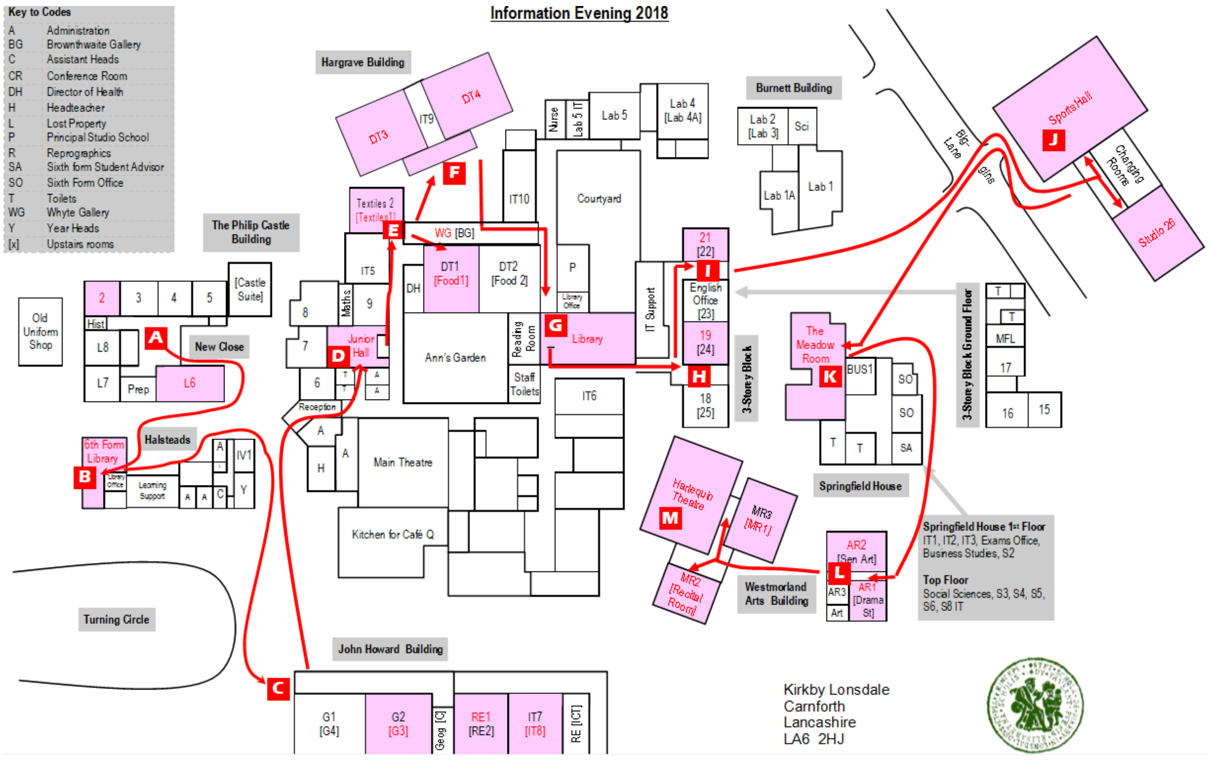
7) Should the main corridor be blocked off at lunchtime?

Mr Reid said that it should be blocked off.

8) What is the current method for helping year sevens to find their way around the school?

Mr Reid said that the current method used to help new year seven students to find their way around the school was an activity where they had to follow instructions and find clues around the site.

Two: Site map



A map of the school used by Sixth Form tour guides. The arrows do not matter for my project.