-Project Title:
How the arrival speed of vehicles between two points, can be made quicker through investigating pathfinding algorithms and seeing which can find the shortest route in the shortest amount of time?

-Name: Kalliszta Grof
-Candidate Number: 3993
-Barton Peveril College
-Centre Number: 58231

0 Contents

1 Analysis

1.1 Statement of Investigation

I am planning to investigate whether the arrival speed of a vehicle between two points A and B can be increased, through investigating pathfinding algorithms. From carrying out his investigation I will be able to answer the question: “Which pathfinding algorithm is the best to use in order to determine the shortest route between two points?”. A possible application of the outcome of my investigation can be used to help ambulances get from A to B quicker. This would aid the ambulance services in this time of crisis due to the COVID-19 outbreak, as the need for the ambulance services has drastically increased. Therefore, ambulance services now need to make the most of every second even more than before, as every second counts when saving lives, so the ambulance need to get from point A to B as quickly as they can. If the ambulance services don’t make the most of every second (e.g. go a longer route when going from A to B), the potential/chances of them saving a person in a life-threatening state decreases.

As I will be investigating pathfinding algorithms, within my investigation I will ignore factors such as traffic, speed limits, maximum speed of vehicle etc which could affect the arrival speed of vehicles in reality and focus on the distance of the path. In my investigation I will be changing the road/maze which the pathfinding algorithms will be applied to and seeing which algorithms can find the shortest distance in the shortest amount of time.

1.2 Background

My investigation’s main purpose is to determine whether the arrival speed of vehicles between two points can be increased. Despite my investigation not being specifically carried out for the ambulance services (but for everyone who drives a vehicle), they are a good example of who my investigation could benefit, which is why I have decided to give a background of why they could benefit from my investigation.

The ambulances services have always needed to get from point A (the hospital) to point B (the location of the person in need of help) and potentially back to point A (back to the hospital) as quickly as they could, especially when the situation of the person needing the help was life-threatening/time-dependent as every second counts. If the situation was life-threatening and the ambulance services were to go a longer route they would lose precious seconds and potentially lose the person (they may die) in need of saving. However, due to the current situation (the COVID-19 break), ambulance services are required even more, so they need to get from point A to point B (and then back to A) in the shortest possible time (therefore via the shortest route). As the longer the route, the longer it is likely to take to get from A to B.

Pathfinding algorithms could be used to aid the ambulance services from getting from point A to B, as pathfinding algorithms are algorithms that find the possible route(/s) from A to B. In order to help the ambulances get from point A to B in the shortest amount of time (therefore the shortest route); the pathfinding algorithms I investigate will need to be ones that can find the shortest route when getting from A to B (as some pathfinding algorithms only find a route from A to B and not the shortest route). Each pathfinding algorithm takes a different length of time to find the shortest route, for example one

pathfinding algorithm may take 1 second and another may take 2 seconds. Therefore, I will investigate pathfinding algorithms to see which algorithms are suitable (the quickest at finding the shortest route) for different types of areas. By using the research/information I have learnt through investigating pathfinding algorithms I will create a program that will compare different pathfinding algorithms and use the results to help vehicle drivers such as ambulances get from point A to B quicker.

COVID-19 is the virus that resulted in the lockdown in 2020, it has affected lots of people and has killed over 40,000 people in the UK. By increasing the arrival speed of ambulances this number can increase at a decreasing rate (slower and slower), due to ambulances getting to point B from A in time. However, just because the ambulance services arrive quickly doesn't mean the person will survive, the ambulance may be unable to help them. But by arriving quickly the chances of being able to help someone are higher.

1.3 End User / Expert

As I am required to investigate pathfinding algorithms for my project, I require an expert in algorithms who I can interview to learn more about suitable algorithms that I can investigate for my project. I have chosen Nyki Inskip to be my expert as she has been a teacher for 11 years, and she currently teaches A-level Maths and Computer Science, therefore is an expert in algorithms. I can ask her for information about algorithms already out there as well as the pros and cons of them. The information I gather from her will help me determine what the best algorithms are for me to investigate/compare.

1.4 Initial Research

1.4.1 Existing, similar programs

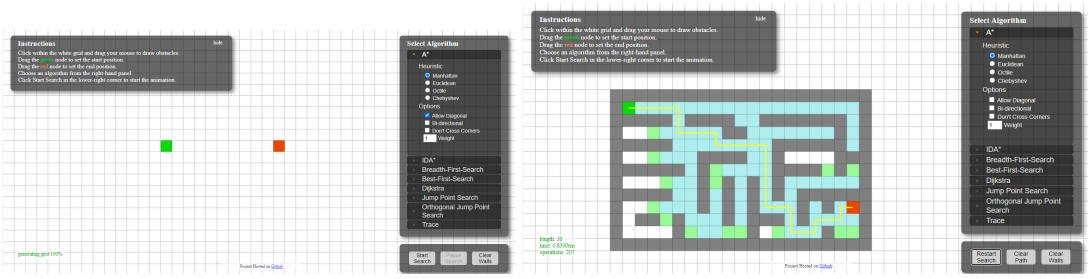
Before interviewing Nyki (my expert) to determine what pathfinding algorithms I will investigate, I have decided to look into existing programs that compare pathfinding algorithms as the program I create will compare pathfinding algorithms as well. By researching similar programs, I can learn about useful/good features to have in a pathfinding algorithm comparison program. But as well as that, I can find out about features to avoid when making my program.

1.4.1.1 Existing / Similar program 1

https://qiao.github.io/PathFinding.js/visual/

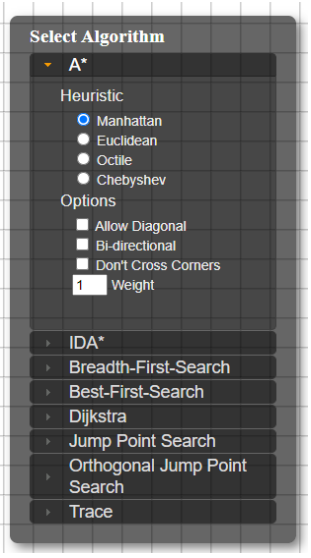
I found a website (https://qiao.github.io/PathFinding.js/visual/), that provides an interactive interface and allows the user to create mazes and simulate/perform pathfinding algorithms on them. Despite the pathfinding algorithms in the website being used to solve mazes generated by the user, the mazes could be used to represent the streets/roads that vehicles can go on. As described in the instructions, the program is easy to use as to move the starting (green square) / end (red square) square/position (of the maze), all the user has to do is click on the (green/red) square and drag it somewhere within the grid. In order to create the maze, the user can click/hold the mouse on any white square in order to draw the walls (the walls will appear as grey squares). The user can convert walls (grey squares) back into space (white squares) by clicking/holding on them using the mouse. In the top right the user can choose from a

variety of pathfinding algorithms to perform on the maze which they have created. In the bottom right corner the user has access to options which allow them to do more things with the program e.g. run the selected pathfinding algorithm or clear all the walls. The website itself doesn't compare the pathfinding algorithms (e.g. in a table), however it does measure the time it takes for the pathfinding algorithm to find the shortest route (in milliseconds). This website has many pros, however it also has a few cons.

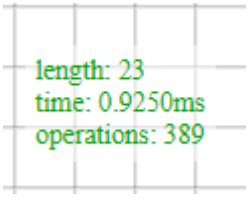


Pros:

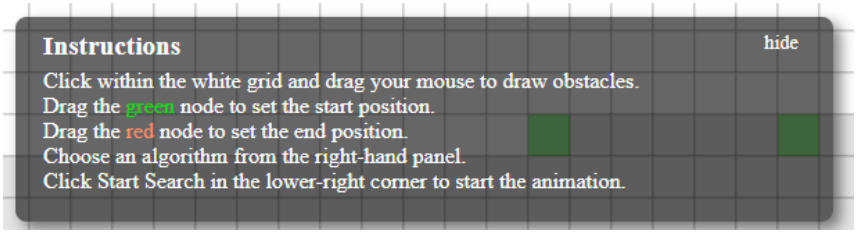
- Allows the user to easily choose between a selection of pathfinding algorithms.



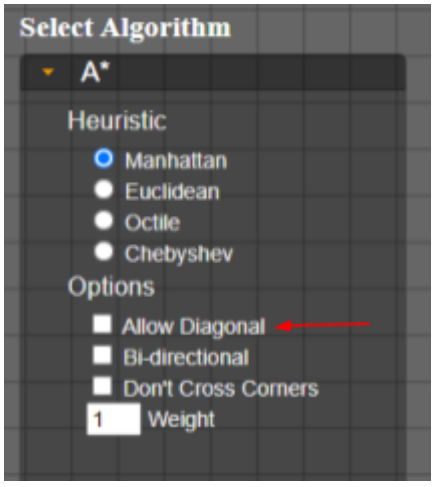
- In the bottom left corner it displays:
  - the shortest path found (in terms of square), (length)
  - the time it took (in milliseconds), (time)
  - the number of operations which were carried out, (operations)
  - In addition to those things being displayed, the number of squares that need to be traversed/looked at for the pathfinding algorithms to find the shortest route between the start square and end square could also have been displayed/kept track of as it enables the audience to compare how many squares each pathfinding algorithm needs to transverse to find the shortest route.



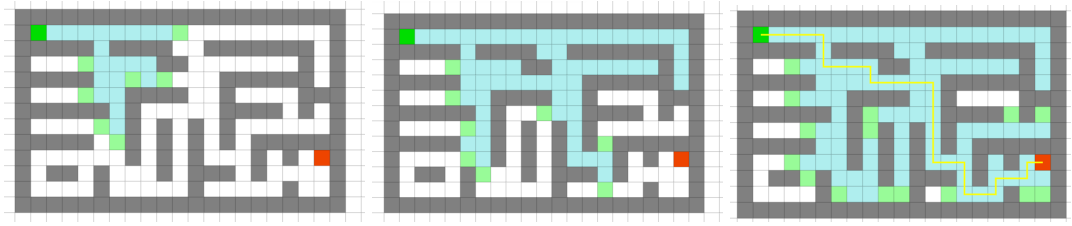
- The program is easy to use, and provides the user with instructions in the top left corner.



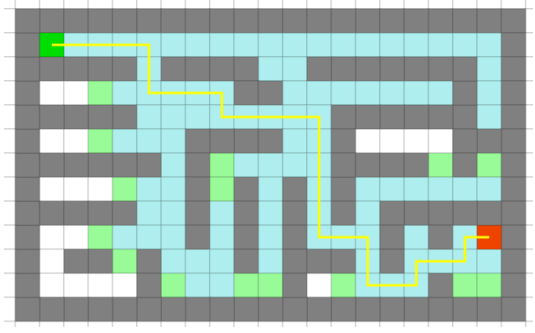
- On the right where the user can select the pathfinding algorithms, there are some options that the user can choose from (e.g. “Allow Diagonal”).



- Each pathfinding algorithm is animated. The website shows what square has **already been** traversed/checked in light blue, it also shows what square is **currently** being traversed/checked in a light green, allowing the user to visually see how each of the pathfinding algorithms work (e.g. images below show how the A* pathfinding algorithm works with the options shown above).

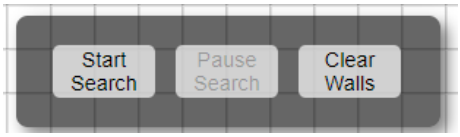


- At the end (once the searching/traversing has finished), the shortest path found is shown visually using a yellow line. In my project’s context, the yellow line would represent the route which a vehicle must take to get to the required location in the shortest amount of time (by taking the shortest route).



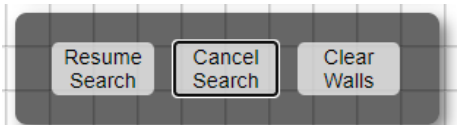
- In the bottom right corner, there are three buttons which let the user do things with the program, e.g. the option like “Clear Wall” is a good feature as it speeds up the process of removing walls. The options change depending what state the program is at/in, this is good as the program doesn’t provide the user with options that aren’t available to be performed (however the Pause Search option is shown to be inactive/unclickable by it being greyed out as it shows the user that they will be able to stop the program once they run it):
    - Before a pathfinding algorithm is run the options are:

        - Start Program (run the selected pathfinding algorithm)
        - Pause Search (which cannot currently be clicked on)
        - Clear Walls (which allows the user to quickly get rid of all of the walls they have drawn/added)



    - When a pathfinding algorithm is mid-way through running the options are:
        - Restart Search
        - Pause Search (now clickable)
        - Clear Walls

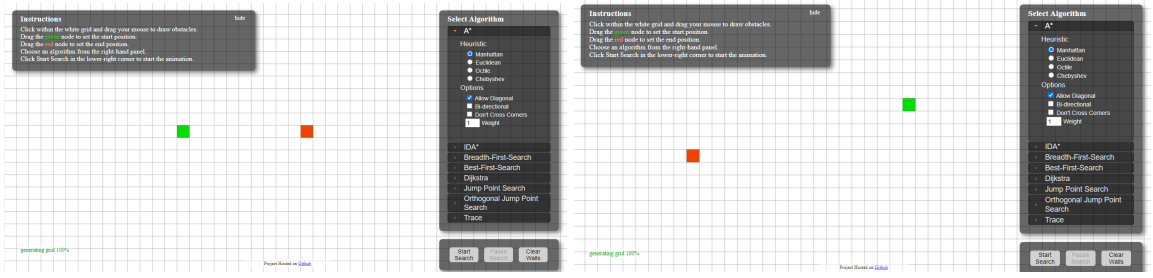
    - When a pathfinding algorithm is paused by the user the options are:
        - Resume Search
        - Cancel Search
        - Clear Walls



    - When a pathfinding algorithm is mid-way through running the options are:
        - Restart Search
        - Clear Path
        - Clear Walls

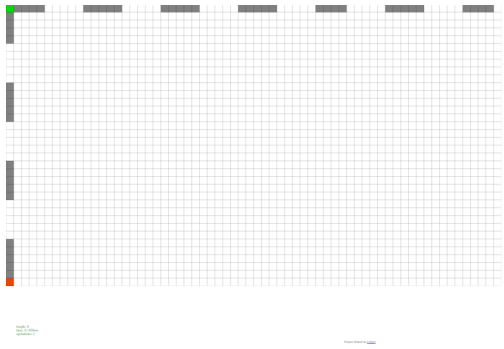
- Another good feature is that the user can see what process they ran last, as it is shown by having a border around the process/option clicked on last.

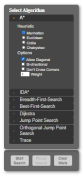
- In addition, the program lets the user choose to move/place the start/end nodes/squares wherever they want.

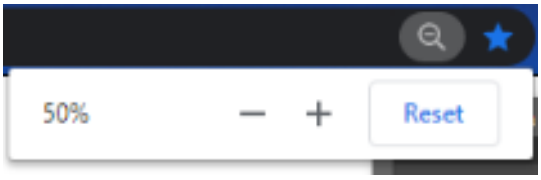


Cons:

- Despite the fact that the time taken to find the shortest path is shown, the user cannot choose to compare the time taken for pathfinding algorithms to run within the program. In order to compare the algorithms, they would need to manually make note of the times then compare them, this would be inefficient as it would take a long time, therefore it would have been a good feature in the program.
- The program doesn't let the user save the results of the pathfinding algorithm (e.g. the time taken) so they cannot access the results once they close/refresh the web page. A useful feature could be to allow the user to save the results into a text file/binary file/database which they could download.
- In addition, it doesn't let the user save the maze that they have created, so they lose the maze that they created after the web page is closed/refreshed.
- The user cannot import mazes. The feature to import mazes could be a useful feature if the user wanted to find the shortest route etc of a specific maze (currently they would have to manually spend time creating the maze using the program).
- There is no option for the user to get a randomly generated maze.
- Another disadvantage of this program/website is that the use cannot change the size of the grid (/number of squares) from 65x36. The maximum size of the grid can only be seen when the 'zoom' in the page is at e.g. 50% (not at 100%) therefore users who are unfamiliar with the zoom function may not even get a grid that big. When creating my own program for comparing pathfinding algorithms, I will consider that some 'mazes' (/roads) may require a larger maximum grid size than this program provided, therefore the program I create will prompt (ask) the user to input the dimensions of the grid themselves (however there will likely still be a certain range to avoid the program from crashing).



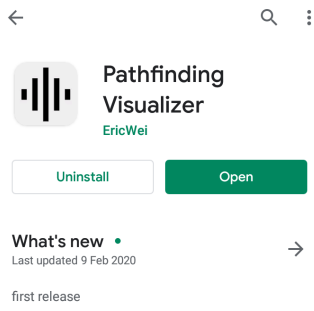




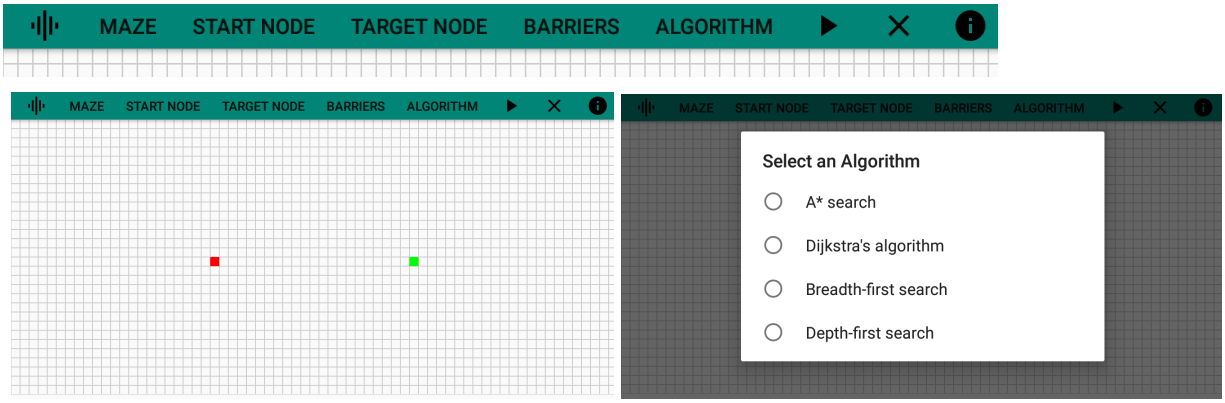
1.4.1.2 Existing / Similar program 2

https://play.google.com/store/apps/details?id=com.ericwei.pathfindingvisualizer&hl=en

In order to gain more insight about features that a program which compares pathfinding algorithms may require, I have decided to look at another similar/existing program. The 2nd program I have looked into is an app called: "Pathfinding Visualizer" (made by "EricWei") that can be downloaded via the "Google Play Store" (app) onto your phone. I downloaded the app so that I could learn about any features my program could have. "Pathfinding Visualizer" is a similar program to what I will be creating as it can find the shortest distance between two points, it allows the user to choose from various pathfinding algorithms. However the program doesn't measure the

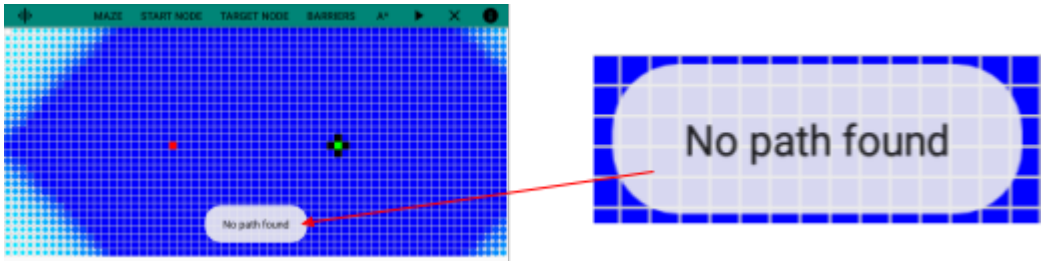


time taken for the pathfinding algorithm to be run, therefore it cannot be used to compare the different pathfinding algorithms. In this app the options are at the top; in order to move the starting square/node (the red square), the user must click on the word “START NODE” which will then enable them to choose a location on the grid to move it to. The same applies to the end square/node (the green square), only this time the user should need to click on “TARGET NODE”. The app lets the user draw their own walls/barriers to create a maze (if the user presses the word “BARRIERS”), however the user can also choose a maze to be randomly generated through pressing “MAZE”. The user can choose from a selection of algorithms (A* Search, Dijkstra’s Algorithm, Breadth-First search, Depth-First Search) by clicking on “ALGORITHM”. In order to run the selected algorithm they must press the play button (the arrow icon). To clear/reset the entire grid the user must press the cross. The “i” in a circle provides information/instructions on how the app works, it provides links to the wikipedia pages of each algorithm allowing the user to read about each algorithm in depth.

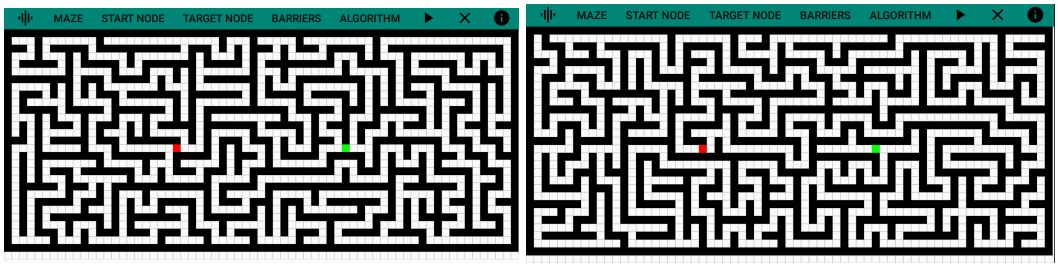


Pros:

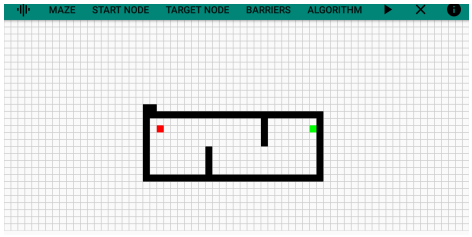
- If there is no possible route from the starting square/node to the target/end node/square then it tells the user by saying “No path found”.



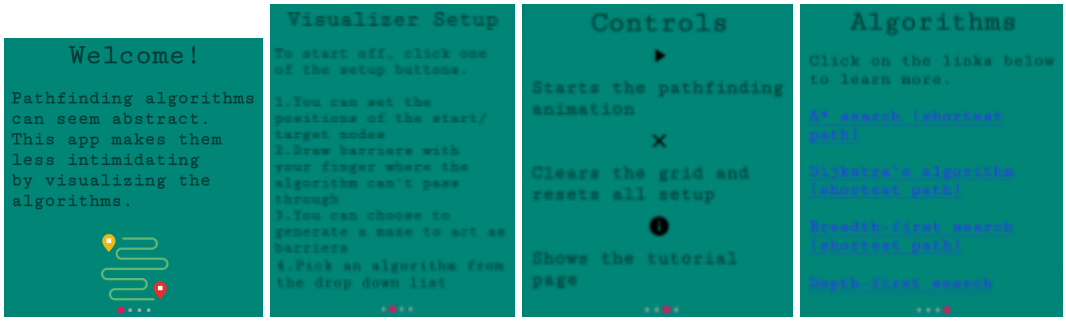
- Program can randomly generate a maze which a pathfinding algorithm can be run on by the user.



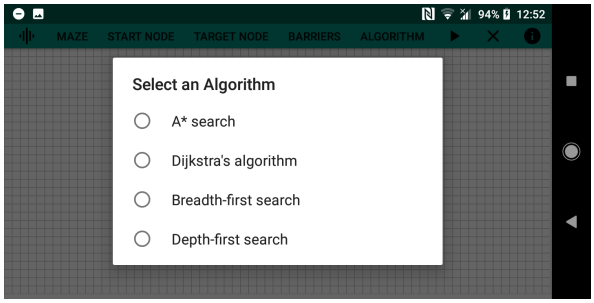
- The program allows the user to draw their own walls/barriers.



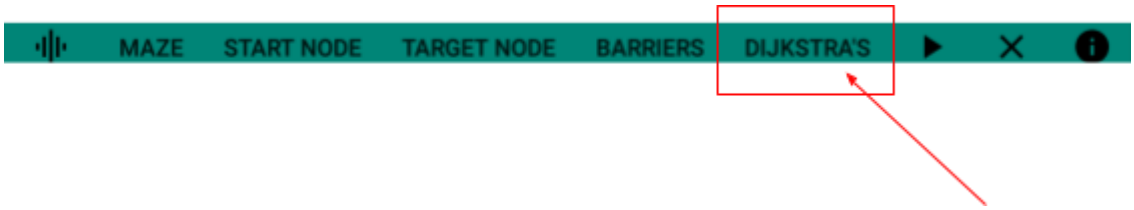
- The program provides the user with instructions/information (however to access it the user most press the “i” (in the circle) in the top right corner so it would have been useful if it appeared every time the user opened the program or at least the first time):



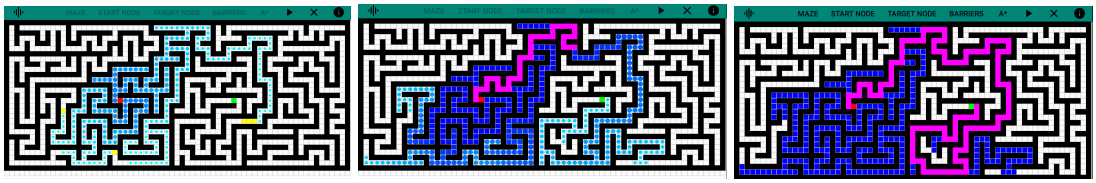
- The user can easily select from four types of pathfinding algorithms.



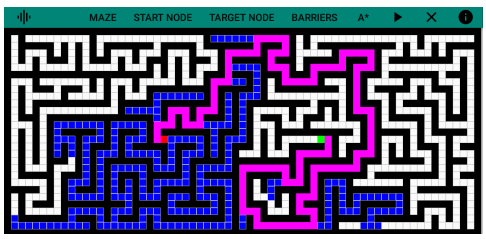
- The program displays what pathfinding algorithm is currently selected at the top.



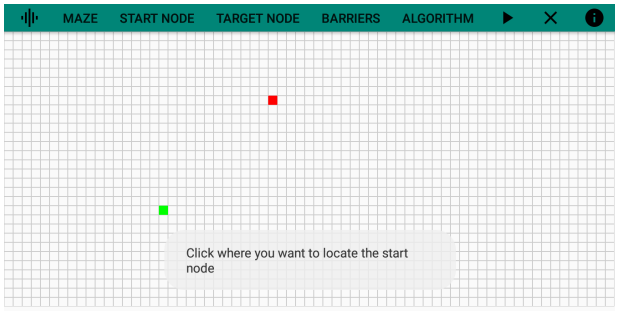
- The application animates each pathfinding algorithm to demonstrate how they work. The more recent squares visited (by the pathfinding algorithm) are light blue (the more recently visited the lighter) and have circles (the more recent the smaller the circle) in them. The yellow squares show the current squares being visited/traversed by the pathfinding algorithm. I believe the 1st program I researched dealt with showing the squares already/currently being traversed in a more suitable way, as they only used two shades of colours (blue for the squares that were already traversed and green for the ones being traversed currently by the pathfinding algorithm) making it easy for the user to understand/see what was/is going on.



- The shortest path is shown by highlighting the squares from the starting square to the target square in a bright pink colour.

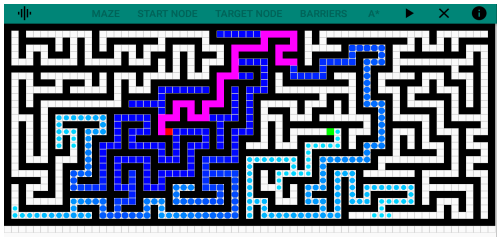


- The program allows the user to move the start & target node where they want them to be (anywhere in the grid) .



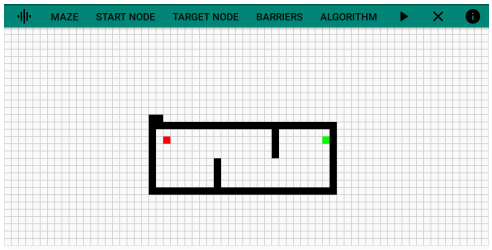
- At the bottom of the screen the program displays the “Path of length” in squares from the start to target/end square/node.

- While the program is running the user **cannot** make changes to the maze, the position of start/target node or change the algorithm being run. These options are made unavailable through making the options on the options bar unclickable and visually showing this through making them greyed out.

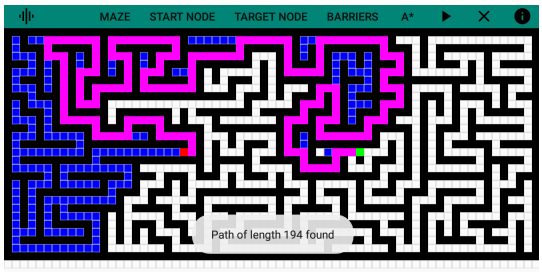


Cons:

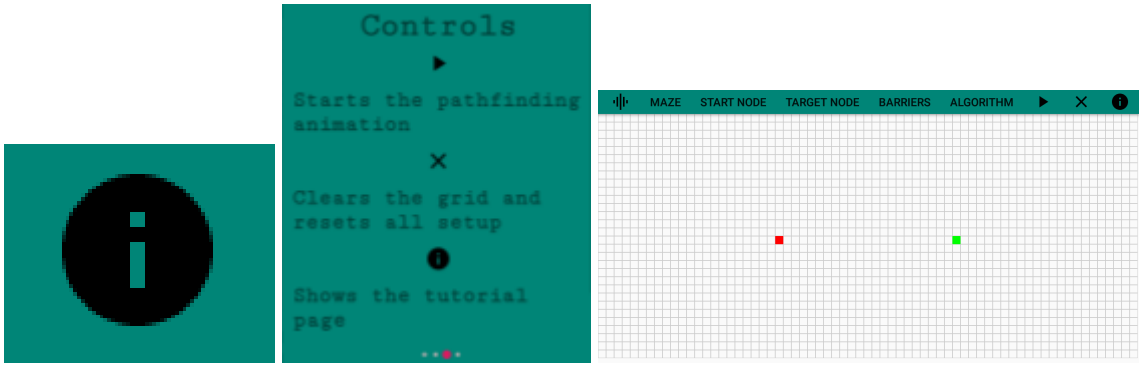
- Unlike in the 1st existing program I researched, the user for this program cannot get rid of a wall/barrier they have added (one-by-one), they can only get rid of them if they completely reset the entire grid. So if the user makes a mistake when drawing the walls/barriers they need to start over.



- There is no save functionality that allows the mazes to be saved.
- The user cannot import their own mazes
- The length of the shortest route (in squares) is only visible/displayed for a short amount of time (it would be better if it required the user’s prompt to show that they read it and only disappear after).



- The entire grid only 67 x 30 squares big, the size cannot be changed
- The time taken for the pathfinding algorithm to find the shortest route is not show
- The user cannot specifically choose to test each of the algorithms on a maze with one solution / multiple solutions when choosing to randomly generate a maze.
- The user is not provided with any instructions when they first use the app they must press the “i” for information & instructions.



1.4.2 Key elements within existing/similar programs

From researching similar/existing programs I have decided to include the following features in my program (the ones in **bold** are features that aren’t in the similar/existing programs but would be a useful feature to include):

Key:
* = feature of 1st program
# = feature of 2nd program
(x) = the project objective they link to

- *The feature of being able to choose from a variety of pathfinding algorithms. * # (1.1.)*
- *Displaying the time taken (in milliseconds). * (3.1.3.4.) (3.3.)*
- *Extension: Displaying the length of the shortest path (in squares). * # (not in objective due to more research later)*
- ***Displaying how many squares/nodes have been traversed/searched before the shortest path from the start square/node to the end/target square/node is reached/found. (3.1.3.6.) (3.3.)***
- *Visually showing to the user how the pathfinding algorithm works (this is a good feature to have however it may not help much when comparing the different algorithms so I may decide not to include it). * # (2.5.2.)*
- *Visually displaying the shortest path. * # (2.5.3.)*

- ***A feature to store the result of how long the pathfinding algorithm took to complete in e.g. a text file/binary file/database, so that it can be accessed after the program is closed and allows the user to compare e.g. the time taken for various pathfinding algorithms to run on specific mazes. (3.1.3.4.) (3.3.)***
- *The feature to randomly generate mazes (in order to save the user time from manually creating mazes like in the existing program). # (2.1.1.)*
- ***The feature to let the user input the dimensions of a maze they want to be generated/created. However, there will still be a certain range to avoid the program from crashing. (2.1.1.2.) (2.1.1.3.)***
- ***The ability to let users import mazes (roads) from e.g. a text file, binary file, database, etc (2.1.2.) (2.3.1.) (3.1.1.)***
- ***The feature to let the user save randomly generated mazes (2.3.)***
- *The feature to allow the user to choose where to have the start & target/end squares/nodes. * # (2.4.)*
- ***Extension: The feature to let the user choose whether they want the maze to have one/multiple solution(s). (2.6.)***
- *The feature that provides the user access to instructions on how to use the program. * # (4.1.)*

1.4.3 Potential data structures / algorithms

1.4.3.1 Algorithms

When looking into potential algorithms that I could investigate I found the following algorithms that could be good to investigate/research more: Breadth-First Search, Depth-First Search, A* Search and Dijkstra's algorithm. By investigating these pathfinding algorithms I will know/have a better idea about which ones would be suitable to use to find the shortest path/route between two points in a road (represented using a maze). A road/maze could be represented in multiple ways for example using squares in a grid, however they can also be represented using graphs. In a graph the distance between each e.g. junction (where two or more paths cross) could act as a node, each node/vertex would be connected via edges which would have an associated value (a weight) assigned to them representing the distance (in e.g. km). For my investigation I will choose to program some of these algorithms in order to see how the arrival speed of vehicles between two points can be made quicker through investigating pathfinding algorithms.

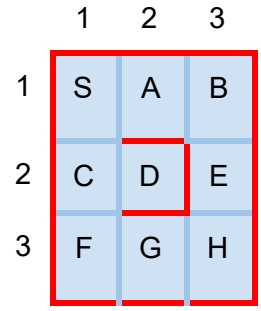
1.4.3.1.1 A* Search

A* search is a pathfinding algorithm that can find the shortest path/route between two points/nodes. When the A* search is run on e.g. a maze which is divided into squares the A* search may be able to travel to all adjacent squares even to ones that are diagonal and not just to the vertical/horizontal neighbours of the square it is currently on (depending on how it is programmed the A* search has been programmed). Each square is given a cost, diagonals' cost are worked out using Pythagoras' Theorem ($a^2 + b^2 = c^2$). I.e if the square was given a cost of 1, meaning that a triangle with a length (a) of 1 and height (b) of 1 was created, the diagonal (c) cost would be √2 (~1.41). (As $1^2 + 1^2 = 2 = c^2$, therefore c = √2 (~1.41)).

Assuming the A* search I am describing cannot travel diagonally. The first step involved in a A* search is checking whether the current square being traversed is the end square (in the first recursion the start

square is the current square and it would be marked as discovered). Next, the G cost and H cost for each of the adjacent squares of the current square is calculated and the current square is marked as explored/traversed. G cost is the distance/cost from the starting square and H cost (heuristic) in the distance/cost from the end square. To work out the H cost you must estimate the distance from the current square to the end square, to do this, for each square you take the x-coordinate of the end square and take away the x-coordinate of the current square being traversed and then square it. After that you do the same for the y-coordinates and add the y-coordinates' result to the calculated number from the x-coordinates (you would then need to square root the number, however you don't need to as the ratio remains the same either way so it is better to not to avoid complication with roots). The next step is to work out the F cost of those squares which is just G cost + H cost. Each adjacent square is marked as discovered and is added to a priority queue which is ordered by the F cost (lowest to highest). The square at the front of the queue is chosen to be the next square to be explored and is removed from the queue. But if there are squares with the same F cost then the program looks at the H cost of each of those squares and chooses the one with the lowest H cost (if that is also equal then the program chooses to visit one of the squares with the same F and H cost randomly). This process is repeated (using recursion) for each square in the queue until the end square is found (or the queue is empty). If at any time during the recursions the adjacent square of the current square have a lower G cost than before (as they were discovered using a different route) then the G, H and F costs of them are updated. When the end square is reached whatever the F cost is will be the shortest path (cost). In order to get the path to the end square from the start square, each square can have a parent/previous square associated with it which basically stores the information about which route it was discovered and then traversed from. By starting from the end square and backtracking the previous squares you can reach the start square and therefore know the shortest route/path between the start and end squares. If I choose to program an A* Search, I will likely program it so that it can only travel to adjacent squares and not diagonal ones. As the A* search being able to travel diagonally could affect the investigation because some pathfinding algorithms may not be able to travel diagonally.

Example of an A Search:*



Priority Queue (A to E):
1) S
2) A, C
3) B, C
4) E, C
5) C (Stops as E was found)

Path (by backtracking using previous squares):
S, A, B, E

1)

Square	S	A	B	C	D	E	F	G	H
Traversed	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
F Cost	5	0	0	0	0	0	0	0	0
H Cost	5	0	0	0	0	0	0	0	0
G Cost	0	0	0	0	0	0	0	0	0
Previous Square	-	-	-	-	-	-	-	-	-

2)

Square	S	A	B	C	D	E	F	G	H
Traversed	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
F Cost	5	3	0	5	0	0	0	0	0
H Cost	5	2	0	4	0	0	0	0	0
G Cost	0	1	0	1	0	0	0	0	0
Previous Square	-	S	-	S	-	-	-	-	-

3)

Square	S	A	B	C	D	E	F	G	H
Traversed	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
F Cost	5	3	3	5	0	0	0	0	0
H Cost	5	2	1	4	0	0	0	0	0
G Cost	0	1	2	1	0	0	0	0	0
Previous Square	-	S	A	S	-	-	-	-	-

4)

Square	S	A	B	C	D	E	F	G	H
Traversed	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE
F Cost	5	3	3	5	0	3	0	0	0
H Cost	5	2	1	4	0	0	0	0	0
G Cost	0	1	2	1	0	3	0	0	0
Previous Square	-	S	A	S	-	B	-	-	-

5)

Square	S	A	B	C	D	E	F	G	H
Traversed	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE
F Cost	5	3	3	5	0	3	0	0	0
H Cost	5	2	1	4	0	0	0	0	0
G Cost	0	1	2	1	0	3	0	0	0
Previous Square	-	S	A	S	-	B	-	-	-

Pros:

- The algorithm always finds the shortest route (if it is possible to get from A to B).
- The algorithm can be programmed to travel diagonally or not
- Explores squares/nodes that appear promising (the ones with the lowest cost/distance)
- Can store the previous/parent squares to get the path from the start to the end **(3.1.3.5.)**

Cons:

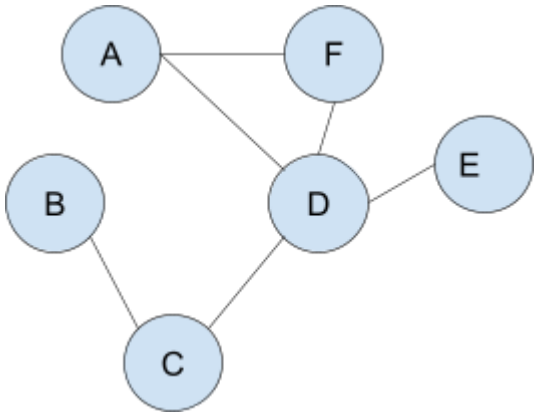
- Isn't good for reaching multiple end/target squares/nodes. However, this issue can be overcome by running A* search on each end/target squares/nodes individually.

1.4.3.1.2 Breadth-First Search

Breadth-First Search is a pathfinding algorithm which can find the shortest route between two points A and B using a queue. Therefore the algorithm explores the node that has least recently been discovered (so the one in front of the queue) (first in, first out).

The queue starts out initially empty, then the first item/node is added into the queue and marked as discovered. The discovered state of a node is stored by the program through using a boolean variable (e.g. called discovered), where if the node is discovered the boolean is True and if it isn't discovered the boolean is False. The first node is removed from the start of the queue and its neighbours are added to the back of the queue and marked as discovered. The program then looks at what node is at the start of the queue, removes it and marks its neighbours as discovered. Every time a node new is discovered by the algorithm it is added to the end of the queue (if any of the neighbours of the node were already discovered they aren't added into the queue). This process is repeated until the end square is found or the queue becomes empty. Just like in the A* Search the previous/parent square of each square can be kept track of in order to get the shortest path from the start square/node to the end square/node.

Example of a Breadth-First Search:



Queue (at different stages) when getting from A to B:

1) 2) A 3) D, F 4) F, C, E 5) C, E 6) E, B 7) B 8)	==========>	Once B (the end) is discovered and then traversed it is found.	If the previous squares were kept track of then the shortest path would be: A, D, C, B
1) *Discovered Boolean:* A = False B = False C = False D = False E = False F = False	2) *Discovered Boolean:* A = True B = False C = False D = False E = False F = False	3) *Discovered Boolean:* A = True B = False C = False D = True E = False F = True	4) *Discovered Boolean:* A = True B = False C = True D = True E = True F = True
5) *Discovered Boolean:* A = True B = False C = True D = True E = True F = True	6) *Discovered Boolean:* A = True B = True C = True D = True E = True F = True	7) *Discovered Boolean:* A = True B = True C = True D = True E = True F = True	8) *Discovered Boolean:* A = True B = True C = True D = True E = True F = True

Pros:

- It will always find a route from the starting point/node (A), to the end/target point/node (B), if there is one.
- It can always find the shortest route/path between two points: A and B
- Can store the previous/parent squares to get the path from the start to the end **(3.1.3.5.)**.

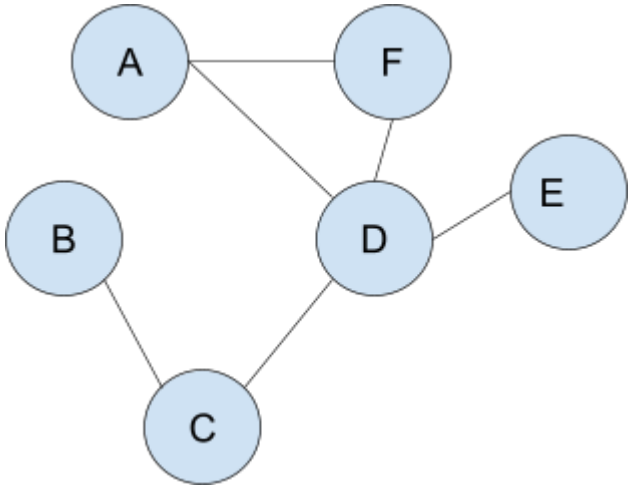
Cons:

- If the starting point/node and end/target point/node are far away from each other, the pathfinding algorithm may take a long time to finish.
- The data about the graph needs to be stored by the program in some way e.g. in an adjacency matrix (can be created in a program using a 2D array) or an adjacency list (can be created in a program using an array of lists/dictionaries).

1.4.3.1.3 Depth-First Search

Depth-First Search is a pathfinding algorithm that is used on graphs/trees (like in the image below). Unlike Breadth-First Search it explores the most recent node discovered (first in, last out) as it uses a stack. Before a Depth-First Search algorithm is run on a graph, each node has to have information stored about in e.g. an array. The stored information will be a boolean value that stores whether the node has been previously discovered (True) or not (False), so that when the algorithm is run it won't infinitely visit/traverse neighbours of a node that have already been discovered. Depth-First Search uses a stack (which is initially empty) to determine what node to traverse next. For example, for the diagram below, the starting node is 'pushed' (put) on the stack and the boolean value of A is stored as True (to store that it has been discovered) by the program, A is then popped off the stack. After that, all of the associated neighbours of A are pushed onto the stack (in this case D and F) and their boolean value is stored as True to store that they have been discovered. Now whichever letter is on top of the stack gets visited next (F). The program 'pops' off (removes) the node (F) from the stack and sees whether all of that node's (F's) neighbours have been discovered, if they haven't then the undiscovered nodes become discovered and are added (pushed) onto the stack. However, if there aren't any undiscovered neighbours then nothing is added (pushed) onto the stack. The program then looks at the item at the top of the stack and repeats this process until the stack is empty again. If the stack becomes empty again it means that all possible nodes have been discovered. Just like in the previous two algorithms the previous/parent square of each square can be kept track of in order to get the shortest path from the start square/node to the end square/node. The images show an example of a Depth-First Search being carried out on a graph.

Example of a Depth-First Search:

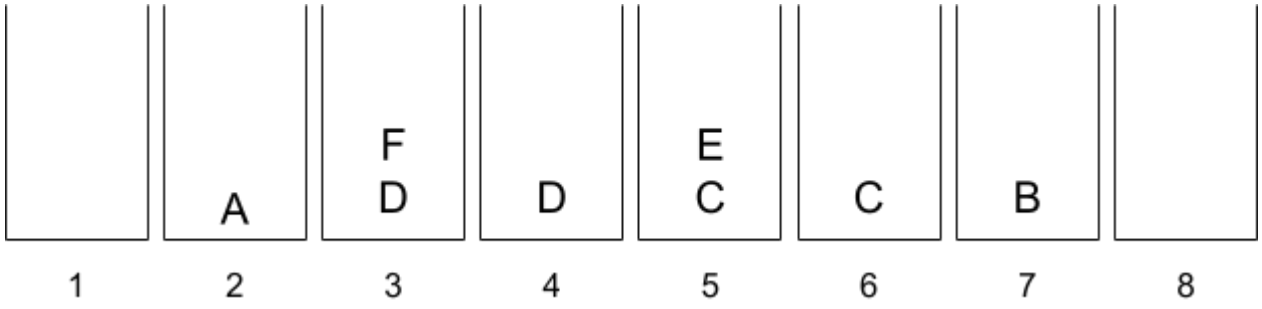


(The process of the algorithm being run is shown in the diagram on the next page.)

If the previous squares were kept track of then the shortest path would be:

A, D, C, B

The Stack (at different stages):



1)	*Discovered Boolean:* A = False B = False C = False D = False E = False F = False	2)	*Discovered Boolean:* A = True B = False C = False D = False E = False F = False	3)	*Discovered Boolean:* A = True B = False C = False D = True E = False F = True	4)	*Discovered Boolean:* A = True B = False C = False D = True E = False F = True
5)	*Discovered Boolean:* A = True B = False C = True D = True E = True F = True	6)	*Discovered Boolean:* A = True B = False C = True D = True E = True F = True	7)	*Discovered Boolean:* A = True B = True C = True D = True E = True F = True	8)	*Discovered Boolean:* A = True B = True C = True D = True E = True F = True

Pros:

- Has a less time-complexity (how long it takes for the pathfinding algorithm to find a route between two points A and B) than Breadth-First Search. The time complexity is O(V+E) (where V = vertices and E = edges).
- Can store the previous/parent squares to get the path from the start to the end **(3.1.3.5.)**

Cons:

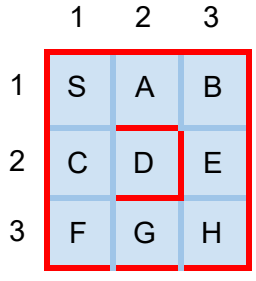
- It is recursive therefore may result in a stack overflow (which is when the stack runs out of memory to store more information), if the graph being traversed/searched is big.
- Depth-First Search cannot be used to find the shortest path as it goes down the first path it finds to get to the end nodes (unless it is made to store the distance from the start node and is modified further).
- May not find a route between two points/nodes (if the program doesn't store which nodes have been traversed).
- Can get stuck in an infinite loop (if the program doesn't store which nodes have been traversed).
- The data about the graph needs to be stored by the program in some way e.g. in an adjacency matrix (can be created in a program using a 2D array) or an adjacency list (can be created in a program using an array of lists/dictionaries).

1.4.3.1.4 Dijkstra's Algorithm

Dijkstra algorithm is a pathfinding algorithm which works using a priority queue to find the shortest distance between two nodes in a graph. A priority queue is a queue that priorities the items (in this case nodes) based on something, in this case the items/nodes with the shortest distance (/lowest cost) will be at the front of the queue. Through research I have learnt that Dijkstra's algorithm acts as a basic starting point in navigation systems for things such as sat navigators and 'Google Maps', therefore it is definitely an algorithm worth investigating.

Dijkstra's algorithm starts by assigning the starting square (e.g. S) a cost/distance of 0 (which would be stored as variable) and assigning the rest of the squares a cost/distance of infinity. Next, all squares are put into a priority queue (the squares with the least amount of cost/distance are put at the front). The program then looks at each neighbour of the current square being traversed (e.g. the starting square) and assigns the actual cost/distance from the start square to that square (the information about the distance between each square would already be stored by the program). Everytime, all the neighbours of the current square are marked as discovered, the priority queue must be updated so that the squares with the shortest distance/cost from the start square are at the front. Each time a neighbour of a square is discovered, the square which the neighbour square was accessed from is stored in a variable e.g. called: previousSquare (previousSquare would only store something like the identifier/name of the previous square not the entire path to get to that square). If the square already stores a previousSquare and a distance, it stores the values of the previousSquare and distance of the shortest path (e.g. if K to N was 6 but K to L to N was 4, the values associated with N would be distance = 4, previousSquare = L). After all the neighbours of a square have been fully explored, the square is removed from the priority queue. The program then looks at the square at the front of the queue and does the same thing to this square, this process repeated until the end/target square is reached. After the end/target square is reached, the program traces the path from the end/target square to the start square by looking at each square's associated previousSquare. The path/route being re-traced from end to start is the shortest path/route.

Example of Dijkstra's Algorithm:



Priority Queue (A to E):

1) S
2) A, C
3) C, B
4) B, D, F
5) D, F, E
6) F, E
7) E,G
8) G

Path (by backtracking using previous squares):

S, A, B, E

1)

Square	S	A	B	C	D	E	F	G	H
Traversed	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Cost	0	∞	∞	∞	∞	∞	∞	∞	∞
Previous Square	-	-	-	-	-	-	-	-	-

2)

Square	S	A	B	C	D	E	F	G	H
Traversed	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
Cost	0	1	∞	1	∞	∞	∞	∞	∞
Previous Square	-	S	-	S	-	-	-	-	-

3)

Square	S	A	B	C	D	E	F	G	H
Traversed	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
Cost	0	1	2	1	∞	∞	∞	∞	∞
Previous Square	-	S	A	S	-	-	-	-	-

4)

Square	S	A	B	C	D	E	F	G	H
Traversed	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE
Cost	0	1	2	1	2	∞	2	∞	∞
Previous Square	-	S	A	S	C	-	C	-	-

5)

Square	S	A	B	C	D	E	F	G	H
Traversed	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
Cost	0	1	2	1	2	3	2	∞	∞
Previous Square	-	S	A	S	C	B	C	-	-

6)

Square	S	A	B	C	D	E	F	G	H
Traversed	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
Cost	0	1	2	1	2	3	2	∞	∞
Previous Square	-	S	A	S	C	B	C	-	-

7)

Square	S	A	B	C	D	E	F	G	H
Traversed	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE
Discovered	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
Cost	0	1	2	1	2	3	2	3	∞
Previous Square	-	S	A	S	C	B	C	F	-

8)

Square	S	A	B	C	D	E	F	G	H
Traversed	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
Discovered	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
Cost	0	1	2	1	2	3	2	3	∞
Previous Square	-	S	A	S	C	B	C	F	-

Pros:

- Will always find the shortest path/route between two points
- Can be used on weighted graphs (weighted graphs are graphs which have edges that have a value associated with them).
- Good to use when you have multiple targets
- Can store the previous/parent squares to get the path from the start to the end **(3.1.3.5.)**

Cons:

- Cannot handle negative edges
- Time consuming (due to it branching in multiple directions).
- The data about the graph needs to be stored by the program in some way e.g. in an adjacency matrix (can be created in a program using a 2D array) or an adjacency list (can be created in a program using an array of lists/dictionaries).

1.4.3.1.5 Comparing

After having researched the four pathfinding algorithms: A* Search, Breadth-First Search, Depth-First Search and Dijkstra's algorithm, I now have a rough idea about how each of these algorithms work as well as the advantages (pros) and disadvantages (cons) of each algorithm. During my research, I found out that all the algorithms can easily return the shortest path from the start square to the end square if the parent/previous squares were kept track of. I also found out that Depth-First Search and Breadth-First Search are relatively similar, the only difference being that Depth-First Search uses a stack to determine the order the nodes are visited/traversed whereas Breadth-First Search uses a queue. Despite Depth-First Search not necessarily being able to find the shortest path, it finds a possible path quicker than Breadth-First Search. However, as my investigation is being carried out to help people in vehicles such as ambulance services get from A to B in the quickest possible time (therefore using the shortest route), it is highly unlikely that I will further investigate Depth-First Search as it usually doesn't provide the shortest route between two point (e.g. A and B), even though the basic algorithm for Depth-First Search could be modified so that it can find the shortest possible route/path. On the other hand, A* search, Breadth-First Search and Dijkstra's algorithm are relatively good algorithms to investigate further, as unlike Depth-First Search they can find the shortest route between two points. During my research into existing algorithms, I considered ways I could program my own pathfinding algorithm by combining elements from the pathfinding algorithm I have researched to see whether it is possible to create a more efficient/quicker pathfinding algorithm. As an extension I may decide to attempt to program my own pathfinding algorithm, I will need to ask my expert later in the project on how I would go about doing this.

Links used for Research:

A Search:*
https://www.youtube.com/watch?v=-L-WgKMFuhE
https://www.slant.co/versus/11584/11585/~dijkstra-s-algorithm_vs_a-algorithm

Breadth-First Search:
https://www.quora.com/What-are-the-advantages-of-using-BFS-over-DFS-or-using-DFS-over-BFS-What-are-the-applications-and-downsides-of-each
- I also had some prior knowledge about this algorithm

Depth-First Search:
https://www.quora.com/What-are-the-advantages-of-using-BFS-over-DFS-or-using-DFS-over-BFS-What-are-the-applications-and-downsides-of-each
https://cs.stackexchange.com/questions/4914/why-cant-dfs-be-used-to-find-shortest-paths-in-unweighted-graphs
- I also had some prior knowledge about this algorithm

Dijkstra's Algorithm:
https://www.youtube.com/watch?v=GazC3A4OQTE
https://www.reddit.com/r/algorithms/comments/b06bdb/what_are_advantages_and_disadvantages_of/
https://www.slant.co/versus/11584/11585/~dijkstra-s-algorithm_vs_a-algorithm
https://www.researchgate.net/publication/296639227_Dijkstra's_algorithm_and_Google_maps

---

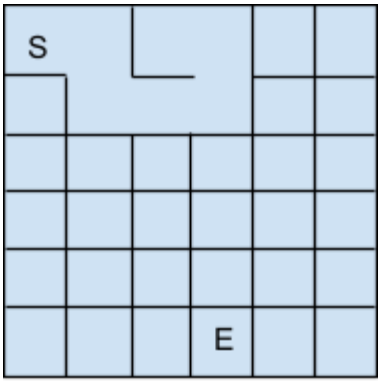
1.4.3.2 Maze Generation

Before algorithms can be run on a maze, first there needs to be a maze that they can be run on. After researching similar/existing programs, I found out that it would be beneficial and time saving to allow the user of my program to generate a maze randomly instead of allowing them to create one manually. Therefore, I researched into possible ways that I could generate a maze.

1.4.3.2.1 Recursive Backtracking

Recursive backtracking is an algorithm that can be used to generate a maze. It works by starting with a grid of squares which have walls between all of their adjacent squares. The algorithm starts by taking the start square as a parameter input and then randomly chooses to remove one of the walls between the start square and one of its adjacent squares. Using recursion this process is repeated but using the chosen adjacent square as the parameter input. If at any time the chosen adjacent square has been traversed (had its walls created) already then another adjacent square is chosen to have its walls between the current square removed. If a dead-end (where all the adjacent squares of a square have been traversed) is hit the program backtracks the recursion until it finds one square that had other possible adjacent squares it could have traversed, the program then randomly selects one and continues the recursion. If all possible adjacent squares have been traversed, it means that the program has backtracked the recursion to the start square and the generation of the maze is complete.

Example of recursive backtracking after some recursions:



Pros:

- Relatively easy to implement.
- Quick to implement.
- Easier to understand and explain than e.g.recursive division.
- It can always generate a maze with a solution.

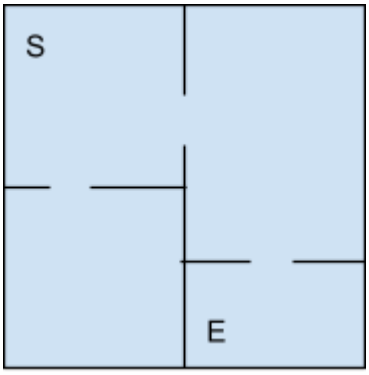
Cons:

- The run-time is not time efficient (not a big factor as I'm not comparing/measuring the time it takes to generate a maze, I will only measure the time it takes to run a pathfinding algorithm on the generated maze).
- As it uses a stack, a stack overflow may occur if the maze being generated is too big to have all the required information stored on the stack.

1.4.3.2.2 Recursive Division (or Divide-and-Conquer)

Recursive division involves starting with a grid of squares with no walls (only border walls), the rectangle(/square) of empty space (of squares) is referred to as a 'chamber'. Using vertical or horizontal lines (that act as walls) each chamber must be divided into more smaller chambers after each recursion. After every line that is drawn, a random location on the line (one square) is chosen to have the wall removed. The process of creating smaller chambers is recursively continued in each smaller chamber until chambers have either a width/height of one square.

Example of recursive division after some recursions:



Pros:

- It can always generate a maze with a solution.

Cons:

- Trickier to implement than e.g. recursive backtracking.
- Trickier to explain and understand than e.g. recursive backtracking.
- Not truly random as the maze is split using lines so there is no possibility of avoiding having straight lines with only one square that isn't a wall.
- The chambers are only connected via one possible square that can be traversed, this is really inefficient between large/main chambers.
- Uses recursion so could result in a stack overflow when generating a big maze.

Conclusion:

After researching possible maze generation algorithms I have decided to implement/use the recursive backtracking algorithm instead of the recursive division algorithm as I feel that it would be quicker to implement. Furthermore, recursive division has a lot more cons/disadvantages than recursive backtracking. For example, I don't like the idea of splitting up the maze into chambers as I don't believe that the maze generated is truly random then. Therefore, in my program I will use the recursive backtracking algorithm to randomly generate a maze.

Links used for Research:

https://en.wikipedia.org/wiki/Maze_generation_algorithm
https://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking
- I also had some prior knowledge/ideas about ways I could generate a maze
https://www.quora.com/What-are-the-advantages-and-disadvantages-of-a-backtracking-algorithm
https://www.quora.com/What-are-advantages-and-disadvantages-of-divide-and-conquer-approach

---

1.4.3 First interview

In order to gain further information about pathfinding algorithms and possible features that could be included in the program, I decided to interview Nyki (the expert for my NEA).

The interview with Nyki (the expert):
Me: in italics
Nyki: in bold italics

I am planning to investigate whether the arrival speed of vehicles between two points can be increased through investigating pathfinding algorithms. (A possible application of the investigation's results would be to aid ambulances in getting from point A to B quickly).

I have researched the following algorithms: A Search, Breadth-First Search, Depth-First Search and Dijkstra algorithm.*

1. *What is your opinion about the algorithms above, which do you think is the best to use for pathfinding and finding the shortest route between two points?*

   As both Dijkstras and breadth-first search algorithms are special cases of A* search then they are all suitable in terms of being admissible (guaranteed to find an optimal solution) and completeness as your graph should be finite (I assume!). The benefit of Dijkstra's over A* is in the efficiency - the stored queue will be shorter and therefore more efficient. In addition, breadth-first search will be more suitable than depth-first as it is complete (guaranteed to find the goal). My preferred one would be Dijkstra's even though its implementation is a little trickier.

2. *I have researched the pathfinding algorithms I have mentioned above. Do you recommend researching another pathfinding algorithm that exists? If so, what and why?*

   As the above ones provide admissibility they are probably the best ones to use - other pathfinding algorithms can be more efficient and simpler to program but don't always find the optimum solution and so wouldn't help with your problem.

3. *From the algorithms that I have researched, is there an algorithm which I shouldn't investigate further?*

   I think you could rule out depth-first search and focus on the other three.

4. *What makes a pathfinding algorithm good?*

   Admissibility and completeness are key for me, however efficiency would be important in the case of emergency vehicles mentioned above.

5. *Do you think data about e.g. how long a pathfinding algorithm took to be complete should be stored in some way in order to allow users of the program to access the data once the program was closed? If so in what format, (e.g. database, binary file or text file)?*

It would be helpful to store it so that the optimal route could be retrieved quicker in the future rather than having to run the full search again. Storing the data in a database/binary file would be more efficient as then multiple data types could be stored in one format (in a text file you would only be able to store strings). I would also store data in a database over a binary file, as a single record can be retrieved easily from a database using a primary key, whereas in a binary file it is tricky to retrieve specific data.

6. *a) What features would you want to see in a program that compares different pathfinding algorithms?*

   A comparison of run speed, the final route found and its weight (whether that be time taken or distance) and maybe memory used?

   b) Would you want the pathfinding algorithms to be visually represented/shown when they are finding the shortest possible path?

   This would be desirable for the person following the route found but not necessary in just comparing them.

7. *Is there anything else beneficial/important that I should know about pathfinding algorithms?*

   Some of them work for finite graphs and others work better on infinite graphs. Also, you may need to split your original graph into trees for some of them which adds complexity and will reduce the efficiency.

Conclusion:
(x) = the project objective they link to

From interviewing Nyki I have learnt that I should no longer further investigate Depth-First Search as it may not find necessary find a solution, therefore as Nyki said it would be more beneficial for me to focus on investigating A, Breadth-First Search and Dijkstra's algorithm **(1.1.)**. In Nyki's opinion the best of the algorithms out of the three I will be investigating is Dijsktra's algorithm, so I will focus on that algorithm a bit more and investigate whether that is the case. Furthermore, when I asked Nyki whether she recommends that I investigate another pathfinding algorithm she replied saying that there are similar and more efficient programs out there but they may not necessarily find a solution or the shortest route. Nyki said that in a program that compares pathfinding algorithms she would want to see features that allow the comparison of: the time taken for(/run speed of) the algorithm to find the shortest path, the final route found (the shortest route) along with its weight/distance and the memory used (which I didn't consider before) **(3.1.3.)**. I will consider keeping track of the memory used however it will likely be an extension task as Nyki said that she would only "maybe" wants to see a feature that compares the memory used **(3.1.3.4.)**. After asking Nyki what makes a pathfinding algorithm good she responded by saying that admissibility, completeness and efficiency are important; when thinking about which algorithm is the most suitable for getting from A point B in the shortest amount of time I will think about which of the pathfinding algorithms best fit that criteria. In addition, from interviewing Nyki I also found out that she believes that storing information such as how long it took a program to run on a specific maze is beneficial as it allows the information to be accessed again and quicker without needing to re-run the search. Nyki said that storing the information/data in a database and binary file is better when multiple data types need to be stored, and as Nyki said it would be beneficial to store information so that it can be*

accessed later I will likely need to store different data types. As a result I will not store the information in a text file. As Nyki said it is easier to retrieve specific data from a database than from a binary file, I will store information such as the time taken for an algorithm to run, in a database. ***(3.)****. As well as that, Nyki said that visually showing the pathfinding algorithm being run and the shortest route is not necessarily for comparing the algorithms however it is if someone is required to follow it. However, by showing how the pathfinding algorithm works visually could be beneficial, as then the path the pathfinding algorithm takes is shown and could be compared visually in some way, therefore as it isn't a necessity I will make this an extension task* ***(2.5.3.)****. As visually showing the path will only be an extension task, instead I could save the path (using coordinates of square) from the start square to the end square (which the user can look at/follow if needed) and output it* ***(3.1.2.3.)****. If I was to store the path then I wouldn't require to store the length of the shortest path as then it could be calculated by the user through counting the number of coordinates listed.*

1.4.4 Key requirements

- The program will be required to generate a maze
- The program will be required to be able to perform the following pathfinding algorithms on the maze:
    - A* Search
    - Breadth-First Search
    - Dijkstra's algorithm
- The program will be required to store data such as the time it took for the pathfinding algorithm to be complete and the shortest route in a database.

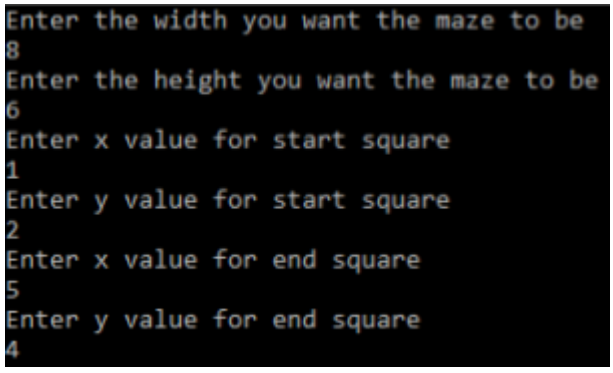
1.5 Further Research

1.5.1 Prototype

Prototype code (in appendix):

Information about prototype:

Example maze 1:



The prototype of the pathfinding program I will create, can randomly generate a maze (using recursive backtracking) and run an A* Search on it to find the shortest route from the start square to the end square. Currently the generated mazes can only have one solution. When the prototype program is run, it asks for the user to enter the width and the height they want the maze to have. After that it asks the user to enter coordinates for the start square and then the end square. The program then generates a maze and runs the A* Search code automatically on it displaying the: G Cost, H Cost, F Cost and coordinates of the square currently being visited until the end square is reached/found (it doesn't display the information for the end square) (as shown on the next page). If there is a solution the program outputs "Found end", if there isn't, the program outputs "No path found". The reason the prototype displays the information I have listed, is to show whether and how the program is running how it should.

Example maze 1, cost outputs:

```
GCost: 0
HCost: 36
FCost: 36
Coords: 1 2

GCost: 1
HCost: 35
FCost: 36
Coords: 1 3

GCost: 2
HCost: 34
FCost: 36
Coords: 1 4

GCost: 3
HCost: 33
FCost: 36
Coords: 1 5

GCost: 4
HCost: 32
FCost: 36
Coords: 2 5

GCost: 5
HCost: 31
FCost: 36
Coords: 3 5

GCost: 6
HCost: 30
FCost: 36
Coords: 3 6
```

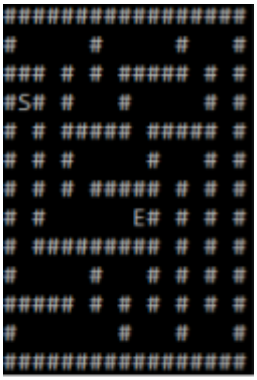
```
GCost: 9
HCost: 27
FCost: 36
Coords: 4 6

GCost: 10
HCost: 26
FCost: 36
Coords: 4 5

GCost: 11
HCost: 25
FCost: 36
Coords: 5 5

GCost: 12
HCost: 24
FCost: 36
Coords: 5 6

GCost: 13
HCost: 23
FCost: 36
Coords: 6 6

GCost: 14
HCost: 22
FCost: 36
Coords: 6 5

GCost: 15
HCost: 21
FCost: 36
Coords: 6 4
```

```
GCost: 16
HCost: 20
FCost: 36
Coords: 6 3

GCost: 25
HCost: 11
FCost: 36
Coords: 8 2

GCost: 26
HCost: 10
FCost: 36
Coords: 8 1

GCost: 27
HCost: 9
FCost: 36
Coords: 7 1

GCost: 28
HCost: 8
FCost: 36
Coords: 7 2

GCost: 29
HCost: 7
FCost: 36
Coords: 6 2

GCost: 30
HCost: 6
FCost: 36
Coords: 5 2
```

```
GCost: 31
HCost: 5
FCost: 36
Coords: 5 3

GCost: 32
HCost: 4
FCost: 36
Coords: 4 3

GCost: 17
HCost: 19
FCost: 36
Coords: 7 3

GCost: 18
HCost: 18
FCost: 36
Coords: 7 4

GCost: 19
HCost: 17
FCost: 36
Coords: 7 5

GCost: 20
HCost: 16
FCost: 36
Coords: 7 6

GCost: 21
HCost: 15
FCost: 36
Coords: 8 6
```

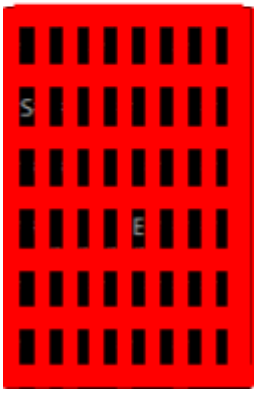
```
GCost: 22
HCost: 14
FCost: 36
Coords: 8 5

GCost: 23
HCost: 13
FCost: 36
Coords: 8 4

GCost: 24
HCost: 12
FCost: 36
Coords: 8 3

GCost: 33
HCost: 3
FCost: 36
Coords: 3 3

GCost: 34
HCost: 2
FCost: 36
Coords: 3 4

GCost: 35
HCost: 1
FCost: 36
Coords: 4 4

GCost: 36
HCost: 0
FCost: 36
Coords: 5 2
Found end
```

Example of a Generated Maze (Example maze 2):



Key:
(First Diagram:)
S = Start square/node
E = End square/node
 # = Wall
= Transversable square / space between two squares

(Second Diagram:)
= Transversable squares
= Possible wall locations (don't count as transversable squares)
==========



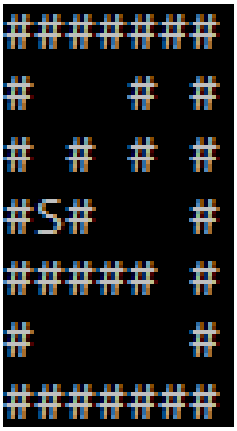
To generate a maze I double the numbers the user enters for the width & height of the maze and translate the start and end coordinates they enter by doubling them as well but also adding one. I do this so that I can store each wall between transversable squares as a square in a 2D array. An issue with this is that the generated maze may look misleading when outputted in the Console. For example, the maze on the left has a width of 8 and a height of 6 (of transversable squares).

Example of when a path cannot be found (Example maze 3):

Currently the user can enter coordinates for the start and end square/node which are outside the maze the user is generating as there is no error handling in place yet (in the prototype). Therefore, the maze created may not have a route from the start square to the end square as the end square is outside the maze (out of bounds). As a result, the prototype program outputs "No path found".

For example in this 3 by 3 maze, the start square has a valid location with the coordinates (1,2). Whereas, the end square has the coordinates (7,9) which are out of bounds. Therefore, there is no path from the start square to the end square.

```
Enter the width you want the maze to be
3
Enter the height you want the maze to be
3
Enter x value for start square
1
Enter y value for start square
2
Enter x value for end square
7
Enter y value for end square
9
```



```
GCost: 0
HCost: 0
FCost: 0
Coords: 1 2

GCost: 1
HCost: 1
FCost: 2
Coords: 1 1

GCost: 2
HCost: 2
FCost: 4
Coords: 2 1

GCost: 3
HCost: 3
FCost: 6
Coords: 2 2

GCost: 4
HCost: 4
FCost: 8
Coords: 3 2

GCost: 5
HCost: 5
FCost: 10
Coords: 3 1

No path found

GCost: 5
HCost: 5
FCost: 10
Coords: 3 1

No path found
```

From programming the prototype I learnt that I must plan/think ahead about things such as how the classes will inherit information from one another. Furthermore, I learnt that I must program the prototype so that it can easily have the other pathfinding algorithms run on it (once I code them) without needing to change the base code of the prototype much. I am not sure whether I have programmed the A* Search properly in my prototype as an A* Search uses a priority queue and I am not sure if I have successfully programmed one. In order to find out, I will do more research into A* Search algorithms myself or find out more information about A* Searches by talking to my expert/other experts.

After having done more research myself and asking different experts about the A* Search, I have realised/learnt that I didn't code it properly in my prototype. Therefore, I will need to modify/re-code it when programming my technical solution. In addition, in the A* Search code section, I will need to make each square store the previous square in its (Square) class in order to get the shortest path (of squares) from the start square to the end square. I learn (from my second interview with my expert Nyki, in the next section) that I should use an adjacency matrix or an adjacency list to store whether it is possible to traverse between two traversable squares. Through doing more research I found out that an adjacency list is used more for sparse graphs and an adjacency matrix is more used for dense graphs and for looking up values. As I will be looking up values in my program a lot, it will be more beneficial for me to use an adjacency matrix to store whether two traversable squares can be traversed between (so whether they are neighbours and whether there is a wall between them). An adjacency matrix can be implemented using a 2D array, and by storing a 1 if you can traverse between the two squares and 0 if you can't. Despite needing to re-program the A* Search, the recursive backtracking algorithm I programmed to generate the maze can be reused in my technical solution (although I might end up modifying a bit if I need to).

1.5.2 Second interview

For my second interview with my expert I divided up the questions into separate sections to make it easier to understand and answer. To make the questions clear I underlined them.

The second interview with Nyki (the expert):
Me: in italics (for interview) / normal (for things said after)
Nyki: in bold italics

---------- Section 1 ----------
For my prototype I focused on programming the generation of mazes and the A search algorithm.*

Currently in my prototype I double the numbers the user enters for the width & height to be and translate the start and end coordinates they enter by doubling them as well but also adding one. I do this so that I can store each wall between transversable squares as a square in a 2D array.

Do you think there is a more suitable way of storing whether there is a wall between two traversable squares than using a 2D array?

Section 1: I think that the way you are currently storing the walls and traversable squares in the 2D array is fine. You could consider using an adjacency list or matrix, as it would allow you to see whether you can traverse between two traversable squares, you can't if they are not neighbours and if there is a wall between them. It is easier and quicker to use an adjacency list or

matrix than using your current method. So it would be beneficial to incorporate an adjacency list or matrix into your program. However, you could continue using your original method.

---------- Section 2 ----------
Here I showed Nyki my draft objectives.

Do you think that I am missing any important objectives that should be included?
Do you think that there are objectives that aren't that relevant and should be removed?

Section 2: These are some very detailed and well written objectives - I can't see anything I would remove, the assignment of some as extensions is sensible so that you don't run out of time.

---------- Section 3 ----------

For my project NEA I will be attempting to create my own pathfinding algorithm, do you have any advice on how I would attempt to do this?

Section 3: If you look at the existing algorithms they are all based on storing traversed/visited nodes or edges and information about them. Each one uses a different way of making sure all possible paths have been covered whilst still maintaining efficiency. These are all attributes that you should consider but you will need to decide which ones are most important to your algorithm then use the original algorithm which also maximised this attribute for some ideas on where to start.

From doing this interview, I have decided to incorporate an adjacency list/matrix into my program **(2.2.)** as Nyki (my expert) said it would be beneficial for me to use one as they allow me to see whether it is possible to traverse between two traversable squares more easily and quickly than using my current method. Before I start programming my technical solution, I will need to research whether I should use an adjacency list or an adjacency matrix to store the information. Furthermore, I found out that the objectives for my program that will compare pathfinding algorithms are detailed and that there aren't any important things it is missing from their point of view. The expert's opinion about the objectives are important as they know what important features should be considered about pathfinding algorithms and programs that compare them (what I will be programming). From doing this second interview, I also learnt that when creating my own pathfinding algorithm I must consider an efficient way of storing the squares/nodes that have been traversed. In addition, I must think of a logical, creative and efficient way of traversing mazes while making sure that all paths are covered. Nyki said that I must choose an attribute that is the most important that I want my own pathfinding algorithm to have (e.g. time-efficient/checks all possible paths). Nyki, suggests that I work on one of the existing algorithms (e.g. A* Search, Breadth-Search, Dijkstra's algorithm) that is the best in terms of that attribute and attempt to improve it. The reason she suggested this is because it is extremely tricky (maybe impossible) to start from scratch and create a better/new pathfinding algorithm type.

1.6 Objectives

Below are the finalised objectives that I will need to make my program fulfill. By successfully completing these objectives my program will be able to successfully fulfill its purpose. The purpose of the program will be to allow for the comparison of pathfinding algorithms, and therefore allow me to use it to conduct my investigation on how the arrival speed of vehicles between two points can be made quicker through investigating pathfinding algorithms.

Any extension tasks are in italics.

Objectives:

1. Allow the user to choose from multiple pathfinding algorithms:
    - 1.1. The user can should be able to choose from the following pathfinding algorithms:
        - 1.1.1. A* Search
        - 1.1.2. Breadth-First Search
        - 1.1.3. Dijkstra's algorithm
        - *1.1.4. Extension: An algorithm that I create*
    - 1.2. The possible algorithms should be displayed clearly in the console *(or on an interface)* so that the user knows the possibilities.
2. Mazes:
    - 2.1. The user should be able to choose from different ways of creating/generating/accessing a maze to represent roads:
        - 2.1.1. The option to randomly generate a maze.
            - 2.1.1.1. User should enter a name for the maze
            - 2.1.1.2. The user should be asked to enter the dimensions of the maze they want to create.
            - 2.1.1.3. There will be a size limit so that the maze cannot have a size which causes the program to crash or not work.
                - 2.1.1.3.1. The maze will have a limit to its dimensions which will be:
                    - 2.1.1.3.1.1. A minimum of 2x2 (so that the maze being generated won't just consist of the start and end squares).
                    - 2.1.1.3.1.2. A maximum of 30 x 30 squares (to avoid a stack overflow error from occurring and causing the program to crash).
        - 2.1.2. The option to import previously saved mazes from a database.
            - 2.1.2.1. The user should be asked to enter the ID of the maze they want to import from a database.
            - 2.1.2.2. If the file doesn't exist, then the user should be able to go back to the previous options and choose to generate a maze or attempt to import another maze instead.
    - 2.2. Mazes will be represented as an adjacency matrix. But they will also be represented/displayed as a grid (2D array), where each coordinate of a grid will represent a node/square/wall in the maze.
    - 2.3. The user should be able to save mazes so that the same / a different algorithm can be run on the same maze.
        - 2.3.1. The maze and its information should be saved in a database, the maze's walls' location will be saved through using a list of 1s and 0s. The walls of the maze will be represented as a 1 and empty spaces/traversable squares as 0s.

2.3.2. Once a maze is saved the maze's ID should be outputted to the user so that they can load the maze

2.3.3. The maze should load in less than 10 seconds, in order to keep the program relatively efficient.

2.4. The user should be able to enter coordinates for the …(insert from below)... to determine their position within the maze (this would be before the maze is generated) (they cannot have the same coordinates):

2.4.1. Start square/node

2.4.2. Target/End square/node

2.5. *Extension: The maze will be visually displayed*

2.5.1. Extension: The maze itself, the start and target/end square/nodes and the walls will be visually displayed.

2.5.2. Extension: How the pathfinding algorithm traverses each square/node will be visually displayed.

2.5.3. Extension: The shortest route from the starting square/node to the end/target square/node will be visually displayed.

2.6. *Extension: The user can choose whether they want the maze to have multiple solutions or just one.*

3. Database:

3.1. The database should have three tables:

3.1.1. A table that stores information to load a maze, should store the following information:

3.1.1.1. mazeID

3.1.1.2. mazeName

3.1.1.3. mazeWidth

3.1.1.4. mazeHeight

3.1.1.5. startSquareX (the x value of the start square)

3.1.1.6. startSquareY (the y value of the start square)

3.1.1.7. endSquareX (the x value of the end square)

3.1.1.8. endSquareY (the y value of the end square)

3.1.1.9. wallStatus (will store information about whether each square in the maze is a wall or not, 1 if wall and 0 if not)

3.1.2. A table that maps each pathfinding algorithm to an associated ID, should store the following information:

3.1.2.1. algorithmID

3.1.2.2. algorithmName

3.1.3. A table that stores information about the maze when an algorithm was run on it should store the following information:

3.1.3.1. processID

3.1.3.2. mazeID

3.1.3.3. algorithmID

3.1.3.4. timeTakenToRun - The time taken for the pathfinding algorithm to be complete (in milliseconds).

3.1.3.5. pathToEnd - The shortest route (using coordinates of squares), from the starting square/node to the end/target square/node.

3.1.3.6. numberOfSquaresTraversed, which is the number of squares explored in the grid (which represents a maze) before the end/target square/node was found.

3.1.3.7. Extension: The amount of memory used before the pathfinding algorithm has finished running.

3.2. The information should be stored in the following format:

3.2.1. In a database

3.3. The information/content should be accessible from:

3.3.1. Within the program (only the data: 3.1.3.4. , 3.1.3.5. , 3.1.3.6.)

3.3.2. From the database it is stored in

4. Malicious:

4.1. The program should have an instructions screen that summarises important information on how to use the program.

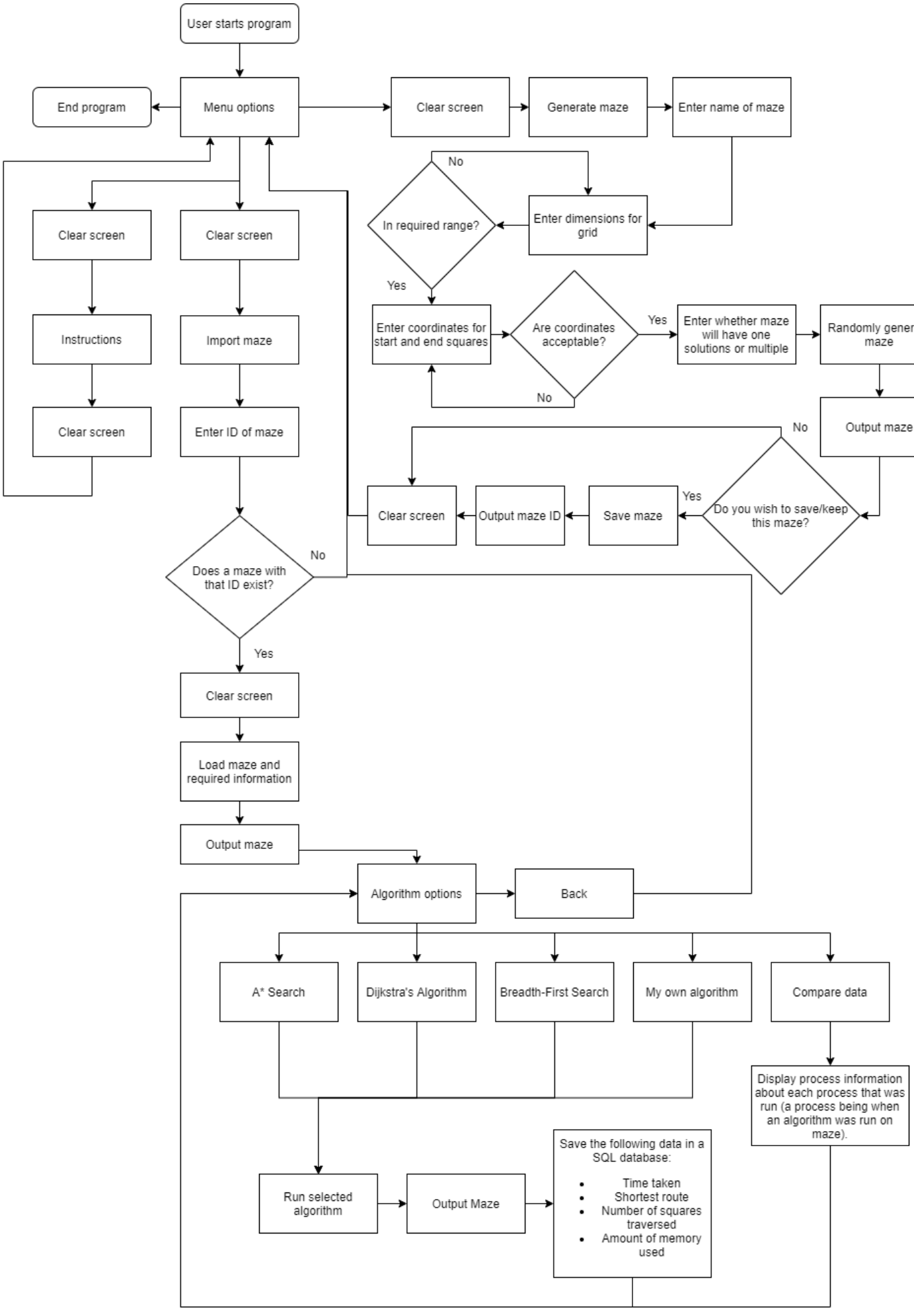
1.7 Modelling

1.7.1 Flowchart showing Program Structure

Before programming my program that allows pathfinding algorithms to be compared, I have decided to plan the structure of the program using a flowchart. By planning the structure using a flowchart model it allows me to have a basic idea of what the program will do and its basic structure making it easier as well as quicker to program.

After creating the flowchart model, I now have a rough idea of what inputs the user is required to input and what the program should output as a response. Furthermore, I also now have an idea about the different option screens my program will require, e.g. Main Menu, Instructions, Generate Maze and Import Maze.

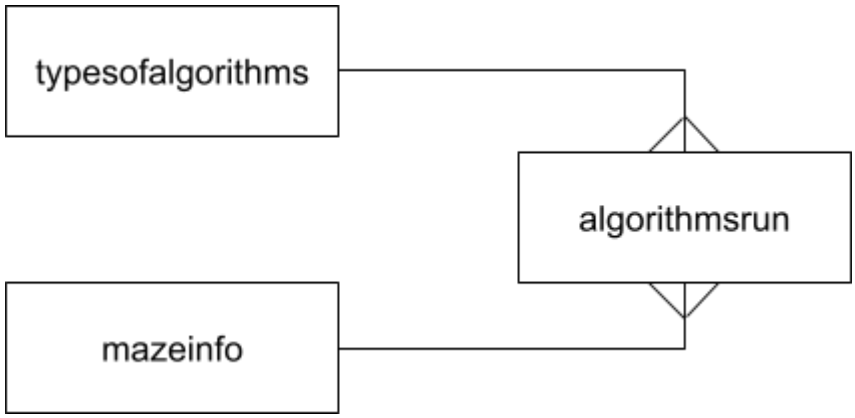
The flowchart model for my program can be found on the next page, it shows what the structure of the program will be like and how it will work.


User starts program
Menu options
End program
Clear screen
Generate maze
Enter name of maze
Enter dimensions for grid
In required range?
No
Yes
Clear screen
Instructions
Clear screen
Clear screen
Import maze
Enter ID of maze
Enter coordinates for start and end squares
Are coordinates acceptable?
No
Yes
Enter whether maze will have one solutions or multiple
Randomly generate maze
Output maze
No
Do you wish to save/keep this maze?
Yes
Save maze
Output maze ID
Clear screen
Does a maze with that ID exist?
No
Yes
Clear screen
Load maze and required information
Output maze
Algorithm options
Back
A* Search
Dijkstra's Algorithm
Breadth-First Search
My own algorithm
Compare data
Display process information about each process that was run (a process being when an algorithm was run on maze).
Run selected algorithm
Output Maze
Save the following data in a SQL database:
Time taken
Shortest route
Number of squares traversed
Amount of memory used

1.7.2 Database Structure

Before choosing what format the database will be created in and actually creating it, I must decide on its structure to make sure that once I create the database it is in normalised form. The reason it must be in normalised form is to avoid there being repeating groups, to make all non-key attributes depend on the key, the whole key and nothing but the key. Furthermore, having a database in a normalised form means that things like data inconsistency and data redundancy are avoided. The following entity relationship diagram (ER diagram) shows what tables the database will have and how they will be linked using foreign keys and primary keys.

Entity Relationship Diagram for pathfindingdatabase:



Entity Description for pathfindingdatabase:

pathfindingdatabase:

mazeinfo(mazeID, mazeName, mazeWidth, mazeHeight, startSquareX, startSquareY, endSquareX, endSquareY, wallStatus)
typesofalgorithms(algorithmID, algorithmName)
algorithmsrun(processID, *mazeID*, *algorithmID*, pathToEnd, timeTakenToRun, numberOfSquaresTraversed)

---

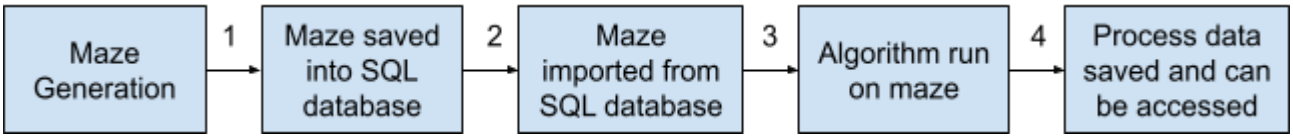
2 Design

2.1 High Level Overview

2.1.2 Interaction of Key Features

The main purpose of the program is to allow the user to generate mazes, run algorithms on the generated mazes and compare data gathered from processes (running an algorithm on a maze). The data gained from a process is needed in order to conduct my investigation on “How the arrival speed of vehicles between two points, can be made quicker through investigating pathfinding algorithms and seeing which can find the shortest route in the shortest amount of time?”. The key features of the program are the features that are required for the program to fulfill its main purpose in order to allow me to conduct my investigation.

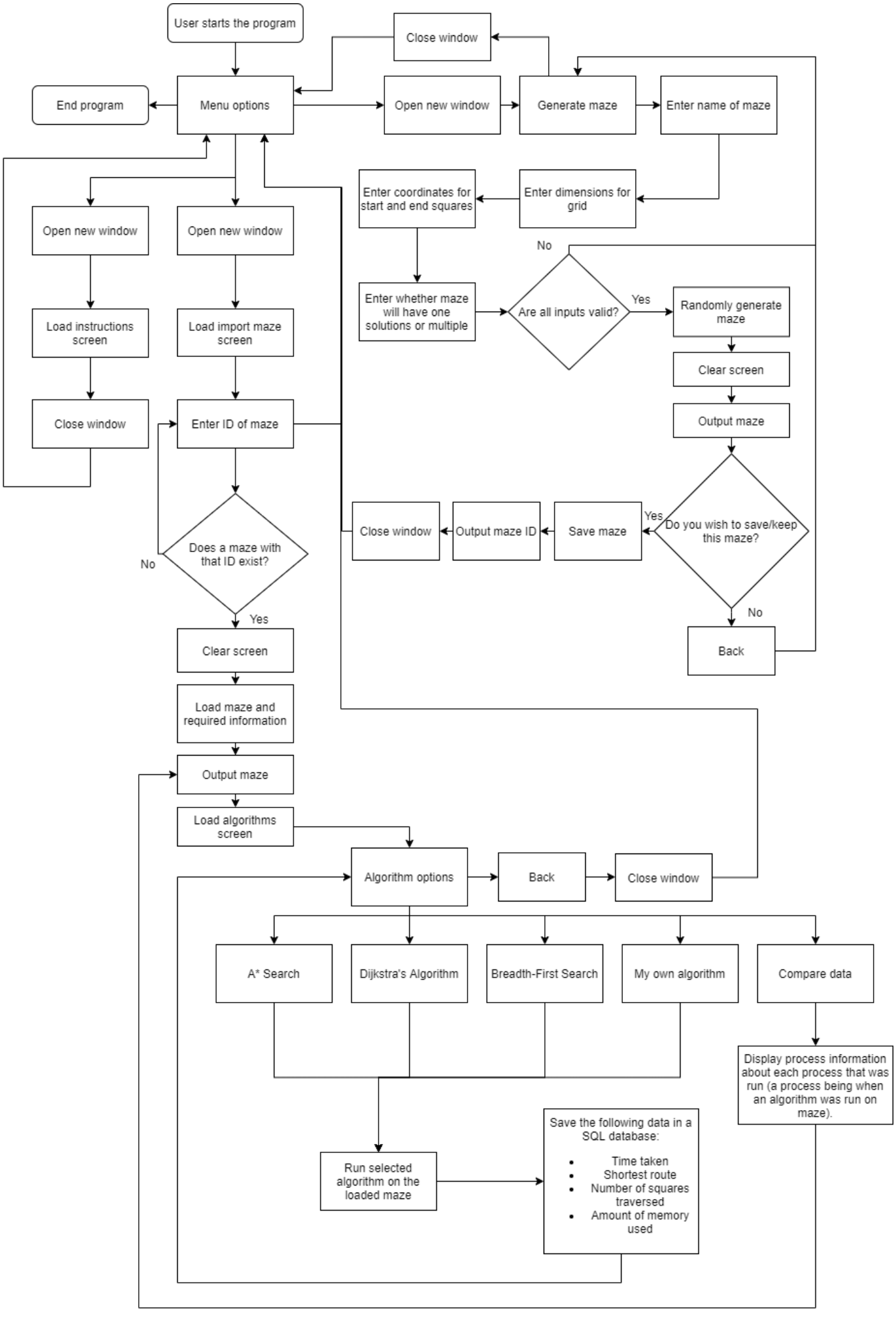
The key features of the program interact and depend on each other a lot, e.g. without the maze generation feature, there would be no mazes to import therefore there would be no maze to run algorithms on, so the program wouldn’t work. The diagram below shows the key features of my program (in blue boxes) and conveys how if any didn’t exist/work the ones after that follow (to the right) wouldn’t work either. So the key features interact a lot as the process that a key features does is needed to be done to let the next process work.



1) A maze is generated in order to let the program store the maze
2) The saved maze is imported by the program
3) The imported maze has an algorithm run on it
4) Data about the algorithm being run on the maze (e.g. the time it took) is saved in the SQL database and can be accessed by the user within the program(/from SQL database) to compare data

2.1.2 Flowchart

On the next page is the finalised version of the flowchart of my program, as it is only an abstraction it isn’t extremely detailed so it doesn’t show every process that occurs. Previously for my prototype I created a flowchart (model) in order to get a rough idea of how the program will work, before starting to work on my technical solution I updated the flowchart I had previously created. Its purpose was to give me a good idea of how my program is likely to be structured (if I created interfaces for my program). By giving me an idea about the structure meant that I could program my program easier and quicker than if I hadn’t planned the structure. In addition, by having an idea of the structure of my program it meant that I had some ideas about what classes, window forms/interfaces, error handling, etc I would require.


User starts the program
Close window
End program
Menu options
Open new window
Generate maze
Enter name of maze
Enter coordinates for start and end squares
Enter dimensions for grid
Open new window
Open new window
No
Enter whether maze will have one solutions or multiple
Are all inputs valid?
Yes
Randomly generate maze
Load instructions screen
Load import maze screen
Clear screen
Close window
Enter ID of maze
Output maze
Does a maze with that ID exist?
No
Yes
Close window
Output maze ID
Save maze
Yes
Do you wish to save/keep this maze?
No
Back
Clear screen
Load maze and required information
Output maze
Load algorithms screen
Algorithm options
Back
Close window
A* Search
Dijkstra's Algorithm
Breadth-First Search
My own algorithm
Compare data
Display process information about each process that was run (a process being when an algorithm was run on maze).
Run selected algorithm on the loaded maze
Save the following data in a SQL database:
Time taken
Shortest route
Number of squares traversed
Amount of memory used

2.2 Choice of programming language

Before starting to program my pathfinding comparison program, I knew that I had to choose a high-level language that I was already familiar with to avoid needing to spend a lot of time learning it. Despite knowing a bit of Python I knew that I was more confident and familiar with Visual Basic (VB) so I decided to program the program using VB. Within VB I also used windows forms as it allowed me to create a GUI (graphic user interface) that the user could interact with easier than if I just used the console to await the user's inputs and output text. Therefore, as I choose to use a GUI, it means the user can only interact with the program using it.

As I am familiar with basic SQL commands, I knew that if I choose to store data in an SQL database then I could successfully create and manipulate data in it. After researching possible ways to store the required data, I did choose to use an SQL database. Due to my knowledge of SQL commands, it meant that I could successfully as well as quickly create a database, tables and program SQL queries within VB.

2.3 Design Techniques

2.3.1 Algorithms

2.3.1.1 Recursive Backtracking

In order to generate a maze randomly, I choose to program the recursive backtracking algorithm as I knew that it would be quicker to program than e.g. the recursive division algorithm. Furthermore, it allowed me to generate mazes more randomly than if I used recursive division.

In order to create/generate a maze randomly, using recursive backtracking, I created a subroutine ("GenerateWalls") which I then implemented to perform recursive backtracking on a maze when it is in a state where every possible wall is a wall; the subroutine uses recursion (as it calls upon itself). I put the subroutine in a class called "Maze" which encapsulates multiple procedures that can be run (to create)/on the maze.

Every time the subroutine is called upon the following happens (the subroutine initially has the start square inputted as the parameter input):

- An array (tempVisitArray) of length 4 is created which stores the boolean state False at each index
- The tempSquare (the square passed along in the parameter) is marked as traversed
- The following happens 4 times
    - A random number between 1 and 4 (inclusive) is generated until in the array tempVisitArray at the index of the generated number the element stored is False
    - If the number generated is 1
        - If a square exists above the current square (tempSquare) then the wall between the two is removed and the square above is marked as traversed
        - The subroutine then calls upon itself and passes the square above as the parameter input

        - If the number generated is 2
            - If a square exists to the right of the current square (tempSquare) then the wall between the two is removed and the square to the right is marked as traversed
            - The subroutine then calls upon itself and passes the square to the right as the parameter input
        - If the number generated is 3
            - If a square exists below the current square (tempSquare) then the wall between the two is removed and the square below is marked as traversed
            - The subroutine then calls upon itself and passes the square below as the parameter input
        - If the number generated is 4
            - If a square exists to the left of the current square (tempSquare) then the wall between the two is removed and the square to the left is marked as traversed
            - The subroutine then calls upon itself and passes the square to the left as the parameter input
        - The element in tempVisitArray at the index of the randomly generated number is made to store the boolean value True
- Next

2.3.1.2 A* Search

My investigation requires me to compare different pathfinding algorithms, after doing research into possible pathfinding algorithms to compare, I chose the A* Search **(1.1.1.)** to be one of them. Therefore, the reason I had to program the A* Search algorithm is so that I could run it on a maze, save data about the process e.g. the time taken to run the algorithm and compare the process to other processes that were run (which may have used different algorithms).

It took me multiple attempts to implement the A* Search, but I did manage to successfully implement it. The procedures needed to run the A* Search are in the “Algorithm” class with other procedures which run other algorithms such as Dijkstra’s Algorithm.

In order to clearly explain how the A* Search algorithm works within my program, I have explained/modelled it using both a step-by-step description and a flowchart. In addition to that, I have also modelled a worked example.

<u>Below are the steps that my program follows to run the A* Search on a maze:</u>

The steps that occur in the function that is run before the A* Search procedure can be run:

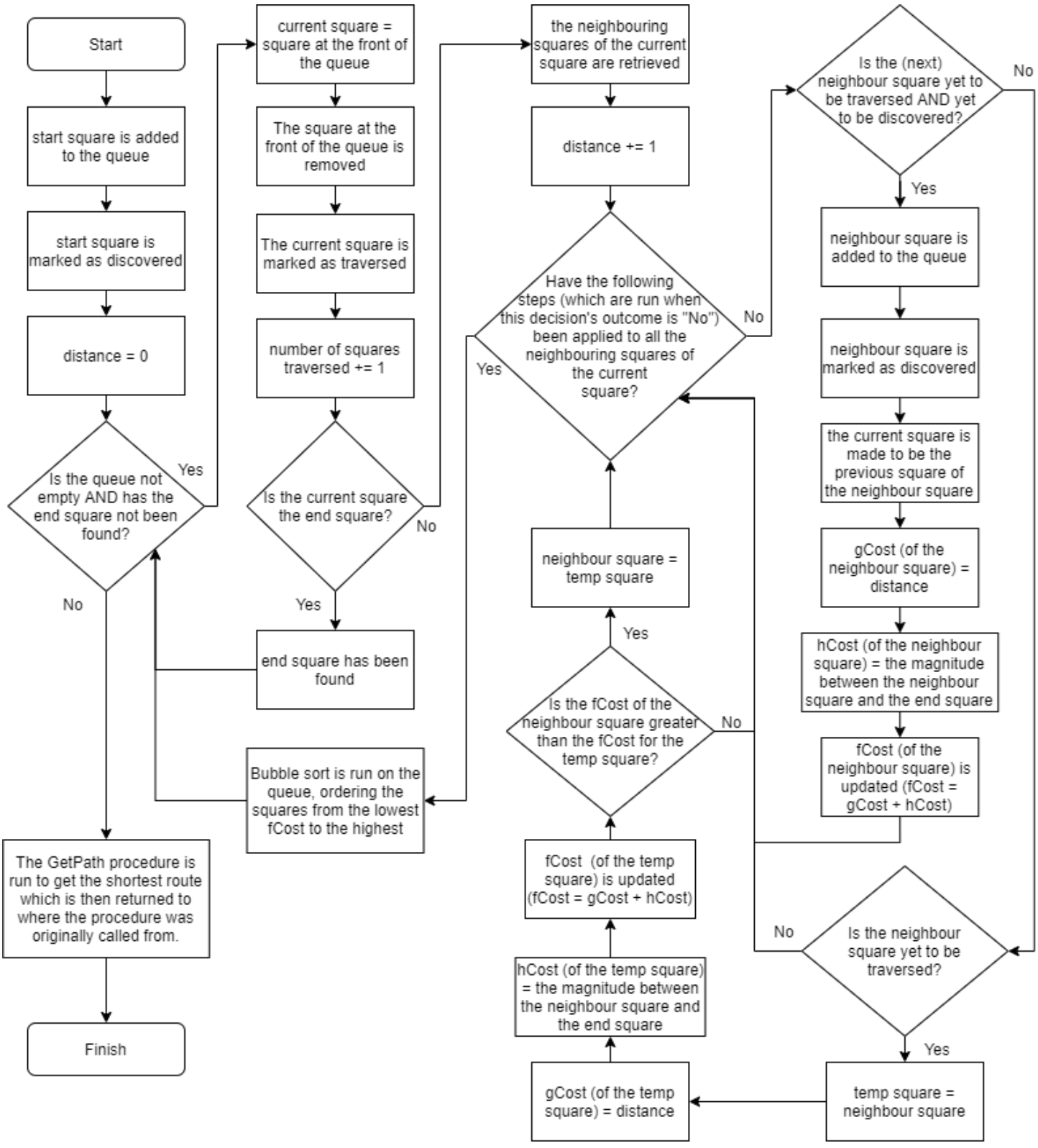
- The start square is added to the queue
- The start square is marked as discovered
- The A* Search subroutine is called upon and as the distance from the start is 0, it is inputted as the parameter value
- Once the A* Search procedure has been run, the path from the start to the end square is retrieved by running the GetPath function, the path is returned to the place where the function to start the A* Search was called upon.

The subroutine that runs the A* Search does the following:

- The program checks whether the queue is empty and if the end has been found, IF both are FALSE then

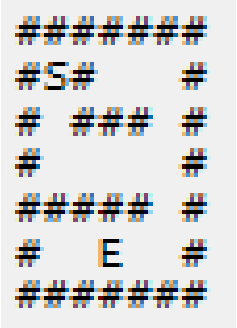
- The value at the front of the queue is retrieved to be the current square and removed from the queue
- The current square is marked as traversed
- The counter that counts the number of squares traversed increments by one
- IF the current square is the end square then
    - The end is marked to have been found
- ELSE
    - The neighbours of the current square are retrieved
    - The distance from the start counter increments by one
    - FOR EACH of the current square's neighbours the following is done:
        - IF both the traversed state and discovered state of the neighbour square is FALSE then
            - The neighbour square is added to the queue
            - The neighbour square is marked as discovered
            - The previous square of the neighbour square is made to be the current square
            - The gCost of the neighbour square is made to be the current distance from the start
            - The hCost of the neighbour square is worked out using pythagoras on its coordinates and the end square's coordinates (however the result is not square rooted it) (the fCost is automatically updated)
        - ELSEIF (if) the traversed status of the neighbour square is FALSE then
            - If the neighbour square would have a lower fCost if the current square was its previous square, then the current square is made to be the previous square of the neighbour square and the costs are updated accordingly. Otherwise nothing is done.
    - NEXT
    - A bubble sort is run on the queue, ordering the contents from the lowest fCost to the highest
    - The subroutine calls upon itself, the distance from the start is passed as a parameter input (recursion occurs)

Below is the flowchart conveying how my program runs the A* Search on a maze:



Below is a worked example of how the A* Search runs on a maze (“Example Maze”):

“Example Maze”:



queueOfSquaresToVisit (the numbers show the step number; they each match to a table below):

1) S(1,1)
2) (1,2)
3) (2,2)
4) (3,2)
5) (3,3), (3,1)
6) E(2,3) (3,1)
7) (3,1)

The shortest path (of “Example Maze”) obtained by the GetPath function after the A* Search procedure is run:

S(1,1), (1,2), (2,2), (3,2), (3,3), E(2,3)

1)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
fCost	5	0	0	0	0	0	0	0	0
hCost	5	0	0	0	0	0	0	0	0
gCost	0	0	0	0	0	0	0	0	0
Previous Square	-	-	-	-	-	-	-	-	-

2)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
fCost	5	3	0	0	0	0	0	0	0
hCost	5	2	0	0	0	0	0	0	0
gCost	0	1	0	0	0	0	0	0	0
Previous Square	-	S(1,1)	-	-	-	-	-	-	-

3)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
fCost	5	3	0	0	3	0	0	0	0
hCost	5	2	0	0	1	0	0	0	0
gCost	0	1	0	0	2	0	0	0	0
Previous Square	-	S(1,1)	-	-	(1,2)	-	-	-	-

4)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
fCost	5	3	0	0	3	0	0	5	0
hCost	5	2	0	0	1	0	0	2	0
gCost	0	1	0	0	2	0	0	3	0
Previous Square	-	S(1,1)	-	-	(1,2)	-	-	(2,2)	-

5)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE
fCost	5	3	0	0	3	0	9	5	5
hCost	5	2	0	0	1	0	5	2	1
gCost	0	1	0	0	2	0	4	3	4
Previous Square	-	S(1,1)	-	-	(1,2)	-	(3,2)	(2,2)	(3,2)

6)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
fCost	5	3	0	0	3	5	9	5	5
hCost	5	2	0	0	1	0	5	2	1
gCost	0	1	0	0	2	5	4	3	4
Previous Square	-	S(1,1)	-	-	(1,2)	(3,3)	(3,2)	(2,2)	(3,2)

7)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
fCost	5	3	0	0	3	5	9	5	5
hCost	5	2	0	0	1	0	5	2	1
gCost	0	1	0	0	2	5	4	3	4
Previous Square	-	S(1,1)	-	-	(1,2)	(3,3)	(3,2)	(2,2)	(3,2)

2.3.1.3 Breadth-First Search

My investigation requires me to compare different pathfinding algorithms, after doing research into possible pathfinding algorithms to compare, I chose Breadth-First Search **(1.1.2.)** to be one of them. As I made it a requirement to program the Breadth-First Search algorithm in the objectives it meant that I had to program it. The Breadth-First Search algorithm can be run on a maze, the data about the process can be saved (e.g. the time taken to run the algorithm) and allow the process to be compared with other processes that were run (which may have used different algorithms).

In order to clearly explain how the Breadth-First Search algorithm works within my program, I have explained/modelled it using both a step-by-step description and a flowchart. In addition to that, I have also modelled a worked example.

Below are the steps that my program follows to run the Breadth-First Search on a maze:

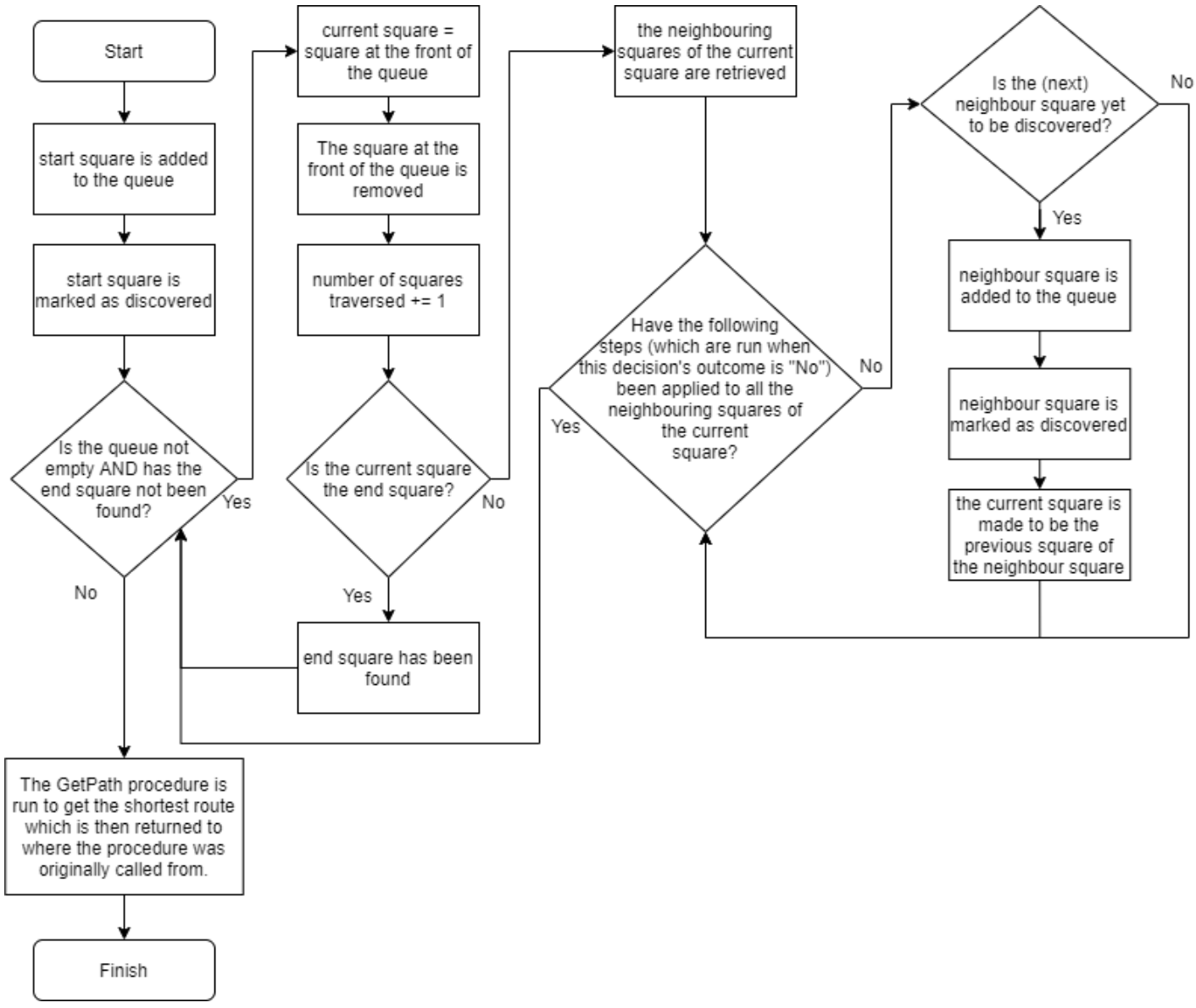
The steps that occur in the function that is run before the procedure that runs the Breadth-First Search algorithm can be run:

- The start square is added to the queue
- The start square is marked as discovered
- The Breadth-First Search subroutine is called upon
- Once the Breadth-First Search procedure has been run, the path from the start to the end square is retrieved by running the GetPath function, the path is returned to the place where the function to start the Breadth-First Search was called upon.

The subroutine that runs the Breadth-First Search does the following:

- The program checks whether the queue is empty and if the end has been found, IF both are FALSE then
    - The value at the front of the queue is retrieved to be the current square and removed from the queue
    - The counter that counts the number of squares traversed increments by one
    - IF the current square is the end square then
        - The end is marked to have been found
    - ELSE
        - The neighbours of the current square are retrieved
        - FOR EACH of the current square's neighbours the following is done:
            - IF the discovered state of the neighbour square is FALSE then
                - The neighbour square is added to the queue
                - The neighbour square is marked as discovered
                - The previous square of the neighbour square is made to be the current square
        - NEXT
        - The subroutine calls upon itself (recursion occurs)

Below is the flowchart conveying how my program runs the Breadth-First Search on a maze:



Below is a worked example of how the Breadth-First Search runs on a maze ("Example Maze"):

"Example Maze":

```
#######
#S#   #
# ### #
#     #
##### #
#  E  #
#######
```

queueOfSquaresToVisit (the numbers show the step number; they each match to a table below):

1) S(1,1)
2) (1,2)
3) (2,2)
4) (3,2)
5) (3,1), (3,3)
6) (3,3), (2,1)
7) (2,1), E(2,3)
8) E(2,3)
9)

The shortest path (of "Example Maze") obtained by the GetPath function after the Breadth-First Search procedure is run:

S(1,1), (1,2), (2,2), (3,2), (3,3), E(2,3)

1)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Previous Square	-	-	-	-	-	-	-	-	-

2)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Previous Square	-	S(1,1)	-	-	-	-	-	-	-

3)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
Previous Square	-	S(1,1)	-	-	(1,2)	-	-	-	-

4)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
Previous Square	-	S(1,1)	-	-	(1,2)	-	-	(2,2)	-

5)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE
Previous Square	-	S(1,1)	-	-	(1,2)	-	(3,2)	(2,2)	(3,2)

6)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
Previous Square	-	S(1,1)	-	(3,1)	(1,2)	-	(3,2)	(2,2)	(3,2)

7)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Previous Square	-	S(1,1)	-	(3,1)	(1,2)	(3,3)	(3,2)	(2,2)	(3,2)

8)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Previous Square	-	S(1,1)	-	(3,1)	(1,2)	(3,3)	(3,2)	(2,2)	(3,2)

9)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Previous Square	-	S(1,1)	-	(3,1)	(1,2)	(3,3)	(3,2)	(2,2)	(3,2)

2.3.1.4 Dijkstra's Algorithm

My investigation requires me to compare different pathfinding algorithms, after doing research into possible pathfinding algorithms to compare, I chose Dijkstra's Algorithm **(1.1.3.)** to be one of them. I use Dijkstra's Algorithm by running it on a maze and measuring/keeping track of data e.g. the number of squares traversed which then allows me to compare data for my investigation.

In order to clearly explain how Dijkstra's Algorithm works within my program, I have explained/modelled it using both a step-by-step description and a flowchart. In addition to that, I have also modelled a worked example.

<u>Below are the steps that my program follows to run Dijkstra's Algorithm on a maze:</u>

The steps that occur in the function that is run before the procedure that runs Dijkstra's Algorithm can be run:

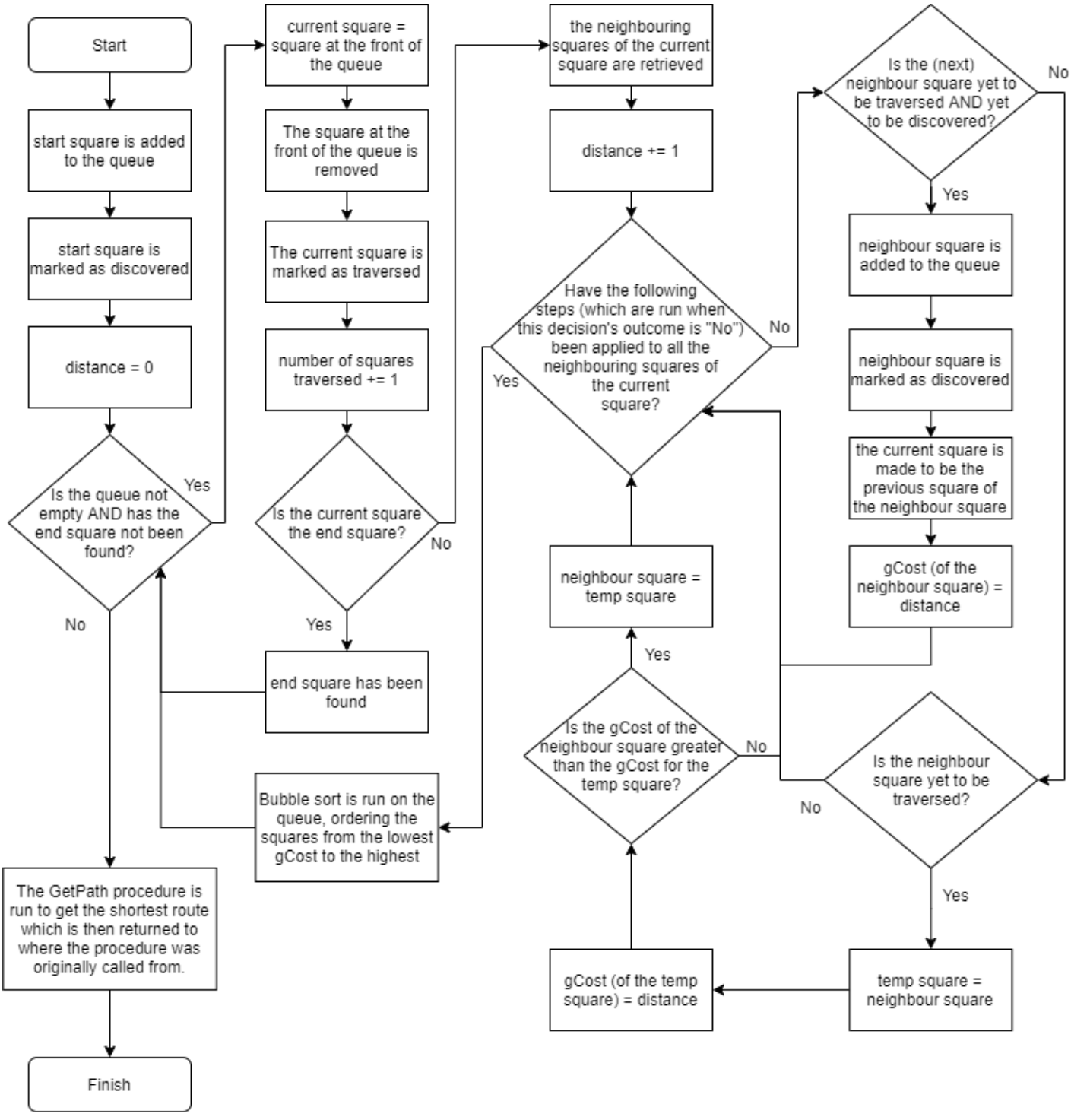
- The start square is added to the queue
- The start square is marked as discovered
- The Dijkstra's Algorithm subroutine is called upon and as the distance from the start is 0, it is inputted as the parameter value
- Once the Dijkstras's Algorithm procedure has been run, the path from the start to the end square is retrieved by running the GetPath function, the path is returned to the place where the function to start Dijkstra's Algorithm was called upon.

The subroutine that runs Dijkstra's Algorithm does the following:

- The program checks whether the queue is empty and if the end has been found, IF both are FALSE then
    - The value at the front of the queue is retrieved to be the current square and removed from the queue
    - The current square is marked as traversed
    - The counter that counts the number of squares traversed increments by one
    - IF the current square is the end square then
        - The end is marked to have been found
    - ELSE
        - The neighbours of the current square are retrieved
        - The distance from the start counter increments by one
        - FOR EACH of the current square's neighbours the following is done:
            - IF both the traversed state and discovered state of the neighbour square is FALSE then
                - The neighbour square is added to the queue
                - The neighbour square is marked as discovered
                - The previous square of the neighbour square is made to be the current square
                - The gCost of the neighbour square is made to be the current distance from the start
            - ELSEIF (if) the traversed status of the neighbour square is FALSE then
                - If the neighbour square would have a lower gCost if the current square was its previous square, then the current square is made to be the previous square and the gCost is updated accordingly. Otherwise nothing is done.

- NEXT
- A bubble sort is run on the queue, ordering the contents from the lowest gCost to the highest
- The subroutine calls upon itself, the distance from the start is passed as a parameter input (recursion occurs)

<u>Below is the flowchart conveying how my program runs Dijkstra's Algorithm on a maze:</u>



Below is a worked example of how Dijkstra's Algorithm runs on a maze ("Example Maze"):

"Example Maze":

```
#######
#S#     #
# ### #
#       #
##### #
#   E   #
#######
```

queueOfSquaresToVisit (the numbers show the step number; they each match to a table below):

1) S(1,1)
2) (1,2)
3) (2,2)
4) (3,2)
5) (3,1), (3,3)
6) (3,3), (2,1)
7) (2,1), E(2,3)
8) E(2,3)
9)

The shortest path (of "Example Maze") obtained by the GetPath function after the Dijkstra's Algorithm procedure is run:

S(1,1), (1,2), (2,2), (3,2), (3,3), E(2,3)

1)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
gCost	0	∞	∞	∞	∞	∞	∞	∞	∞
Previous Square	-	-	-	-	-	-	-	-	-

2)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
gCost	0	1	∞	∞	∞	∞	∞	∞	∞
Previous Square	-	S(1,1)	-	-	-	-	-	-	-

3)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
gCost	0	1	∞	∞	2	∞	∞	∞	∞
Previous Square	-	S(1,1)	-	-	(1,2)	-	-	-	-

4)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
gCost	0	1	∞	∞	2	∞	∞	3	∞
Previous Square	-	S(1,1)	-	-	(1,2)	-	-	(2,2)	-

5)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE
gCost	0	1	∞	∞	2	∞	4	3	4
Previous Square	-	S(1,1)	-	-	(1,2)	-	(3,2)	(2,2)	(3,2)

6)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE
Discovered	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
gCost	0	1	∞	5	2	∞	4	3	4
Previous Square	-	S(1,1)	-	(3,1)	(1,2)	-	(3,2)	(2,2)	(3,2)

7)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE
Discovered	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
gCost	0	1	∞	5	2	5	4	3	4
Previous Square	-	S(1,1)	-	(3,1)	(1,2)	(3,3)	(3,2)	(2,2)	(3,2)

8)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE
Discovered	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
gCost	0	1	∞	5	2	5	4	3	4
Previous Square	-	S(1,1)	-	(3,1)	(1,2)	(3,3)	(3,2)	(2,2)	(3,2)

9)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Traversed	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE
Discovered	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
gCost	0	1	∞	5	2	5	4	3	4
Previous Square	-	S(1,1)	-	(3,1)	(1,2)	(3,3)	(3,2)	(2,2)	(3,2)

2.3.1.5 My Algorithm

Despite that it was only an extension objective to create my own algorithm **(1.1.4.)**, I still successfully fulfilled the objective. The algorithm I have created can successfully find the shortest path from the start square to the end square, however it does traverse more squares than the A* Search so it's not the most efficient algorithm. The algorithm I have created is similar to Depth-First Search (more information can be found in the analysis section) as it explores each route/path within the maze fully and only then goes to the next one, however it uses a queue instead of a stack (which the Depth-First Search uses). My algorithm can be run on a maze to allow data to be measured e.g. the number of squares traversed, this data is needed to allow me to compare data for my investigation.

In order to clearly explain how my algorithm works within my program, I have explained/modelled it using both a step-by-step description and a flowchart. In addition to that, I have also modelled a worked example.

<u>Below are the steps that my program follows to run my algorithm on a maze:</u>

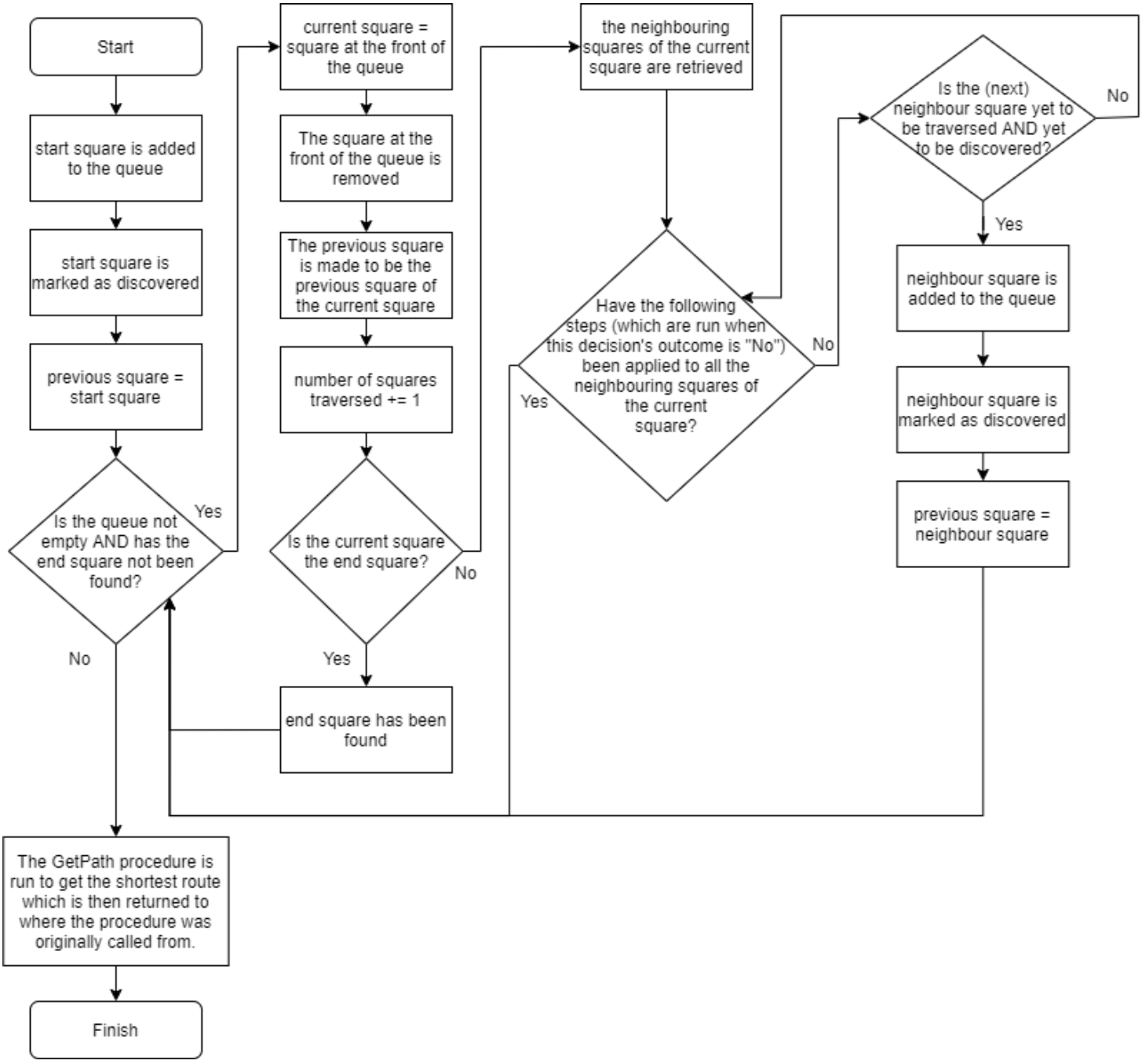
The steps that occur in the function that is run before the procedure for my algorithm can be run:

- The start square is added to the queue
- The start square is marked as discovered
- The subroutine that runs my algorithm is called upon, the start square is passed along as the parameter input (to be the previous square)
- Once the procedure that runs my algorithm has been run, the path from the start to the end square is retrieved by running the GetPath function, the path is returned to the place where the function to start my algorithm was called upon.

The subroutine that runs my algorithm does the following:

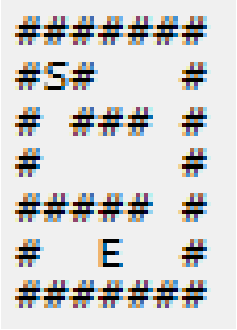
- The program checks whether the queue is empty and if the end has been found, IF both are FALSE then
    - The value at the front of the queue is retrieved to be the current square and removed from the queue
    - The previous square (the parameter input) is made to be the previous square of the current square
    - The counter that counts the number of squares traversed increments by one
    - IF the current square is the end square then
        - The end is marked to have been found
    - ELSE
        - The neighbours of the current square are retrieved
        - FOR EACH of the current square's neighbours the following is done:
            - IF the discovered state of the neighbour square is FALSE then
                - The neighbour square is added to the queue
                - The neighbour square is marked as discovered
                - The subroutine calls upon itself, the current square is passed as a parameter input to be the previous square (recursion occurs)
        - NEXT

Below is the flowchart conveying how my program runs my algorithm on a maze:



Below is a worked example of how my algorithm runs on a maze (“Example Maze”):

“Example Maze”:



queueOfSquaresToVisit (the numbers show the step number; they each match to a table below):

1) S(1,1)
2) (1,2)
3) (2,2)
4) (3,2)
5) (3,1)
6) (2,1)
7) (3,2)
8) E(2,3)
9)

The shortest path (of “Example Maze”) obtained by the GetPath function after the procedure my algorithm is run:

S(1,1), (1,2), (2,2), (3,2), (3,3), E(2,3)

1)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Previous Square	-	-	-	-	-	-	-	-	-

2)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Previous Square	-	S(1,1)	-	-	-	-	-	-	-

3)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
Previous Square	-	S(1,1)	-	-	(1,2)	-	-	-	-

4)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
Previous Square	-	S(1,1)	-	-	(1,2)	-	-	(2,2)	-

5)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
Previous Square	-	S(1,1)	-	-	(1,2)	-	(3,2)	(2,2)	-

6)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE
Previous Square	-	S(1,1)	-	(3,1)	(1,2)	-	(3,2)	(2,2)	-

7)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
Previous Square	-	S(1,1)	-	(3,1)	(1,2)	-	(3,2)	(2,2)	(3,2)

8)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Previous Square	-	S(1,1)	-	(3,1)	(1,2)	(3,3)	(3,2)	(2,2)	(3,2)

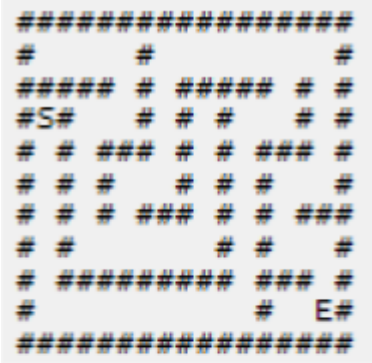
9)

Square	S(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	E(2,3)	(3,1)	(3,2)	(3,3)
Discovered	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Previous Square	-	S(1,1)	-	(3,1)	(1,2)	(3,3)	(3,2)	(2,2)	(3,2)

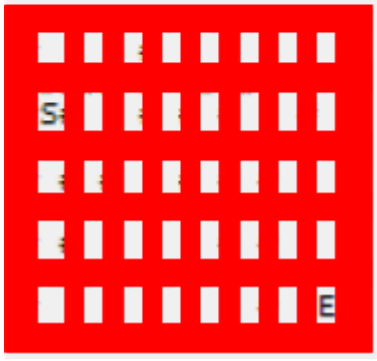
2.3.2 Data Structures

2.3.2.1 2D Array

One of the reasons I made node squares, the start square, the end square and walls all inherit the "Square" class (and make them all a type of square), is so that I could store them all in a single 2D array in order to abstractly store a maze. By storing the maze in one 2D array, it meant that I could output each square's associated string by running the OutputString function, for each square, either in the "Square" class or in one of the classes that inherits the "Square" class and overrides the OutputString function. As the OutputString function can get overridden, when it is, the string associated with the class which is inheriting the "Square" class e.g. "StartSquare" is returned (e.g. "S") instead of the default string (" ").



Key:
(top/first diagram):
S = start square
E = end square
= node square / space between two node squares that isn't a wall
 # = wall
(bottom/second diagram):
= node square
= possible walls



The top/first image shows how a maze is displayed through outputting the associated string of each square in the 2D array. The 2D array for this maze would have the same contents stored in it that is displayed, but instead of the string being stored, it would store the associated square of that string. The bottom diagram shows which squares are always traversable (the node squares) and how the squares between them can be randomly made into walls.

2.3.2.2 Adjacency Matrix

In order to store whether it is possible to traverse between two node squares, I implemented an adjacency matrix using a 2D array (that stores boolean data). The 2D array/adjacency matrix for each maze is created to have both its dimensions (width and height) each have a length of the total number of squares in the associated maze (as the 0 index isn't used). Each square is given a node number that maps to an index number within the array. If two squares are neighbours and there isn't a wall between them, then a True is stored at the index of the two squares' node numbers. Otherwise, if both the criterias are not met, False is stored. As the node numbers of two squares meet twice in the 2D array, True/False (whichever applies) is stored in both the indexes/cells they meet at (as shown in the diagram below).

	0	1	2	3
0	F	F	F	F
1	F	F	T	T
2	F	T	F	F
3	F	T	F	F

Key: T = True | F = False
= isn't used

The data stored here is the same as the indexes 1 & 3 meet twice in the 2D array so the same data is stored twice. Despite that unnecessary data gets stored, it means that when looking up using two squares' node numbers whether they can be traversed between, it's quicker to find out if they can or not as the order of the node numbers doesn't matter.

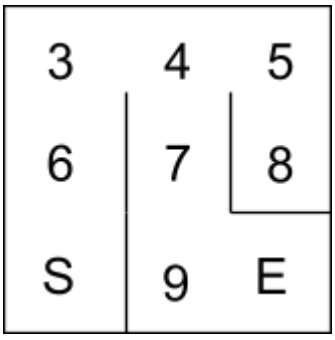
Adjacency Matrix:

	0	1	2	3	4	5	6	7	8	9
0	F	F	F	F	F	F	F	F	F	F
1	F	F	F	F	F	F	T	F	F	F
2	F	F	F	F	F	F	F	F	F	T
3	F	F	F	F	T	F	T	F	F	F
4	F	F	F	T	F	T	F	T	F	F
5	F	F	F	F	T	F	F	F	T	F
6	F	T	F	T	F	F	F	F	F	F
7	F	F	F	F	T	F	F	F	F	T
8	F	F	F	F	F	T	F	F	F	F
9	F	F	T	F	F	F	F	T	F	F

Example of how an adjacency matrix is used to store which squares are possible to traverse between in a maze.

■ = isn't used

Maze with the node numbers of each square:



Node numbers of the start and end square:

S = 1
E = 2

2.3.2.3 Queue

I implemented a circular priority queue as it is a first in first out abstract data structure, which meant I could use it to store a queue of squares that have been visited but not yet discovered. As the Breadth First Search only requires a normal (circular) queue and not a (circular) priority queue, the reordering of the queue using a bubble sort based on a cost type is performed in a subroutine in the Queue class that can be called upon from other classes. A* Star uses the f cost to order the squares in the queue making the square with lowest f cost go at the front of the queue, however if multiple squares have the same f cost, those squares are ordered by h cost. Dijkstra's Algorithm makes the queue order squares based on the g cost (the distance from the start square).

When implementing the "Queue" class, I created the following procedures:

- New (created an instance of a "StackOfSquare")
- Enqueue (adds the parameter input to the back of the queue)
- Dequeue (removes and return the square at the front of the queue)
- Peek (returns the square at the front of the queue, without removing it)
- IsFull (checks whether the stack is full, returns a boolean value)
- IsEmpty (checks whether the stack is empty, returns a boolean value)
- GetMaxSize (returns the maximum possible size of the queue)
- LoopRound (this procedure is what makes the queue a circular queue, it looks at whether values need to be looped around and loops them around if they do)
- CheckNumInList (checks whether the parameter inputted integer is in the parameter inputted list)
- BubbleSortQueue (sorts the queue based on the inputted cost type)

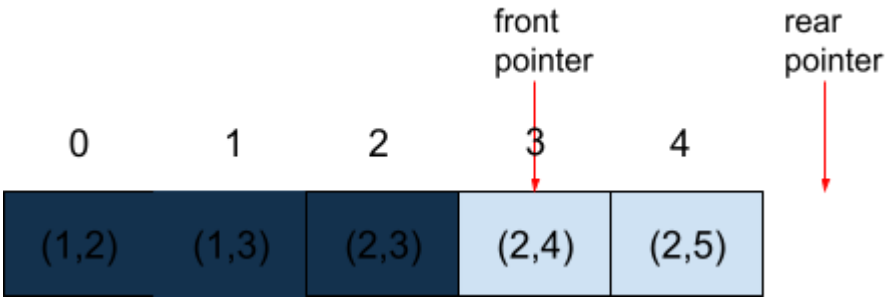
How the Queue loops round:

If the queue is not full, the program checks to see whether the front or rear pointer is greater than the value stored in the maxSize variable. If the rear pointer is greater, the following code is run “rear = (rear Mod maxSize) - 1” (if the front pointer is greater the same code is run but with “front”). The code that is run makes the pointer wrap/loop round to the start, making the queue a circular queue.

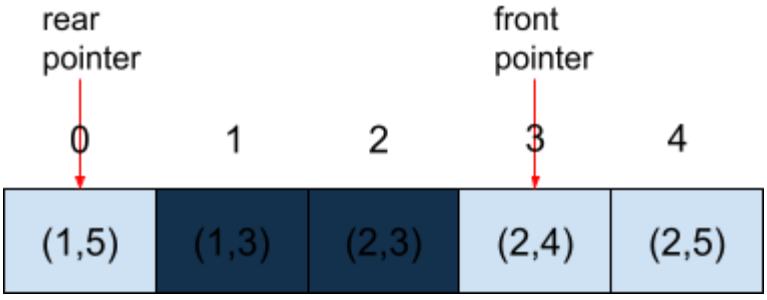
E.g.

(Data in a queue is never deleted only over-written)
■ = data that is not part of the active queue

Before:
maxSize = 4 (has a length of 5 though as arrays start at the index 0)
front = 3
rear = 5 (after increase of rear)



After the LoopRound subroutine is run:
maxSize = 4 (has a length of 5 though as arrays start at the index 0)
front = 3
rear = 0 (after increase of rear)



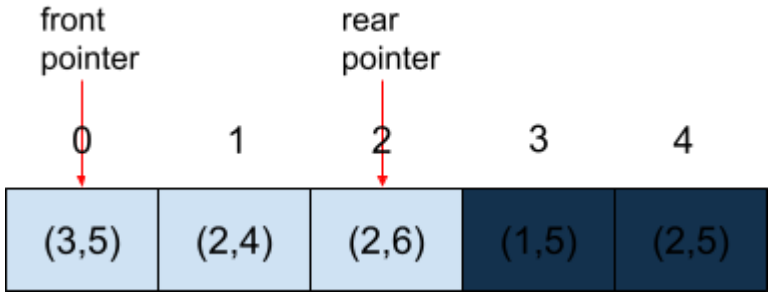
BubbleSortQueue:

The contents of the active section of the queue is removed and added to a list and then an array. A bubble sort is run on the array, based on the input cost type as the parameter input of the procedure, the program will order the array (from lowest to highest). If the program orders the array based on the h cost then the values with only the same (minimum) f costs must be reordered. If there are multiple squares with the same minimum f & h cost then they are randomly ordered. Starting from the front of the array, the squares are added back into the queue.

E.g., if (1,3) is the start square and (2,9) the end square:

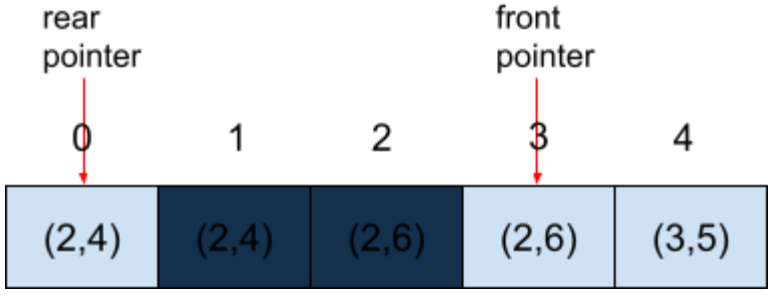
(Data in a queue is never deleted only over-written)
■ = data that is not part of the active queue

Before the bubble sort:



fCost: 21	fCost: 29	fCost: 13	fCost: 19	fCost: 19
hCost: 17	hCost: 25	hCost: 9	hCost: 17	hCost: 16
gCost: 4	gCost: 4	gCost: 4	gCost: 2	gCost: 3

After the bubble sort:



fCost: 29	fCost: 29	fCost: 13	fCost: 13	fCost: 21
hCost: 25	hCost: 25	hCost: 9	hCost: 9	hCost: 17
gCost: 4	gCost: 4	gCost: 4	gCost: 4	gCost: 4

2.3.2.4 Stack

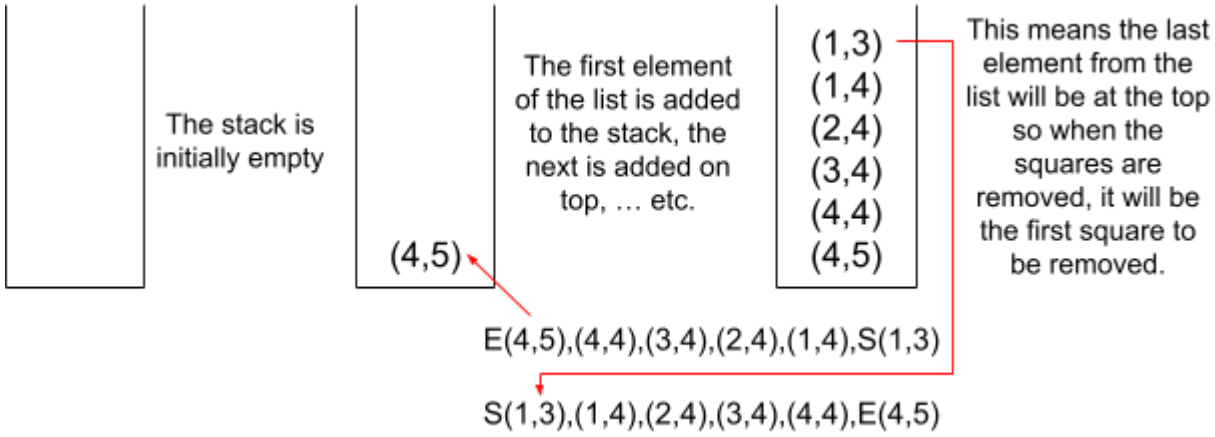
I implemented a stack as it is a first in last out abstract data structure, therefore once I had a list of the shortest path from the end square to the start square it allowed me to easily add each square to the stack. In order to get the shortest path from the start square to the end, I then made the program remove each square from the stack to put them back into the list but in the order where the start square is the first element and the end square is the last.

The Stack (at different stages):

Key: S = start square | E = end square

E.g. If the path before the rearrangement of the order was:
E(4,5),(4,4),(3,4),(2,4),(1,4),S(1,3)

Then the new order would be:
S(1,3),(1,4),(2,4),(3,4),(4,4),E(4,5)



When implementing the (stack) “StackOfSquare” class, I created the following procedures:

- New (created an instance of a “StackOfSquare”)
- Push (adds the parameter input to the top of stack)
- Pop (removes and return the square at the top of the stack)
- Peek (returns the square at the top of the stack, without removing it)
- IsFull (checks whether the stack is full, returns a boolean value)
- IsEmpty (checks whether the stack is empty, returns a boolean value)

2.3.3 Database Design

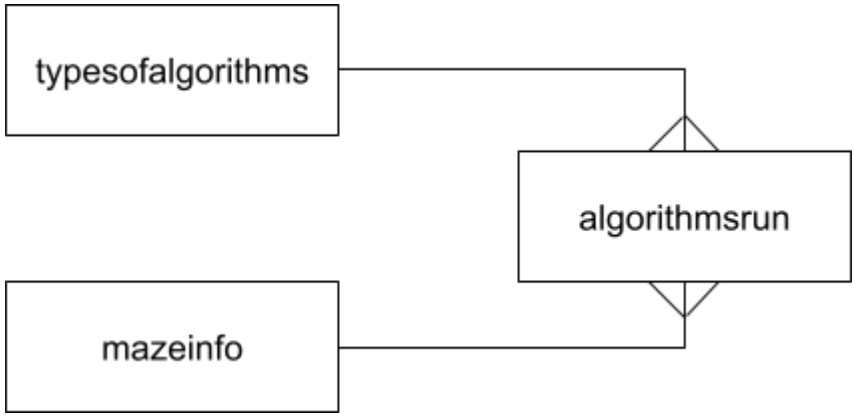
2.3.3.1 Entity-Relationship Diagram

In the modeling section of the document, I created an entity relationship diagram to plan how the tables in the database will be linked (using primary & foreign keys). When linking tables it is important to normalise the database to avoid there being repeating groups, to make all non-key attributes depend on the key, the whole key and nothing but the key. Furthermore, having a database in a normalised form means that things like data inconsistency and data redundancy are avoided. As the draft entity relationship diagram was already in normalised form, I kept the structure of how each table links together, the same.

The following entity relationship diagram (ER diagram) shows how each of the tables in the SQL database are linked together using foreign keys and primary keys.

.

Entity Relationship Diagram for pathfindingdatabase:



I chose to have three separate tables as I needed to store three types of information. mazinfo stores information needed to import a maze: mazeID, mazeName (not needed to load the maze but needed for the user to identity possible IDs of the maze if they only remember the name of the maze), mazeWidth, mazeHeight, startSquareX, startSquareY, endSquareX, endSquareY, wallStatus. Whereas, typesofalgorithms stores the associated algorithmID of each algorithm (algorithmName), so that the algorithm name can be accessed via its ID from the table. algorithmsrun stores the information about processes run (a process is when an algorithm is run on a maze): processID, mazeID, algorithmID, pathToEnd, timeTakenToRun, numberOfSquaresTraversed. By having foreign keys in the table “algorithmsruns” for the mazeID and algorithmID it means that information about a process such as the algorithm run and the maze it was run on can be identified & accessed as well as the data about the process.

2.3.3.2 Entity Description

Entity Description for pathfindingdatabase:

Key:
-Primary Key
-Foreign Key

pathfindingdatabase:
mazeinfo(mazeID, mazeName, mazeWidth, mazeHeight, startSquareX, startSquareY, endSquareX, endSquareY, wallStatus)
typesofalgorithms(algorithmID, algorithmName)
algorithmsrun(processID, *mazeID*, *algorithmID*, pathToEnd, timeTakenToRun, numberOfSquaresTraversed)

The pathfindingdatabase is fully normalised as in each table, there are no repeating groups, all non-key attributes depend on the key, the whole key and nothing but the key. The tables are linked together using the primary keys that are underlined and the *foreign keys that are in italics*.

As I mentioned before, “mazeinfo” is used to import a maze, each of its fields is required to load a maze. The mazeWidth and mazeHeight is used to store the dimensions of the maze. The coordinates of the start square are stored using the startSquareX field (to store the x coordinate) and startSquareY field (to store the y coordinate) . Similarly the coordinates of the end square are stored using the endSquareX field (to store the x coordinate) and endSquareY field (to store the y coordinate). The wallStatus field stores a list of 0s and 1s which determines whether each square in the 2D array (that represented the maze) was a wall or not, if it was a wall a 1 is stored if it wasn’t a wall a 0 is stored. Therefore, using the data stored in the table “mazeinfo”, the maze can be reconstructed and imported successfully. The field mazeName isn’t actually used to import a maze, its purpose is so that if the user forgets the mazeID but remembers the mazeName of a maze, they can look at mazes stored in the table “mazeinfo” and determine the possible ID the maze they are looking for could have (as mazes can have the same mazeName just not the same mazeID). By having these fields in the “mazeinfo” table it fulfills the objective **(3.1.1.)**.

The reason there are only two fields in “typesofalgorithms” is so that each algorithm (algorithmName) can have an algorithmID assigned to it, so if I ever wanted to add more algorithms, in the SQL database I would only need to add new records to the “typesofalgorithms” table and I wouldn’t need to change anything in the other tables. It is a requirement in the objectives that the table “typesofalgorithms” has these fields **(3.1.2.)**.

As stated by the objective **(3.1.3.)** & **(3.3.1.)**, the “algorithmsrun” table must have the fields: processID, mazeID, algorithmID, pathToEnd, timeTakenToRun, numberOfSquaresTraversed. The reason the primary key processID needs to be stored is because it's what uniquely identifies each process. The foreign keys are needed as they link the table to other tables (as I have mentioned before), also they allow the program to know what algorithm and maze was involved with each process. The two foreign keys couldn’t have acted as a composite key as the same algorithm can be run on the same maze. The rest of the data in the table “algorithmsrun” is stored because they are data that I’m measuring for my investigation.

2.3.4 SQL Query Design

2.3.4.1 Example Queries

SELECT MAX(mazeID)
FROM mazeinfo

The SQL statement above is used within VB to retrieve the maximum mazeID from the mazeinfo table (from the pathfinding database). The retrieved mazeID is needed in order to allocate a new maze the next possible mazeID (number), by using data that already exists and incrementing the highest mazeID retrieved by one it ensures that each maze has a unique mazeID assigned to them. In order for the SQL statement to be run by the program it must be wrapped in VB code in. The VB code it must be wrapped in, establishes a connection between VB, SQLyog and UniServer Zero, allowing the query to be run within SQLyog and the results (of the query) to be returned to the program.

SELECT processID, algorithmName, pathToEnd, timeTakenToRun, numberOfSquaresTraversed
FROM algorithmsrun, typesofalgorithms
WHERE algorithmsrun.mazeID = '" & id & "'
AND typesofalgorithms.algorithmID = algorithmsrun.algorithmID

The SQL statement above uses linked clauses to retrieve data from multiple tables. The line “AND typesofalgorithms.algorithmID = algorithmsrun.algorithmID” from the SELECT query is what links the two table “algorithmsrun” (in which the algorithmID is the primary key) and “typesofalgorithms” (in which the algorithmID is a foreign key). Without the linking cause, the data retrieved wouldn’t be the same to the data we want to be retrieved. The SQL statement above uses a user input: “id” (which is the ID of the maze that the user wants to load) within the query. The variable “id”’s value is incorporated into the SQL statement that selects the following: “processID, algorithmName, pathToEnd, timeTakenToRun, numberOfSquaresTraversed” from a record with the mazeID (“id”).

2.3.5 User Interface Design

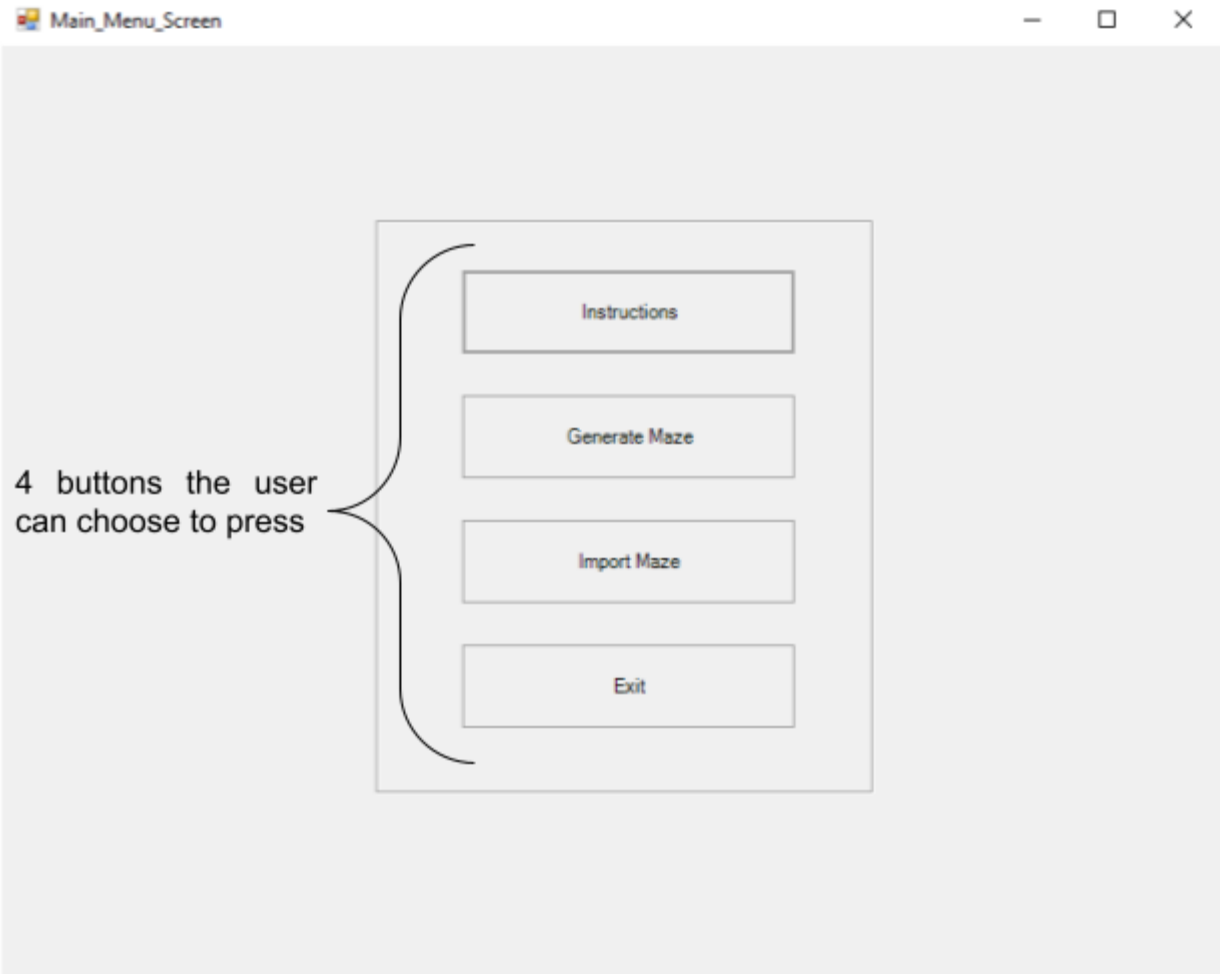
Instead of using a text-based user interface I chose to make my program use a GUI (Graphic User Interface), as it reduced the number invalid inputs I had to deal with if I only allowed the user to press buttons (on some windows). Despite the fact that you don’t need to code each object on the interface yourself and could just use the provided GUI to create a GUI for your program; I choose to program all of the screens myself in order to learn/practise manually coding a GUI in VB.

2.3.5.1 Main Menu

Initial Draft:

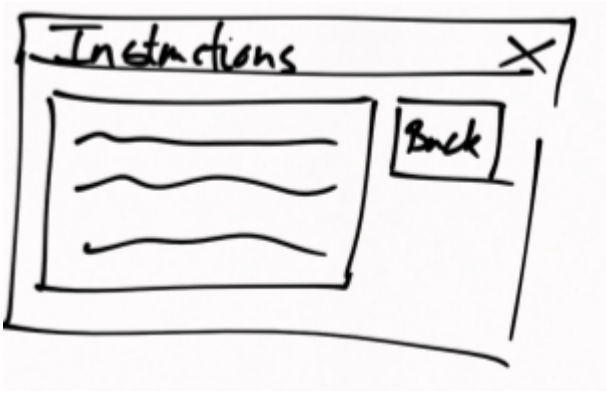
Final Version:

The screen below is shown/loaded when the program is run. Based on the flowchart model that I did, I planned for the main menu screen to branch to 3 other screens. So I added buttons that when pressed, lead to one of the three screens associated with the text on them. In the flowchart model, the main menu screen screen also allowed the user to exit the program, so I added an “Exit” button to allow the user to exit the program. I made the order of the buttons to be (from top to bottom): “Instructions”, “Generate Maze”, “Import Maze” and then “Exit”. The reason I put them in this order is because the user will likely press them in that order. In order to make the screen more organised, I centered the buttons on the window to make them easy to notice and to make the screen layout look relatively professional.



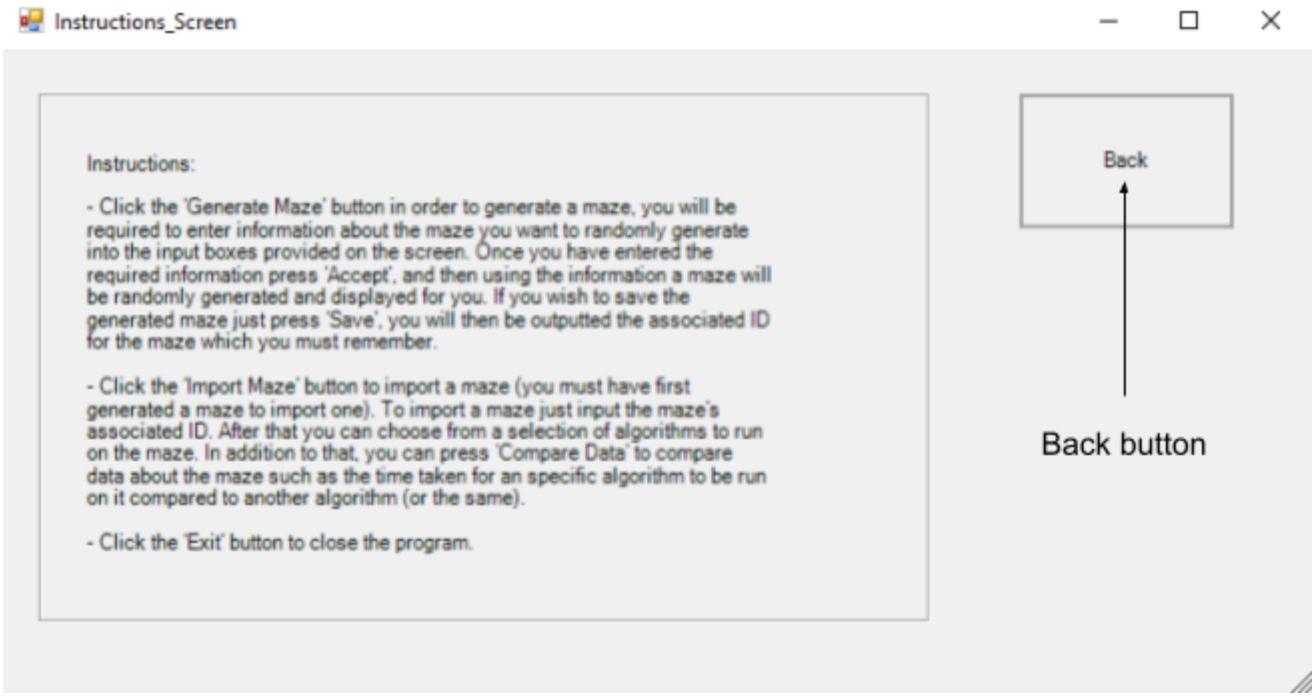
2.3.5.2 Instructions

Initial Draft:



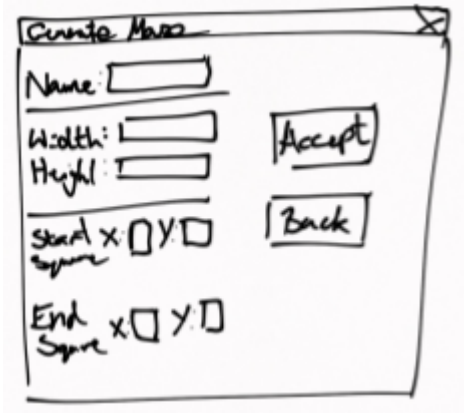
Final Version:

When the “Instructions” button is pressed by the user on the main menu screen the screen shown below is loaded. Despite the program not really needing an instructions screen, I felt that it might be useful to give a quick description of how the program works. Furthermore, one of the objectives **(4.1.)** states that there is a requirement for an instructions screen. I divided the instructions into 3 bullet points, each describes what happens (& etc) when the user clicks on one of the other buttons on the main menu screen. On the “Instructions” screen I added a “Back” button to allow the user to close this window, and return to the main menu screen.



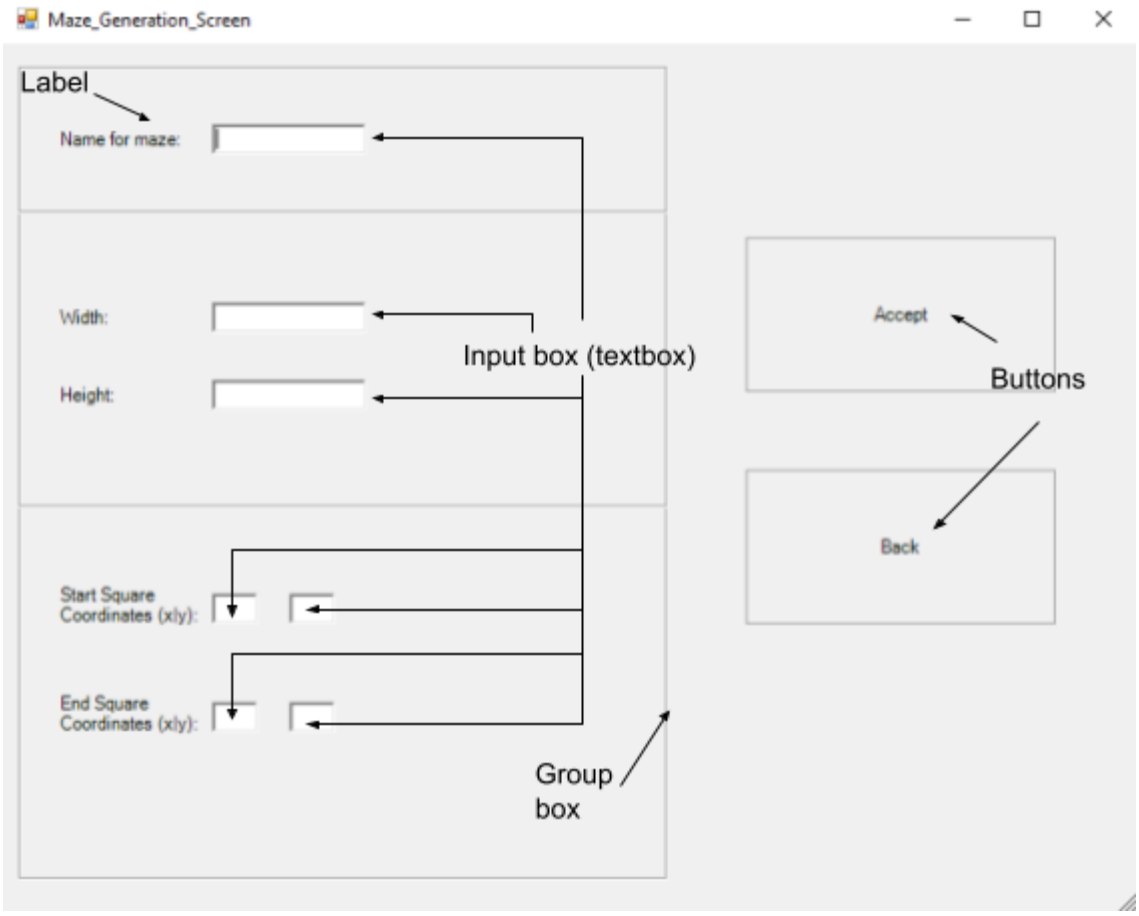
2.3.5.3 Generate Maze Screen

Initial Draft:

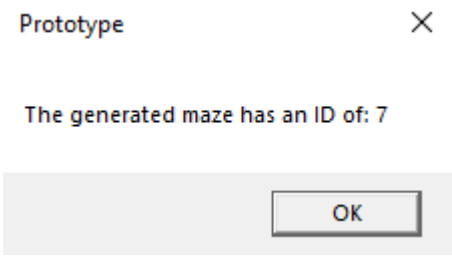


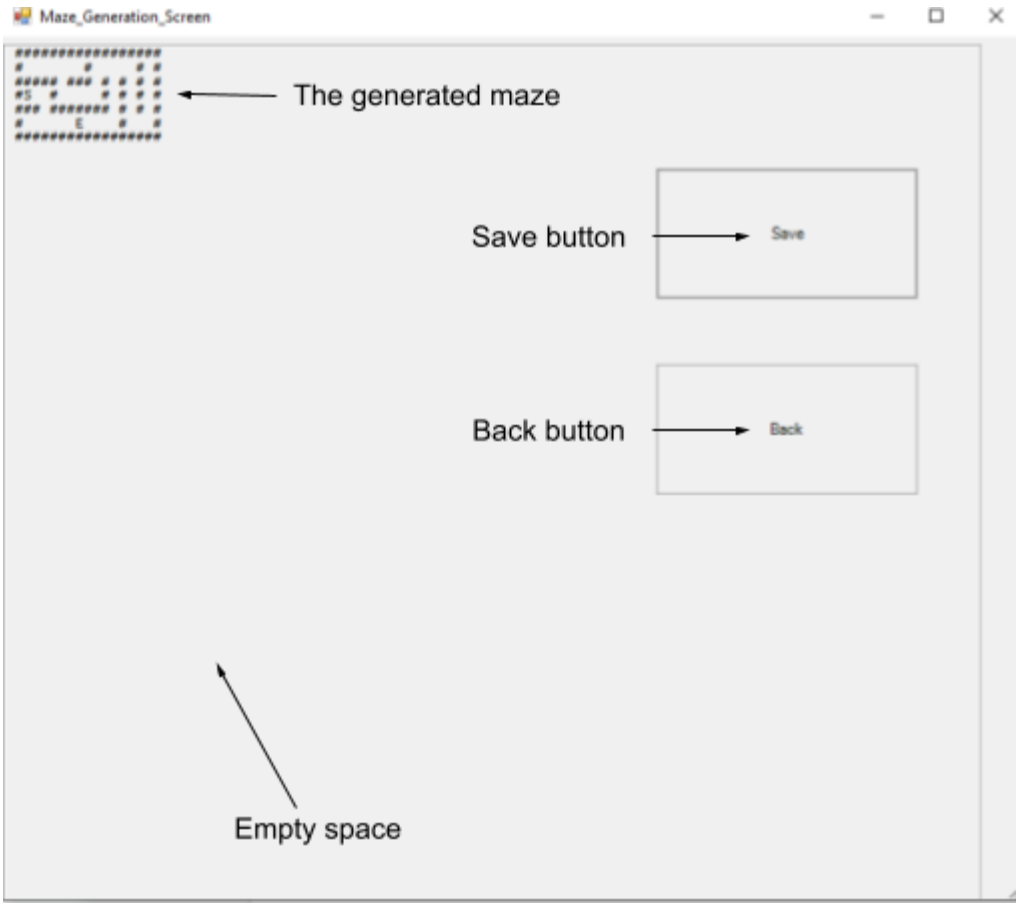
Final Version:

The screen shown below is loaded when the user chooses to generate a maze, it can be used by the user to randomly generate a maze. In order to make the program efficient, I made the program prompt users to enter all the data needed to generate a maze on the same screen, so that if a user wanted to change an input they could easily just change the input. Whereas if it was text-based and they made a mistake or wanted to change something, they would need to cancel the inputting of the data and re-input each data again. Each input box awaits an input from the user, the thing the user must enter is shown through the labels next to each input box. To make the interface even more organised I used group boxes to group similar input boxes together. In addition to that, I have made all the input boxes be on the left and the buttons on the right in order to keep similar things together and keep the window organised. The purpose of the “Accept” button is to allow the user to generate a maze with the current inputs, when this button is pressed error handling takes place, if any errors occur they are outputted below the input boxes of the input(/s) that is(/are) invalid.



The window shown below is loaded if the inputs to generate a maze are valid. The reason there is a lot of empty space on this windows form is so that the window doesn't need to change size if the generated maze is the maximum possible size (of 30x30). The "Back" button allows the user to go back to the screen where they can input values in order to make modifications to the previous data they have inputted. The "Save" button allows the user to save the generated maze in the SQL database ("pathfindingdatabase"), so that the maze can be imported. If the user presses the "Save" button, the program outputs the associated maze ID of the generated maze using a message box, as shown on the right. The reason I made a message box output the ID is because it stands out from window forms, therefore it increases the chances that the user reads what the maze ID is. Also as a message box requires user interaction the user is more likely to read the ID (and not miss it) than if I was to output the ID on the windows form for only a couple seconds.



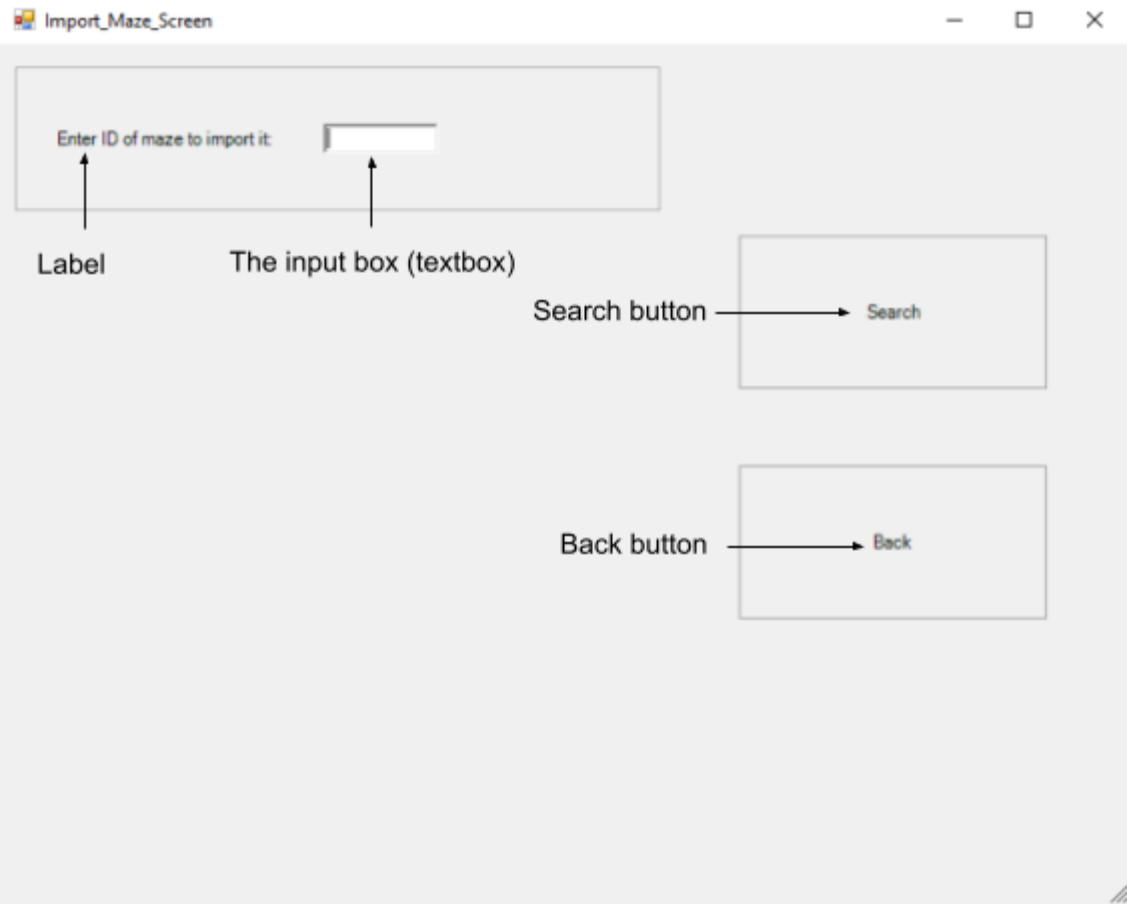


2.3.5.4 Import Maze Screen

Initial Draft:

Final Version:

The screen below is loaded if the user chooses to import a maze (on the main menu screen). Like on other screens, I have made the input box be on the left and the buttons on the right in order to group similar things together and keep the window organised. On this screen the label's purpose is to tell the user what they are required to input into the input box to the right of it, so into the input box the user must enter the maze ID they want to search for in. The purpose of the "Search" button is to allow the user to search for the maze with the ID they inputted. The purpose of the "Back" button is to allow the user to close this window, and return to the main menu screen.

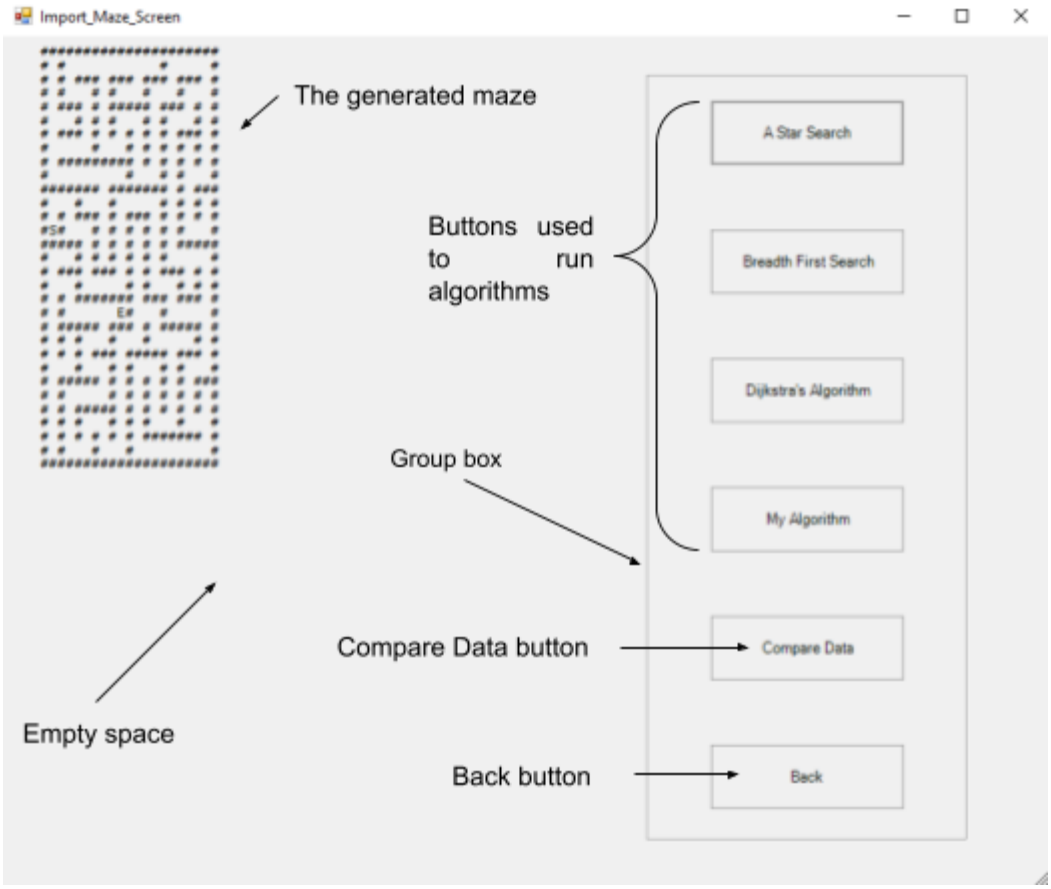


2.3.5.5 Run Algorithms Screen

Initial Draft:

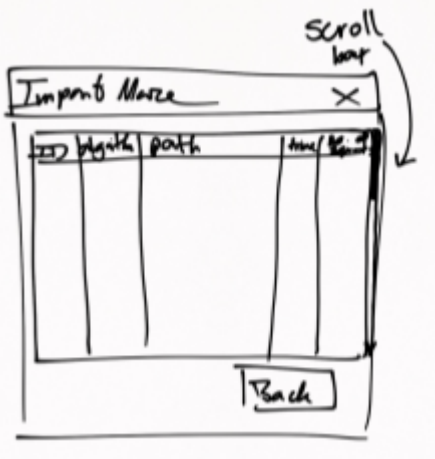
Final Version:

The screen below is loaded if the user inputs a valid and existing maze ID (on the previous screen shown on the “Import_Maze_Screen”). The reason there is this much empty space on this windows form is so that the window doesn’t need to change size if the maze has the maximum possible size (of 30x30). I have made the maze be outputted on the left and the buttons to run algorithms on the right in order to keep similar things together and keep the window organised. I also used group boxes to group the buttons together. The purpose of the buttons used to run algorithms, is to allow the user to run the algorithm associated with each button (labelled), on the imported maze. The purpose of the “Compare Data” button, when pressed, is to load a table of processes to allow the user to compare them. I added the “Back” button to allow the user to close this window, and return to the main menu screen.



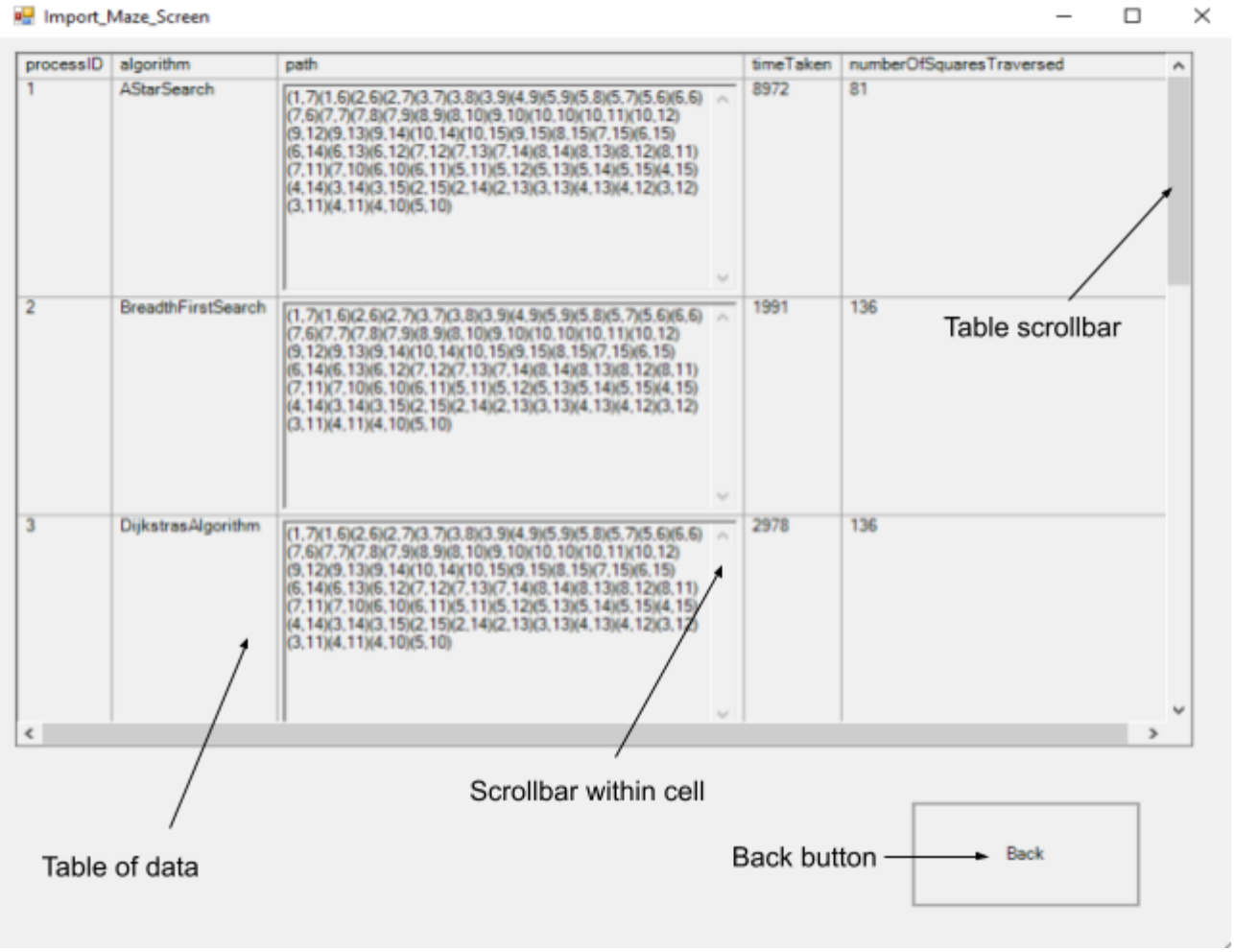
2.3.5.6 Compare Data Screen

Initial Draft:



Final Version:

The screen below is loaded if the user presses the “Compare Data” button on the screen that allows them to run algorithms on an imported maze. The screen itself displays a table of data that can be used to compare algorithms and a “Back” button that can be pressed by the user to go back to the previous screen on which they can choose to run more algorithms on the imported maze. I added a scrollbar to the table to allow the user to view all of the processes (records), even the ones that don’t currently fit on the windows form. On larger mazes, the path stored (from the start square to the end square) may not always fit into the cell within the table, to overcome this issue I added a scrollbar within the cell which becomes active if the displayed path coordinates do not fit into the cell, allowing the user to scroll down to see all the coordinates in the shortest path.

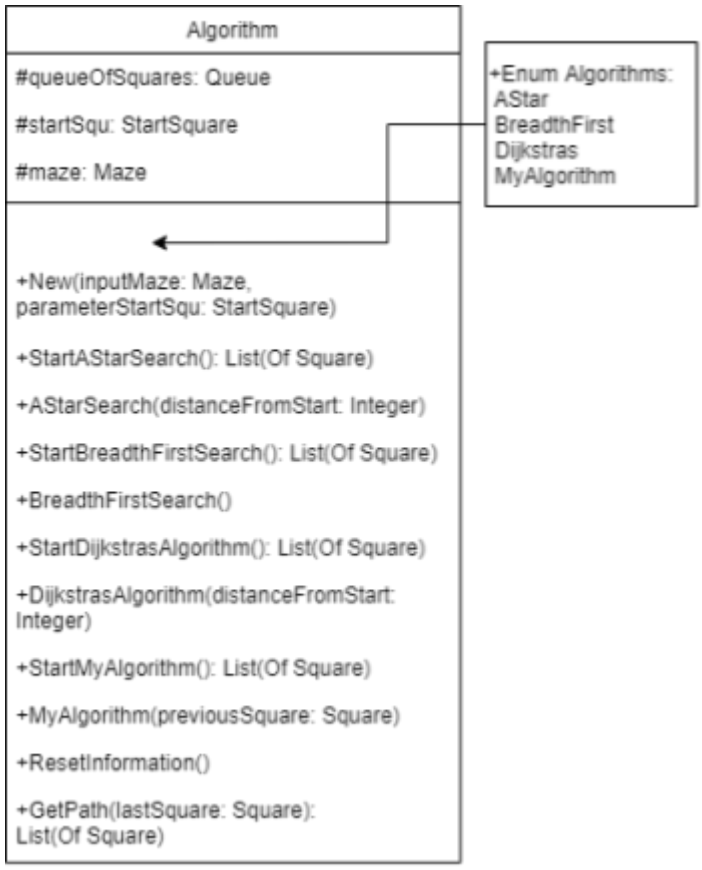


2.3.6 Object-oriented Design

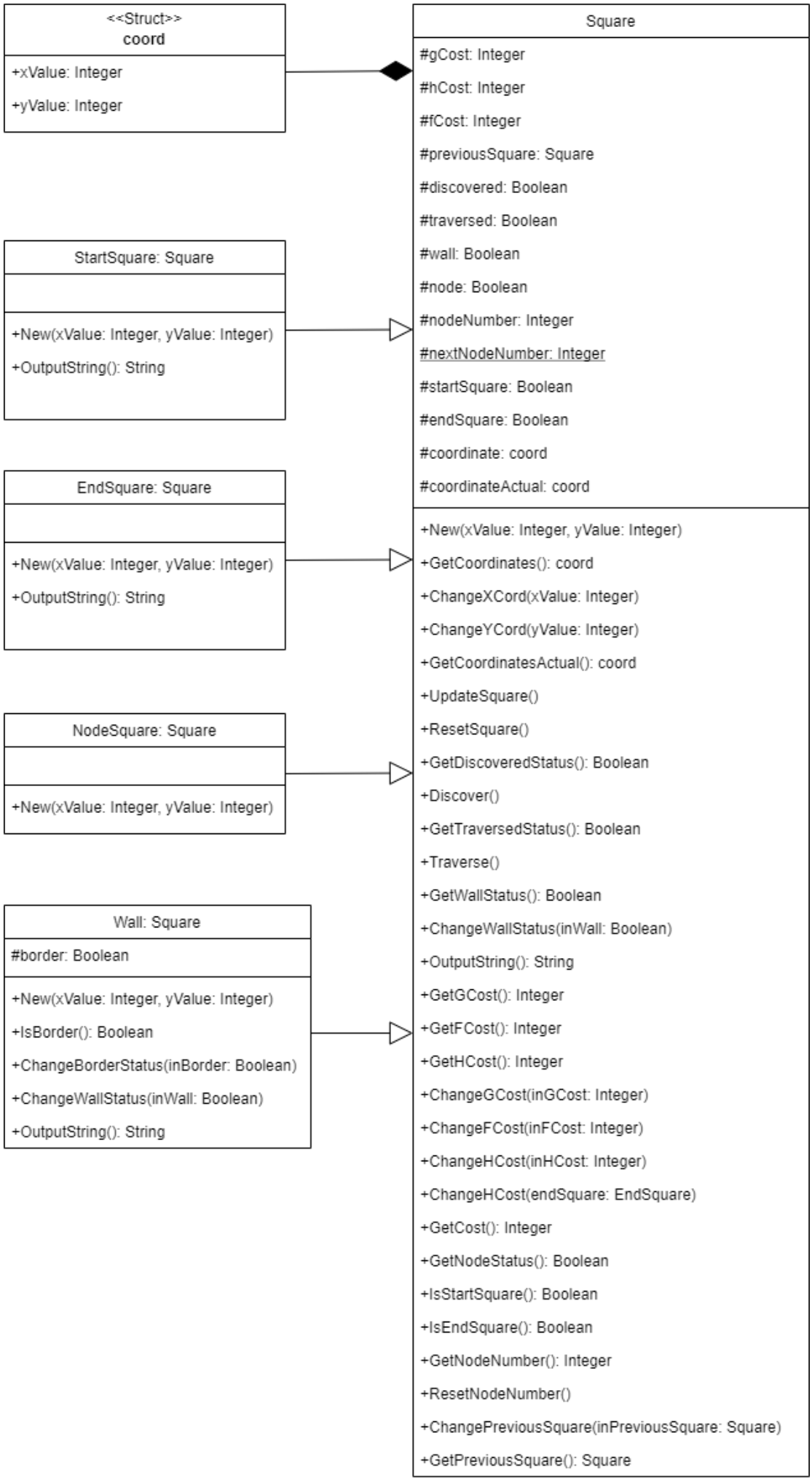
2.3.6.1 UML Class Diagrams

Note: The UML class diagrams focus on the relationships between classes (there are some descriptions of what the classes do e.g. for the expectation classes); the descriptions of what each class does, can be found in other sections of the design section of the document.

The UML class diagram on the right shows the properties and methods of the Algorithm class. The Enum “Algorithms” is part of the Algorithm class, it is required to be accessed by other classes therefore it has a public access modifier. All of the methods of the Algorithm class are public to allow them to be accessed from outside the class by other classes. The purpose of the Algorithm class is to run algorithms on a maze and return the shortest path found.

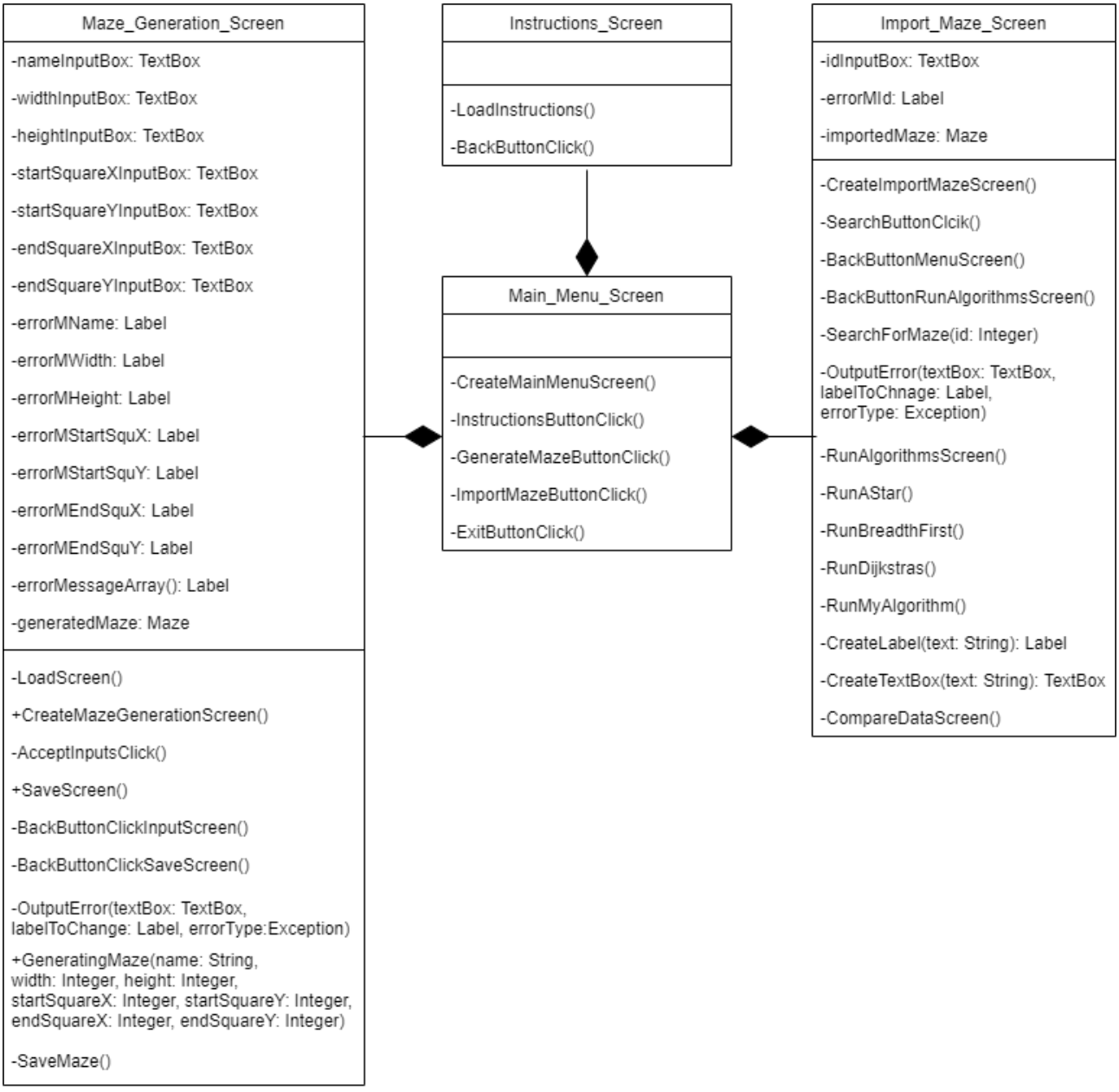


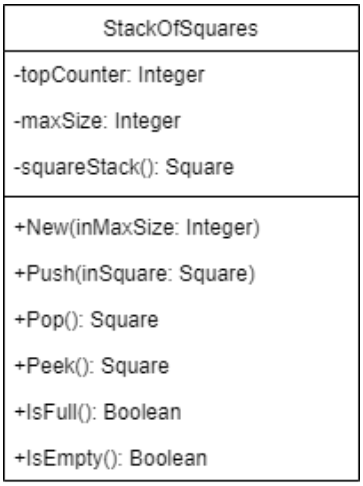
In the UML class diagram on the next page, the classes StartSquare, EndSquare, NodeSquare and Wall all inherit the Square class, therefore they all have the properties and methods in the Square class as well as additional ones. As the properties in the Square class are protected they can be directly accessed from within classes that inherit the Square class. All the methods have a public access modifier in the Square class, therefore they can be accessed by any class (and not just by the class itself or the ones that inherit it). The class diagram shows a composition (strong) relationship between the Square class and the “coord” structure, as the structure is actually a part of the Square class, it means that if the Square class didn’t exist neither would the coord structure. The StartSquare, EndSquare and Wall classes all use subtype polymorphism as they all override the overridable function OutputString() and as a result do & return something different to what the Square class would. The purpose of the Square class and the classes that inherit it, is to store information about each square within the maze (about both the traversable squares and walls).


<<Struct>>
coord
+xValue: Integer
+yValue: Integer
Square
#gCost: Integer
#hCost: Integer
#fCost: Integer
#previousSquare: Square
#discovered: Boolean
#traversed: Boolean
#wall: Boolean
#node: Boolean
#nodeNumber: Integer
#nextNodeNumber: Integer
#startSquare: Boolean
#endSquare: Boolean
#coordinate: coord
#coordinateActual: coord
+New(xValue: Integer, yValue: Integer)
+GetCoordinates(): coord
+ChangeXCord(xValue: Integer)
+ChangeYCord(yValue: Integer)
+GetCoordinatesActual(): coord
+UpdateSquare()
+ResetSquare()
+GetDiscoveredStatus(): Boolean
+Discover()
+GetTraversedStatus(): Boolean
+Traverse()
+GetWallStatus(): Boolean
+ChangeWallStatus(inWall: Boolean)
+OutputString(): String
+GetGCost(): Integer
+GetFCost(): Integer
+GetHCost(): Integer
+ChangeGCost(inGCost: Integer)
+ChangeFCost(inFCost: Integer)
+ChangeHCost(inHCost: Integer)
+ChangeHCost(endSquare: EndSquare)
+GetCost(): Integer
+GetNodeStatus(): Boolean
+IsStartSquare(): Boolean
+IsEndSquare(): Boolean
+GetNodeNumber(): Integer
+ResetNodeNumber()
+ChangePreviousSquare(inPreviousSquare: Square)
+GetPreviousSquare(): Square
StartSquare: Square
+New(xValue: Integer, yValue: Integer)
+OutputString(): String
EndSquare: Square
+New(xValue: Integer, yValue: Integer)
+OutputString(): String
NodeSquare: Square
+New(xValue: Integer, yValue: Integer)
Wall: Square
#border: Boolean
+New(xValue: Integer, yValue: Integer)
+IsBorder(): Boolean
+ChangeBorderStatus(inBorder: Boolean)
+ChangeWallStatus(inWall: Boolean)
+OutputString(): String

Despite the fact that nothing inherits the “Maze” class, I made the properties protected so that if I was to complete the extension objective of letting a maze have multiple solutions I could have created another class that would have inherited the Maze class. The methods all have a public access modifier so that they can be accessed by any class. Functions like GetStartSquare() allow other classes to access properties like startSqu indirectly, allowing the properties to be returned and used in other classes. The main purpose of the Maze class is to store information about a generated/imported maze like the list of squares that make up the maze (stored in the variable grid).

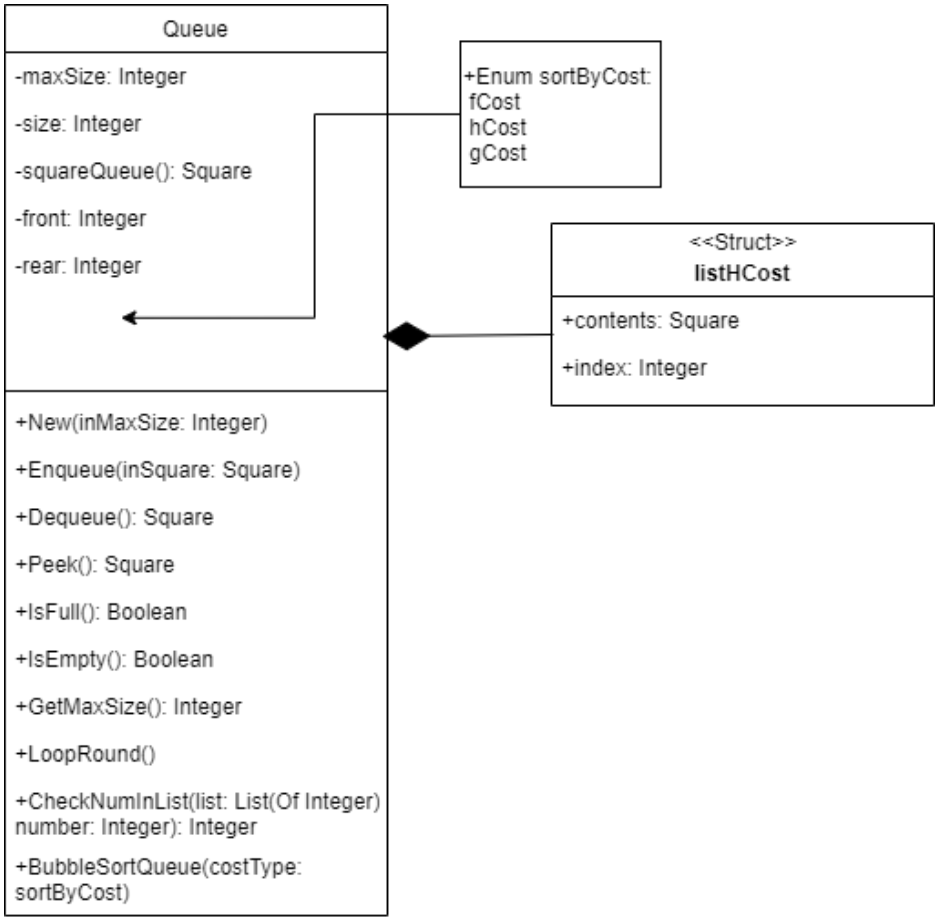
The UML class diagram section below shows the relationship between the classes which make up all the window forms. As an instance of each class doesn't exist if the Main_Menu_Screen class doesn't, they all have a composition relationship with the Main_Menu_Screen class. The role of the Main_Menu_Screen class is to allow the user to click on a button to load the instructions screen, the maze generation screen, the screen import a maze or to exit the program. The Instructions_Screen class' purpose is to display the instructions on how the program works to the user. The purpose of the Maze_Generation_Screen class is to allow the user to input data that can be used to randomly generate a maze. The purpose of the Import_Maze_Screen class, is to allow the user to import previously generated mazes, run algorithms on mazes and to compare data (gained from running algorithms on mazes).





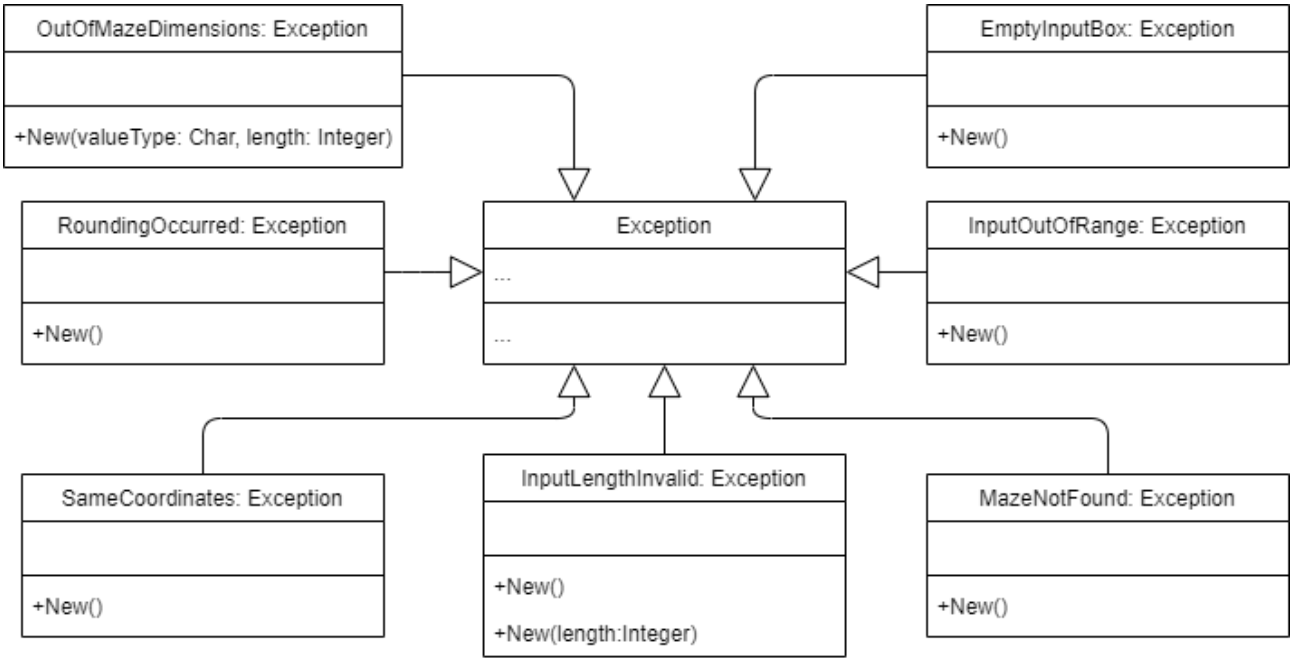
The UML class diagram on the left shows the properties and methods of the StackOfSquares class. The properties all have a private access modifier as they are only required to be accessed from within the StackOfSquares class. The methods are all public so that they can be called upon from other classes where a StackOfSquares is created. An instance of the StackOfSquare class is created in the Algorithm class, it is used to re-order the path to be from the start square to the end square (and not vice versa).

The UML class diagram on the right is a diagram of the Queue class I created. As nothing inherits from the Queue class, most its properties are private so the properties can only be accessed by methods within the class itself. The class diagram shows a composition (strong) relationship between the Queue and the structure listHCost, as the structure is actually part of the Queue class so if the Queue didn't exist neither does the structure listHCost. The Enum "sortByCost" is required to be accessed by other classes therefore it has a public access modifier. All of the methods of the Queue class are public to allow them to be accessed from outside the class by other classes. The Queue class is used by procedures (in the Algorithm class) that run algorithms on a maze to determine what order squares in a maze should be traversed.



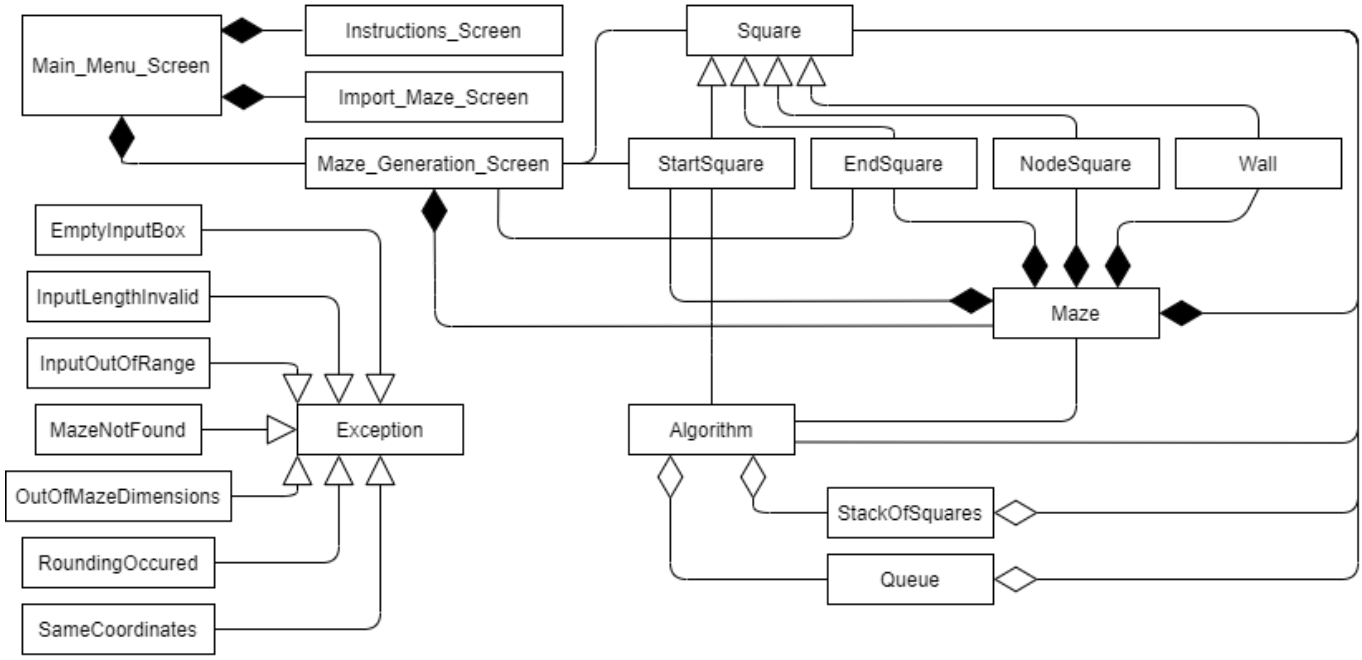
Through inheriting the in-built class “Exception”, it meant I could create custom exceptions and assign custom error messages by calling upon the New() subroutine of the Exception class (using myBase.New()) and inputting the custom error message in the parameters. After I had created custom exceptions like “RoundingOccured”, it meant that I could throw the exception and catch it using a try catch. As I outputted the error messages to the windows form so the user would know what the issue with the input is, I needed to create quite a few custom exceptions that have error messages that clearly describe the issue that occured. The UML class diagram below shows all the classes (custom exceptions) I created and made inherit the Exception class. All the methods are public in my custom exception, if they weren’t the exceptions couldn’t be thrown.

In the “InputLengthInvalid” class I used ad hoc polymorphism, as I made two New subroutines, one uses an input (Integer) and the other doesn’t use any inputs. The one that uses an input can use the input as part of the error message to make it clear to the user what the input must be less than, however as the exception may not always be thrown with a parameter input, I made a subroutine that would output an appropriate error message without any specific lengths.



2.3.6.2 Relationship between Classes

The UML class diagram below shows the relationship between all the classes I have created for my program (and the in-built exception class). As shown in the diagram, the main part of the program isn't linked with the custom exceptions I have created. The reason there is no link between them within the diagram, is because despite the fact that they are thrown by some of the classes, them being thrown doesn't count as a link. The relationship between the Maze and Algorithm class is association as they both reference each other in the program, however one doesn't own the other. Whereas the relationship between the Maze class and the classes: Square, StartSquare, EndSquare, NodeSquare, Wall, is a composition relationship. The reason it is a composition relationship is because if the Maze class didn't exist neither would the types of Square(s). In addition to that, there is a composition relationship between the Maze_Generation_Screen and the Maze class, where the Maze_Generation_Screen class has strong ownership over the Maze class. There is an aggregation relationship where the Algorithm class weakly owns the StackOfSquare and Queue classes. The reason the Algorithm class has weak ownership of these classes is because they theoretically still exist without the Algorithm class, however they are still owned by the Algorithm class. For the same reasons, there is also an aggregation relationship between the StackOfSquares class and the Square class where StackOfSquares has weak ownership of the Square class. Furthermore, there is also an aggregation relationship between the Queue class and the Square class, where the Queue class has weak ownership of the Square class.



2.3.7 Security and integrity of data

2.3.7.1 Security

My program doesn't have any security measures that prevents anyone with unauthorised access from using the program, as it doesn't handle sensitive data, it just allows people to run pathfinding algorithms on randomly generated mazes and compare data. Therefore, there was no need for security measures like login systems, etc.

If the tables “mazeinfo” and”algorithmsrun” in the SQL database (“pathfindingdatabase”) had all their records deleted it wouldn’t affect the creation of new records. It would just mean that previously stored data would be deleted, meaning the program would still work. However, if the data/records in the table “typesofalgorithms” were deleted, it would prevent processes from being loaded/new processes from being saved as the program wouldn’t know which algorithm ID is associated with which algorithm. So as there are no security measures in place to prevent people from deleting data from the “typesofalgorithms” table, so if someone does delete the records, the program will no longer work.

As my program stores data in a SQL database it means that if something causes the program to close unexpectedly e.g. a power cut, no data is lost. The reason for this is because the program saves data while it is run. The SQL database stores the data to load mazes and data about processes, so the data that was once temporarily stored by the program and saved (before it closed/stopped unexpectedly) can be loaded back from the database. If the program stops unexpectedly, already saved mazes and information about processes can be imported/loaded the same way as if the program didn’t stop.

2.3.7.2 Integrity

By using a GUI instead of the console, it means that users can only input data into input boxes (textboxes) and interact with things like buttons. So by using a GUI instead of the console, it drastically reduces the error handling needed, as in the console instead of buttons the user would need to e.g. choose which algorithm to run by inputting e.g. the number associated with the option they want to select (meaning that errors would need to be handled).

In order to prevent the user from entering invalid input when generating a maze or when importing a maze, I implemented error handling into my program. In addition to catching exceptions that already exist, I created my own exceptions that can be thrown and caught. I made each custom exception by creating a class that inherits from the “Exception” class, then I made the New() procedure that is used to create an instance of the class, call upon on the New() procedure of the “Exception” class and pass the error message of the new exception as the parameter input.

I made the following custom exceptions:

- EmptyInputBox - is relevant to throw when a textbox (input box) is left empty
- InputLengthInvalid - is relevant to throw when the input exceeds a character limit
- InputOutOfRange - is relevant to throw when the inputted integer isn’t in the required range
- MazeNotFound - is relevant to throw when the inputted maze ID can’t be found
- OutOfMazeDimensions - is relevant to throw when the start/end square coordinates aren’t in the inputted dimensions of the maze
- RoundingOccurred - is relevant to throw when there is a decimal input and therefore a rounding has occurred
- SameCoordinates - is relevant to throw when the start and end square have the same x and y coordinates

---

3 Technical Solution

Contents page for code (classes including complex code in **bold**)

Classes not highlighted bold, likely still include some complex code as well. The "**Algorithm Class**" is the most complex class therefore I have also underlined it.

Code

Code (in appendix):

Code for Technical Solution starts at 134

---

4 Testing

4.1 Introduction to Testing

In order to provide evidence on how well my technical solution has fulfilled the objectives and how robust the program is, I performed multiple tests on my program. I have created a video to provide visual evidence for most of the tests (the link to my video is below this paragraph). As it would have been time consuming to show me completing each algorithm by hand to show that the program performs each algorithm correctly in the video, I have taken pictures of my workings which show how the program should run each algorithm on one of the mazes (“maze2”) in the video.

4.2 Video Evidence

Video Evidence for testing: https://youtu.be/mggdXntCx_4

4.3 Figures/Evidence used during Testing

Figure 1: Comparison of the generated version and the imported version of the maze “maze” to show that it is the same.

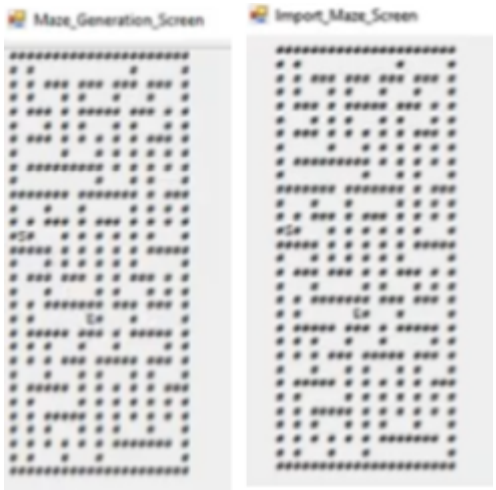


Figure 2: Image of the maze “maze2”

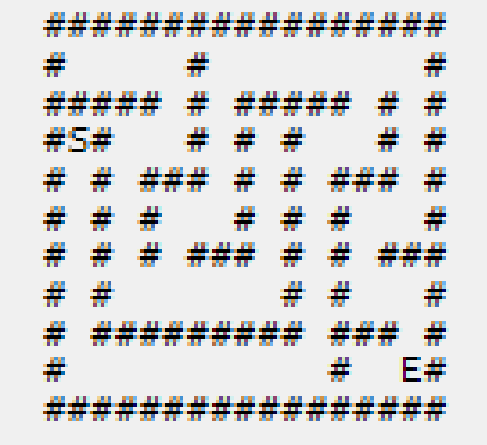


Figure 3: These are my hand-written workings on how the program should run the **A-Star Search** Algorithm on the maze “maze2”. The bottom of this figure contains images of what the path & numberOfSquaresTraversed the program calculated to be, they are the same to what I calculated.

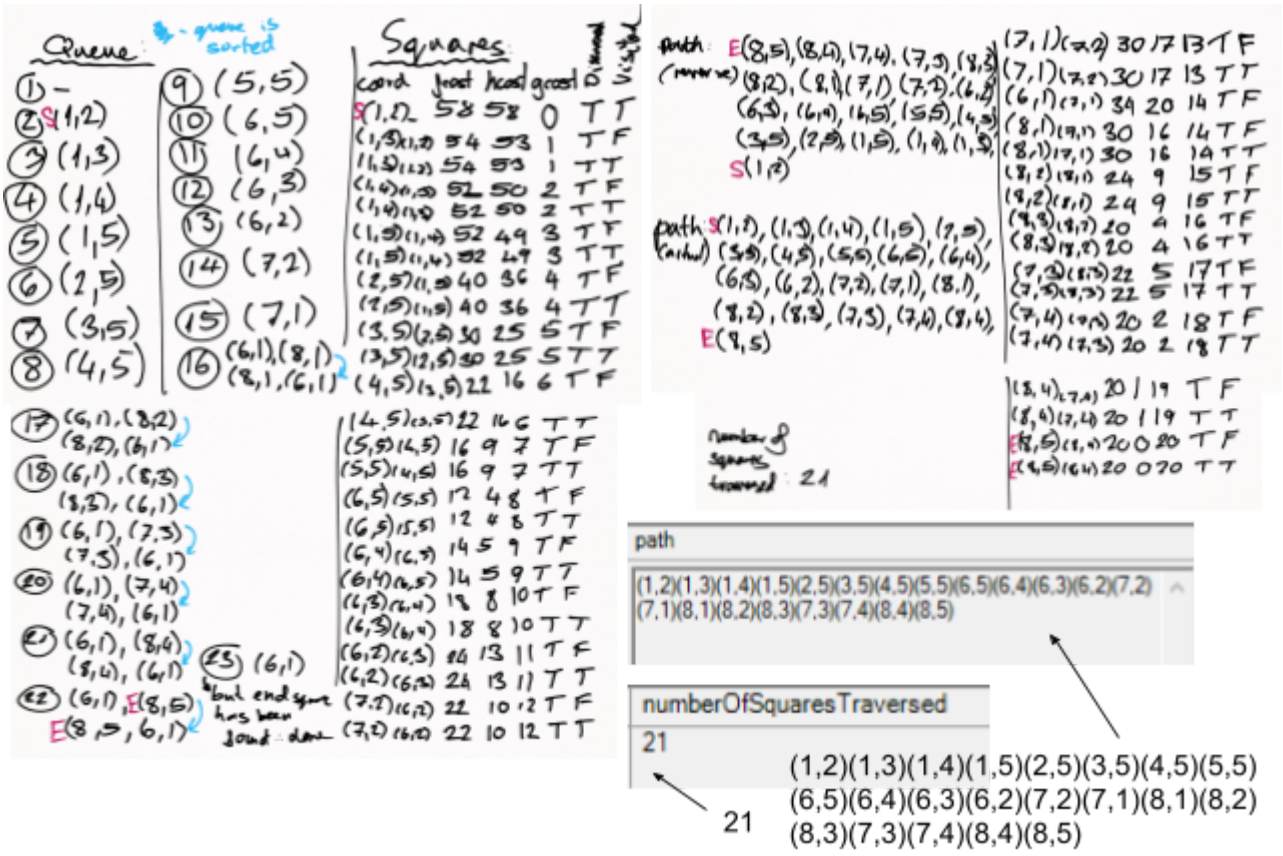
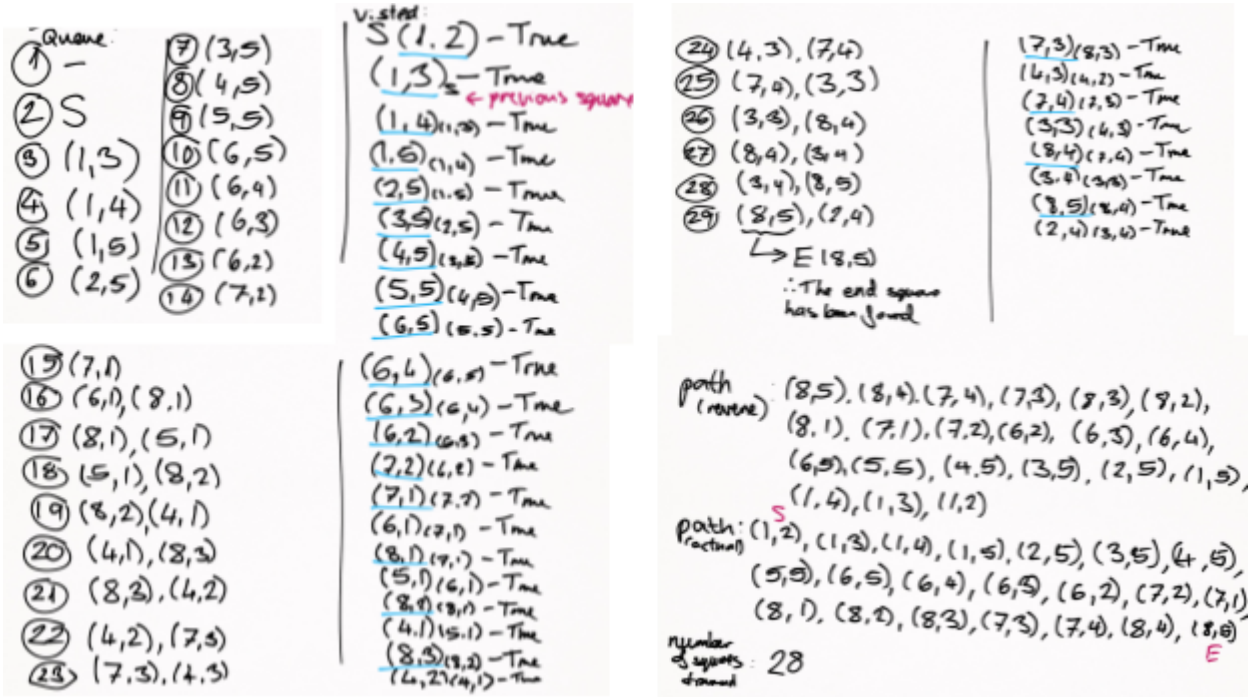


Figure 4: These are my hand-written workings on how the program should run the **Breadth-First Search** Algorithm on the maze “maze2”. The bottom of this figure contains images of what the path & numberOfSquaresTraversed the program calculated to be, they are the same to what I calculated.



path

(1,2)(1,3)(1,4)(1,5)(2,5)(3,5)(4,5)(5,5)(6,5)(6,4)(6,3)(6,2)(7,2)(7,1)(8,1)(8,2)(8,3)(7,3)(7,4)(8,4)(8,5)

numberOfSquaresTraversed

28

(1,2)(1,3)(1,4)(1,5)(2,5)(3,5)(4,5)(5,5)(6,5)(6,4)(6,3)(6,2)(7,2)(7,1)(8,1)(8,2)(8,3)(7,3)(7,4)(8,4)(8,5)

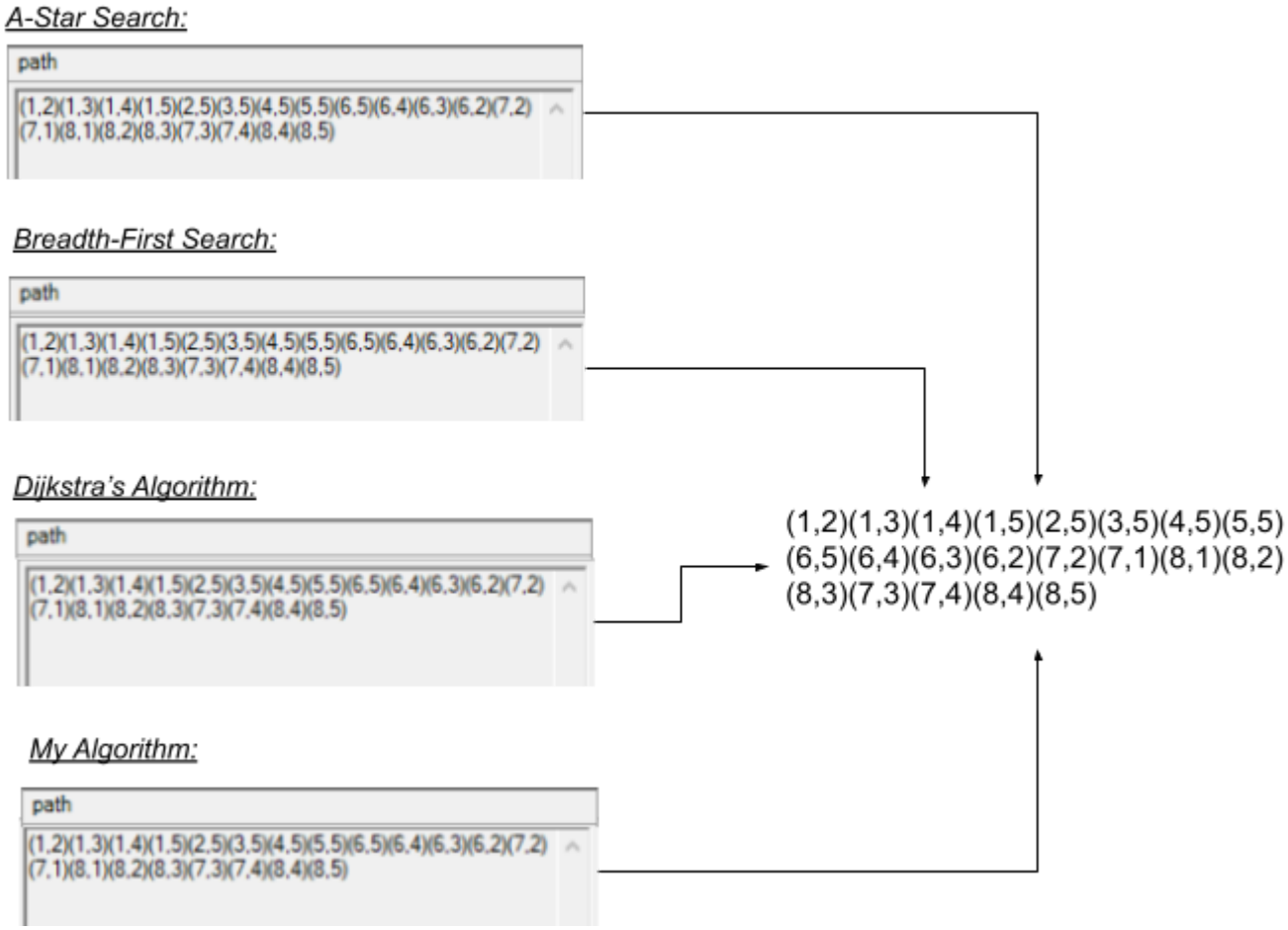
28

Figure 5: These are my hand-written workings on how the program should run the **Dijkstra's Algorithm** on the maze "maze2". The bottom of this figure contains images of what the path & numberOfSquaresTraversed the program calculated to be, they are the same to what I calculated.

(1,2)(1,3)(1,4)(1,5)(2,5)(3,5)(4,5)(5,5)(6,5)(6,4)(6,3)
(6,2)(7,2)(7,1)(8,1)(8,2)(8,3)(7,3)(7,4)(8,4)(8,5)

28

Figure 6: This shows the shortest route/path calculated by each algorithm from the start square to the end square. All the shortest paths calculated match, therefore **My Algorithm** has successfully calculated the shortest path.



4.4 Testing Table

Test Number	Description of purpose, Associated Objective as (x)	Input	Expected Outcome	Pass /Fail	Evidence (the timestamp for videos sometimes shows the start time of the event/evidence)
Validation (some validation in objective categories)					
1	To test to see whether the program handles the error correctly when the input boxes on the 'Generate Maze' screen are left empty.	N/A (as all input boxes are left empty)	Associated error messages are outputted. E.g. "This box cannot be left empty" and "Conversion from string "" to type 'Integer' is not valid."	Pass	Video - 0:29
2	To test whether the program handles invalid (out of range) data inputted into the given input boxes how it should.	Width: "0" Height: "1000"	Output the error message: "Input isn't in the valid range" and the contents of the input box should be deleted.	Pass	Video - 0:23
3	To test whether the program can successfully handle a string input in an input box that only expects an integer input.	Width: "str"	Output the error message: "Conversion from string "str" to type 'Integer' is not valid." And the contents of the input box containing the invalid input should be removed.	Pass	Video - 0:36
4	To test whether the program will prevent the user from using a decimal instead of an integer to generate a maze.	Height: "6.3"	Output the message: "You cannot enter a decimal, the input must be an integer". And the contents of the input box containing the invalid input should be removed.	Pass	Video - 0:39
5	To test whether the program accepts valid inputs for the width and height.	Width: "10" Height: "15"	No error message should be outputted for the width/height. But the program shouldn't generate a maze yet, as other inputs are still invalid.	Pass	Video - 0:43
6	The purpose of this test is to test whether the program handles invalid inputs for both the x and y coordinates for the start square how it should.	For the start square "11" is inputted as x and "16" for y.	For both x and y coordinates (of the start square) the associated error messages should be outputted: "X: Input isn't inside the given maze dimensions, input should be between 1 10" and "Y: Input isn't inside the given maze dimensions, input should be between 1 15". Also, the contents of the	Pass	Video - 0:48

			input box should be deleted.		
7	The purpose of this test is to test whether the program handles one valid and one invalid input for the x and y coordinates for the end square how it should.	For the end square “11” is inputted as x and “9” for y.	For the x coordinate (of the end square) the error message “X: Input isn’t inside the given maze dimensions, input should be between 1 10” should be outputted and the contents of its input box should be deleted. However, for the y coordinate nothing should happen as it is valid.	Pass	Video - 0:58
8	The purpose of this test is to show that if all the data inputted is valid the program generates a maze.	Name for maze: “maze” Width: “10” Height: “15” Start Square coordinates (x|y): “1” “7” End Square coordinates (x|y): “5” “9”	The maze should successfully generate a maze using the inputted data and display it to the user.	Pass	Video - 1:11
9	To test whether the program can handle an invalid data type input, in the input box for the ID of the maze (on the import maze screen).	“stringinput”	Output the error message: “Conversion from string “stringinput” to type ‘Integer’ is not valid”. And delete the invalid content of the input box.	Pass	Video - 3:40
10	To test whether the program can handle an invalid integer input that cannot be an ID for a maze.	“0”	Output the error message: “A maze with that ID cannot be found”.	Pass	Video - 3:34
11	To test whether the program can handle the error that occurs when a maze of an ID of nothing is attempted to be searched for.	N/A (as input box is left empty)	Output the error message: “Conversion from string “” to type ‘Integer’ is not valid.”	Pass	Video - 3:33
12	To test whether the program can handle the error that occurs when a maze of an ID which is a decimal is searched for.	“2.5”	Output the message: “You cannot enter a decimal, the input must be an integer”. And the invalid contents of the input box should be removed.	Pass	Video - 2:23
Objectives 1 - Allow the user to choose from multiple pathfinding algorithms:					
13	**(1.1.) & (1.2.)** To see whether the program clearly displays the	On the Import Maze screen the Search	An interface should be displayed to the user where they can choose to	Pass	Video - 3:49

	possible pathfinding algorithms that the user can choose from to run on a maze.	button is pressed with an valid input for the ID of "1".	run a pathfinding algorithm from the following: "A-Star Search", Breadth-First Search", "Dijkstra's Algorithm" or "My Algorithm"		
14	**(1.1.1.)** To see if the A-Star Search works how it should, I ran the A-Star Search on the maze 'maze2' and then I manually ran the algorithm by hand on the same maze and compared the two processes and their results.	The maze with an ID of "2" is loaded, then the button to run the A-Star Search on the maze ("maze2") is pressed.	path: "(1,2)(1,3)(1,4)(1,5)(2,5)(3,5)(4,5)(5,5)(6,5)(6,4)(6,3)(6,2)(7,2)(7,1)(8,1)(8,2)(8,3)(7,3)(7,4)(8,4)(8,5)" numberOfSquaresTraversed: "21"	Pass	Video - 5:34 Video - 5:57 ***Figure 2 (the maze "maze2")*** ***Figure 3***
15	**(1.1.2.)** To see if the Breadth-First Search works how it should, I ran the Breadth-First Search on the maze 'maze2' and then I manually ran the algorithm by hand on the same maze and compared the two processes and their results.	The maze with an ID of "2" is loaded, then the button to run the Breadth-First Search on the maze ("maze2") is pressed.	path: "(1,2)(1,3)(1,4)(1,5)(2,5)(3,5)(4,5)(5,5)(6,5)(6,4)(6,3)(6,2)(7,2)(7,1)(8,1)(8,2)(8,3)(7,3)(7,4)(8,4)(8,5)" numberOfSquaresTraversed: "28"	Pass	Video - 5:39 Video - 5:57 ***Figure 2 (the maze "maze2")*** ***Figure 4***
16	**(1.1.3.)** To see if Dijkstra's Algorithm works how it should, I ran Dijkstra's Algorithm on the maze 'maze2' and then I manually ran the algorithm by hand on the same maze and compared the two processes and their results.	The maze with an ID of "2" is loaded, then the button to run Dijkstra's Algorithm on the maze ("maze2") is pressed.	path: "(1,2)(1,3)(1,4)(1,5)(2,5)(3,5)(4,5)(5,5)(6,5)(6,4)(6,3)(6,2)(7,2)(7,1)(8,1)(8,2)(8,3)(7,3)(7,4)(8,4)(8,5)" numberOfSquaresTraversed: "28"	Pass	Video - 5:45 Video - 5:57 ***Figure 2 (the maze "maze2")*** ***Figure 5***
17	**(1.1.4.)** To see if My Algorithm can calculate the shortest route/path I compared the shortest route/path it calculated to all the shortest route/path calculated by the other already existing algorithms.	The maze with an ID of "2" is loaded, then the button to run My Algorithm on the maze ("maze2") is pressed.	path: "(1,2)(1,3)(1,4)(1,5)(2,5)(3,5)(4,5)(5,5)(6,5)(6,4)(6,3)(6,2)(7,2)(7,1)(8,1)(8,2)(8,3)(7,3)(7,4)(8,4)(8,5)"	Pass	Video - 6:01 Video - 6:10 (only for a short period) ***Figure 2 (the maze "maze2")*** ***Figure 6***
Objectives 2 - Maze					
18	**(2.1.)** Test to show that the user can choose to generate or import a maze.	The program is run and the main menu screen is displayed	The program should allow the user to choose to generate or import a maze by displaying buttons that can be clicked.	Pass	Video - 0:05

		which has buttons that can be pressed to generate / import a maze.			
19	**(2.1.1.1.)** The user should be able to enter a name for the maze when generating one. This test's purpose is to show that the user is successfully promoted to enter a name for the maze.	The user presses the "Generate Maze" button on the main menu.	The user should be promoted with multiple input boxes (textboxes), one of them should allow/promote the user to enter a name for the maze.	Pass	Video - 0:16
20	**(2.1.1.1.)** The user should be able to enter a name for the maze when generating one. This test's purpose is to show that the program allows the user to enter a valid name for the maze.	The user inputs "maze" as the name and other valid inputs. They then press the "Accept" button.	The program should allow the user to generate a maze with the inputs.	Pass	Video - 1:19
21	**(2.1.1.2.)** Test to show that the user is asked/promoted to enter dimensions for the maze they want to create.	The user presses the "Generate Maze" button on the main menu.	The user should be promoted with multiple input boxes (textboxes), one of them should allow/promote the user to enter dimensions for the maze.	Pass	Video - 0:16
22	**(2.1.1.3.1.1.)** Test to show that the program accepts the extreme minimum value for the width and height of the maze.	Width: "2" Height: "2" And other valid inputs.	The program should generate a maze with a dimension of 2x2 traversable squares.	Pass	Video - 2:16
23	**(2.1.1.3.1.2.)** Test to show that the program accepts the extreme maximum value for the width and height of the maze.	Width: "30" Height: "30" And other valid inputs.	The program should generate a maze with a dimension of 30x30 traversable squares.	Pass	Video - 2:36
24	**(2.1.2.1.)** Test to show that a user can import a maze from the SQL database through inputting its ID.	ID: "1"	The associated maze with an ID of "1" should be loaded and displayed ready to have pathfinding algorithms run on it.	Pass	Video - 3:47
25	**(2.1.2.1.)** Test to show that the program prevents the user from importing a maze using an ID that is valid but doesn't exist. And that the user can choose to	ID: "100"	Output the error message: "A maze with that ID cannot be found".	Pass	Video - 3:27 Video - 1:50 (shows that the "Back" can be pressed to return to the main menu

	enter another ID or go back by pressing the back button if they wanted to, in order to e.g. generate a maze instead.				screen).
26	**(2.2.)** Despite this test not being able to show that the maze is stored/represented as an adjacency matrix and as an 2D array. It can partially show that the maze is represented as a 2D, as the associated character of each square (within the 2D array) is outputted after a maze has been generated/imported.	The user generates a maze (e.g. “maze”) by inputting valid values, then pressing the button “Accept”.	The maze is outputted using the associated character of each square (within the 2D array), in order to display the maze to the user.	Pass	Video - 1:34
27	**(2.3.)** A test to show that the program successfully allows the user to save a maze.	Once a maze has been generated, the user presses the “Save” button. Then they import the maze “maze” using its ID “1”.	Information to load the maze should be stored in the table “mazeinfo” within the SQL database. In addition, the maze outputted on the generation screen should match the maze displayed on the import screen once the maze has been imported using its ID.	Pass	Video - 1:23 Video - 3:54 Video - 4:12 ***Figure 1*** (in the testing section)
28	**(2.3.1.)** A test to show that the maze’s walls’ locations are stored in an table (within an SQL database) using 1s and 0s	N/A (but the user must have generated a maze, e.g. “maze”, beforehand as the maze’s walls’ locations is information about an existing maze, so it cannot be accessed if it doesn’t exist).	Within a table (in the SQL database), information about a maze’s walls should be stored, as 1s and 0s. For the test the information should be about the generated maze “maze”.	Pass	Video - 4:17
29	**(2.3.2.)** The program should output the associated maze ID of the generated maze after the user chooses to save it.	The user presses the “Save” button. After generating a maze e.g. the maze “maze”	The next available maze ID that was assigned to the maze generated should be outputted using a message box. E.g. for the maze “maze”, the outputted ID should be “1” and for the maze “small” it should be “3”.	Pass	Video - 1:42 Video - 2:30
30	**(2.3.3.)** The maze should load in less than	A valid input for the ID of e.g.	The program should load the maze using its ID in	Pass	Video - 3:50 (the program

	10 seconds to keep the program relatively efficient.	"1" is entered to allow a maze to be loaded. The user then presses the "Search" button and the program starts loading the maze.	less than 10 seconds.	As it took ~4 seconds.	starts loading the maze) Video - 3:54 (the maze is loaded by the program)
31	**(2.4.)** The purpose of this test is to show that the user cannot generate a maze where the start and end square have the same position/coordinates.	Same coordinates for the start & end square. For both x(s): "1" and for both the y(s): "9".	Output the error message "The start and end square cannot have the same position/coordinates", and delete the contents of the input boxes with invalid data.	Pass	Video - 1:04
32	***(Extension 2.5.1.)*** To test whether the program displays the maze (the start & end square and the walls). And to see whether the maze displayed on the generation screen matches the imported.	Valid inputs to generate the maze and the clicking of the buttons to save & import it. The input of "1" as the mazeID to import the maze.	The maze should be displayed. The displayed maze on the generation screen should match the maze displayed when it is generated.	Pass	Video - 1:23 Video - 3:54 ***Figure 1*** (in the testing section)
Objectives 3 - Database					
33	**(3.1.)** A test to show that the database consists of three tables.	N/A	On the left hand side of the SQyog under "pathfindingdatabase" there should be three tables listed.	Pass	Video - 4:07 
34	**(3.1.1.)** A test to show that the following data (that is used to load a maze) is stored in a table in the SQL database: mazeID, mazeName, mazeWidth, mazeHeight, startSquareX, startSquareY. endSquareX, endSquareY, wallStatus	N/A	The data about each process should be stored in a table in the SQL database. The data that should be stored are: mazeID, mazeName, mazeWidth, mazeHeight, startSquareX, startSquareY. endSquareX, endSquareY, wallStatus	Pass	Video - 4:12
35	**(3.1.2.)** A test to show that the following data is stored in a table in the SQL database: algorithmID, algorithmName	N/A	The data about each process should be stored in a table in the SQL database. The data that should be stored are: algorithmID, algorithmName	Pass	Video - 4:07

36	**(3.1.3.)** A test to show that the following data is stored in a table in the SQL database: processID, mazeID, algorithmID, timeTakenToRun, pathToEnd, numberOfSquaresTraversed	N/A	The data about each process should be stored in a table in the SQL database. The data that should be stored are: processID, mazeID, algorithmID, timeTakenToRun, pathToEnd, numberOfSquaresTraversed	Pass	Video - 4:23
37	**(3.2.1.)** To show that information used to load the maze, information about algorithms and about previously run processes are all stored in a database (an SQL database).	N/A	Information used to load the maze, information about algorithms and about previously run processes should all be stored in the SQL database (in different tables).	Pass	Video - 4:08 Video - 4:12 Video - 4:23
38	**(3.3.1.)** The data stated in the objectives (3.1.3.4.), (3.1.3.5.), (3.1.3.6.) should be accessible from within the program by the user. This test shows whether that is the case.	The button “Compare Data” is clicked.	The program should retrieve data from multiple data and load the required information. The information loaded must include: the time taken to run the program, the path from the start square to the end square and the number of squares traversed.	Pass (despite the output time may not be correct, a time is still outputted).	Video - 5:00
39	**(3.3.2.)** The data should appear and be accessible from the tables in the (SQL) database. This test shows whether that is the case.	N/A	The data about the processes (the algorithms run on a maze) should be saved and displayed in the tables within the SQL database.	Pass	Video - 4:23
Objectives 4 - Malicious					
40	**(4.1.)** To show that the program has an instruction screen that it can successfully display to the user.	“Instruction” button clicked	Instruction screen should open and be displayed to the user.	Pass	Video - 0:08
Whole System Tests					
41	A test to show that multiple mazes can be generated and saved successfully without needing to close the program and reload it.	Multiple mazes should be generated and saved using valid inputs by the user.	Each maze should be saved successfully in the “mazeinfo” table within the SQL database (“pathfindingdatabase”).	Pass	Video - 3:12 (“maze” is generated) Video - 2:03 (“maze2” is generated) Video - 2:25 (“small” is

					generated) Video - 3:10 ("big" is generated) Video - 4:12 (shows that they are all saved in the database)
42	A test to show that if the user chooses to go back to the previous screen (after generating a maze) in order to change values / information that is used to generate a maze, a different maze is generated.	The user presses the "Back" button after having generated a maze, then they change a/some input(/s) e.g. change the end square's coordinates to (5,10). Then they press the "Accept" to generate a new maze.	The first version of the maze ("maze") being generated and the maze generated after information is changed, in this case the end square's coordinates are changed to (5,10), should be different. Therefore, the displayed representations of the mazes should not be the same.	Pass	Video - 1:21 (first version) Video - 1:31 (second version)
43	The purpose of this test is to see whether the data displayed in the table on the compare data screen updates after more algorithms are run on a maze (so there become more processes).	The user clicks the "Compare Data" button after running some algorithms on a maze, they then return to run more algorithms on the maze and press the "Compare Data" button again.	The new processes should be outputted in the table on the compare data screen, along with the previous ones.	Pass	Video - 5:57 Video - 6:10
44	The purpose of this test is to show that a maze of maximum size (30x30) can be successfully fully displayed (once it has been generated).	The width and height are both inputted as "30", the other inputs inputted must be valid in order to successfully generate a maze.	The maze (of size 30x30) should be fully displayed on the screen (without any of it falling outside the boundaries of the window).	Pass	Video - 3:14
45	The purpose of this test is to show that the program successfully loads the information about processes from the "algorithmsrun"	The "Compare Data" button is pressed after some algorithms are run on the	The information that is stored in the SQL database after an algorithm is run on a maze, should be successfully loaded and outputted.	Pass	Video - 5:03 Video - 6:19

	table (from the SQL database) and outputs them on the compare data (import maze) screen.	imported maze.			
46	The purpose of this test is to show that the program can successfully access information from multiple linked tables.	The “Compare Data” button is pressed after some algorithms are run on the imported maze.	Using each algorithms’ ID the algorithm’s name should be accessed from the table “typesofalgorithms” and outputted for each record.	Pass	Video - 4:31 Video - 5:03
47	A test to show that the back button on the instruction screen works how it should.	The “Back” button is pressed (on the instructions screen).	The instructions screen should close, leaving the main menu screen open.	Pass	Video - 0:14
48	A test to show that the back button on the import maze screen (before a maze is imported using an ID) works how it should.	The “Back” button is pressed (on the import maze screen).	The import maze screen should close, leaving the main menu screen open.	Pass	Video - 1:49
49	A test to show that the back button on the import maze screen (after a maze is imported using an ID) works how it should.	The “Back” button is pressed (on the import maze screen).	The import maze screen should close, leaving the main menu screen open.	Pass	Video - 1:50
50	A test to show that the back button on the import maze screen (when the compare data screen is loaded) works how it should.	The “Back” button is pressed (to return from the compare data screen to the run algorithms one)	The import maze screen (the screen that had the compare data screen loaded on it), should be cleared and the screen that allows the user to run algorithms on the imported maze should be loaded.	Pass	Video - 5:19
51	A test to show that by pressing the “Exit” button the program successfully closes.	The user presses the “Exit” button.	The pathfinding comparison program should close successfully without crashing.	Pass	Video - 6:15

4.5 Extension Tasks (that I did not complete)

Despite having completed the objectives that weren't extension tasks as well as some extension tasks, I didn't manage to complete all the extension tasks in the given time.

(3.1.3.7.) Before attempting to make my program measure/store the amount of memory used to run a pathfinding algorithm on maze, I realised that it would be tricky to achieve. As well as that, it could potentially affect the time by slowing down the time it takes for a pathfinding algorithm to be run on a maze, and the time is more important to measure than the memory used for my investigation as when a car goes from point A to B in the shortest possible time, it is the time that matters and not the memory being used. Even if I did want to go through with the idea of measuring the amount of memory used, I likely wouldn't be able to measure it within VB (Visual Basic).

(2.5.2.) (2.5.3.) Although I did output the maze visually, I never made the program visually show each square gets traversed or visually show the shortest path from the start square to the end square. As I am investigating how the arrival speed of vehicles could be made quicker through investigating pathfinding algorithms, displaying the squares being traversed or shortest path wasn't necessary for my program to do. Therefore, as I had a deadline I focused on getting the program to fulfill the important objectives and not the extension objectives.

(2.6.) My investigation may have been more beneficial if I let the user choose to make the mazes have multiple solutions or just one. However, as I mentioned earlier, I focused on getting the program to fulfill the main objectives properly and not the extension objectives.

---

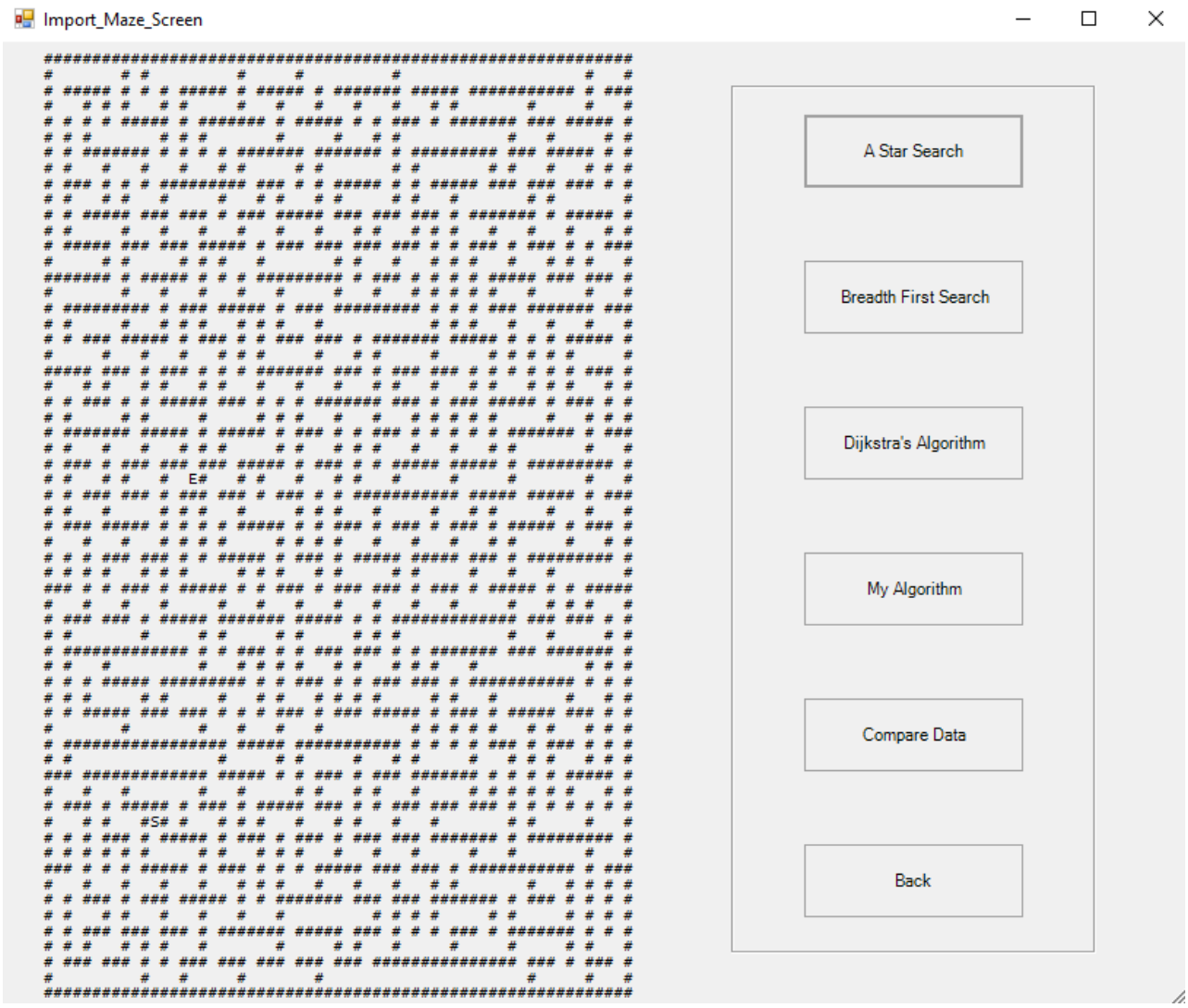
5 Evaluation

5.1 Overall Effectiveness Of System

Overall, my program isn't that effective as although it can successfully run algorithms on mazes, it cannot measure the time it takes to finish running the algorithm successfully. The time measured in microseconds is sometimes 0 which isn't correct, but sometimes the measured time is correct. For my investigation, in order to see how the arrival speed of vehicles between two points can be made quicker through investigating pathfinding algorithms to see which can find the shortest route in the shortest time, I require accurate measurements of how long a pathfinding algorithm took to run.

After running the algorithms on mazes, I noticed that the results/time measure were more believable/reliable than the results measured/taken during the recording of the video (for the testing section). The reason likely being because the software I used to record the testing video probably slowed down the time it took for algorithms to run on the mazes, as both the programs had to compete for resources. However, even now there are some anomalies when measuring the time it takes for an algorithm to run on a maze and the program still sometimes measures the timeTaken as 0, so the program isn't 100% reliable but it is definitely more reliable than I thought before.

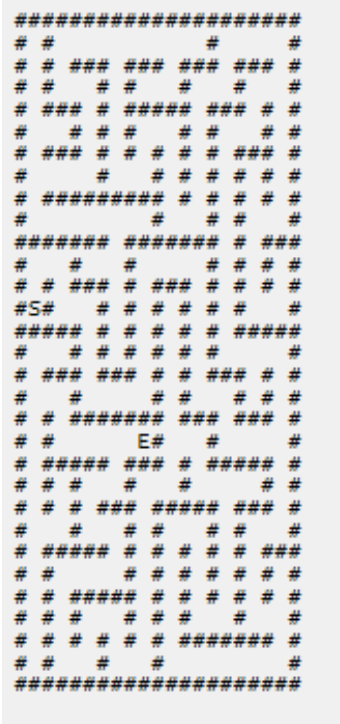
After conducting an investigation using my program, I noticed that on larger mazes (e.g. that have dimensions of 30x30) with only one solution, the A* Search and my algorithm run much more quickly than Breadth-First Search and Dijkstra's Algorithm. Despite my algorithm being the quickest at running on large mazes, the difference between the numberOfSqauresTraversed measured in other algorithms compared to mine was big (as seen in the images on the next couple pages) which suggests that there may be an issue with the algorithm I created, such as the algorithm not always finding the shortest possible paths in a maze with multiple solutions. If I had implemented the feature of letting the user choose to generate a maze with one or two solutions, I could have tested whether my algorithm still works. In real life vehicles can travel on a variety of different roads in order to get from point A to B, so if my algorithm may not be able to always find a/the shortest path or isn't reliable, it is safer to say that it is not the best algorithm to use. Therefore, as my own algorithm may not always work properly, through conducting my investigation I have found out that the A* Search is the best to run on large mazes as it is the quickest on large mazes despite its complexity. The evidence for this is conveyed in the images on the next pages.



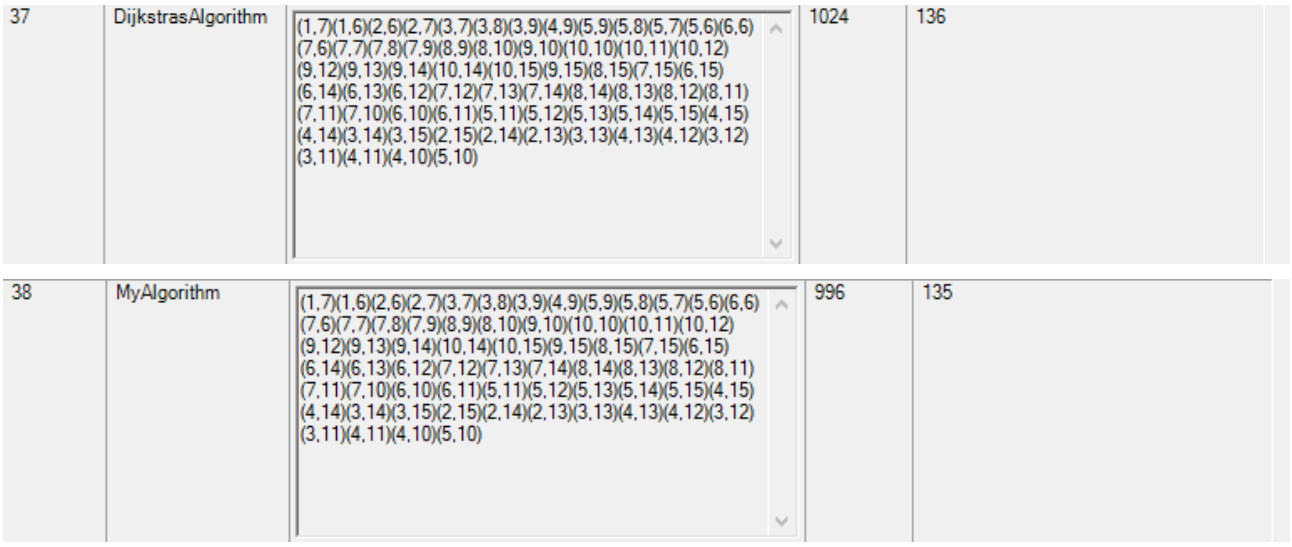
processID	algorithm	path	timeTaken	numberOfSquaresTraversed
19	AStarSearch	(6,25)(6,26)(7,26)(8,26)(8,27)(7,27)(7,28)(8,28)(8,29)(7,29) (7,30)(6,30)(6,29)(6,28)(5,28)(5,27)(5,26)(5,25)(4,25)(4,24) (3,24)(3,25)(3,26)(3,27)(4,27)(4,28)(4,29)(3,29)(3,30)(2,30) (1,30)(1,29)(1,28)(1,27)(2,27)(2,26)(2,25)(1,25)(1,24)(2,24) (2,23)(3,23)(4,23)(5,23)(6,23)(7,23)(8,23)(9,23)(9,24) (10,24)(10,25)(10,26)(11,26)(11,27)(11,28)(12,28)(12,27) (12,26)(12,25)(13,25)(13,26)(13,27)(14,27)(14,26)(15,26) (15,25)(14,25)(14,24)(14,23)(15,23)(16,23)(16,24)(16,25) (16,26)(17,26)(17,27)(18,27)(18,28)(18,29)(17,29)(17,30) (18,30)(19,30)(20,30)(21,30)(22,30)(23,30)(24,30)(25,30) (25,29)(26,29)(27,29)(27,30)(28,30)(28,29)(28,28)(28,27)	19947	431
20	BreadthFirstSearch	(6,25)(6,26)(7,26)(8,26)(8,27)(7,27)(7,28)(8,28)(8,29)(7,29) (7,30)(6,30)(6,29)(6,28)(5,28)(5,27)(5,26)(5,25)(4,25)(4,24) (3,24)(3,25)(3,26)(3,27)(4,27)(4,28)(4,29)(3,29)(3,30)(2,30) (1,30)(1,29)(1,28)(1,27)(2,27)(2,26)(2,25)(1,25)(1,24)(2,24) (2,23)(3,23)(4,23)(5,23)(6,23)(7,23)(8,23)(9,23)(9,24) (10,24)(10,25)(10,26)(11,26)(11,27)(11,28)(12,28)(12,27) (12,26)(12,25)(13,25)(13,26)(13,27)(14,27)(14,26)(15,26) (15,25)(14,25)(14,24)(14,23)(15,23)(16,23)(16,24)(16,25) (16,26)(17,26)(17,27)(18,27)(18,28)(18,29)(17,29)(17,30) (18,30)(19,30)(20,30)(21,30)(22,30)(23,30)(24,30)(25,30) (25,29)(26,29)(27,29)(27,30)(28,30)(28,29)(28,28)(28,27)	26926	590
22	DijkstrasAlgorithm	(6,25)(6,26)(7,26)(8,26)(8,27)(7,27)(7,28)(8,28)(8,29)(7,29) (7,30)(6,30)(6,29)(6,28)(5,28)(5,27)(5,26)(5,25)(4,25)(4,24) (3,24)(3,25)(3,26)(3,27)(4,27)(4,28)(4,29)(3,29)(3,30)(2,30) (1,30)(1,29)(1,28)(1,27)(2,27)(2,26)(2,25)(1,25)(1,24)(2,24) (2,23)(3,23)(4,23)(5,23)(6,23)(7,23)(8,23)(9,23)(9,24) (10,24)(10,25)(10,26)(11,26)(11,27)(11,28)(12,28)(12,27) (12,26)(12,25)(13,25)(13,26)(13,27)(14,27)(14,26)(15,26) (15,25)(14,25)(14,24)(14,23)(15,23)(16,23)(16,24)(16,25) (16,26)(17,26)(17,27)(18,27)(18,28)(18,29)(17,29)(17,30) (18,30)(19,30)(20,30)(21,30)(22,30)(23,30)(24,30)(25,30) (25,29)(26,29)(27,29)(27,30)(28,30)(28,29)(28,28)(28,27)	28951	590

23	MyAlgorithm	(6,25)(6,26)(7,26)(8,26)(8,27)(7,27)(7,28)(8,28)(8,29)(7,29) (7,30)(6,30)(6,29)(6,28)(5,28)(5,27)(5,26)(5,25)(4,25)(4,24) (3,24)(3,25)(3,26)(3,27)(4,27)(4,28)(4,29)(3,29)(3,30)(2,30) (1,30)(1,29)(1,28)(1,27)(2,27)(2,26)(2,25)(1,25)(1,24)(2,24) (2,23)(3,23)(4,23)(5,23)(6,23)(7,23)(8,23)(9,23)(9,24) (10,24)(10,25)(10,26)(11,26)(11,27)(11,28)(12,28)(12,27) (12,26)(12,25)(13,25)(13,26)(13,27)(14,27)(14,26)(15,26) (15,25)(14,25)(14,24)(14,23)(15,23)(16,23)(16,24)(16,25) (16,26)(17,26)(17,27)(18,27)(18,28)(18,29)(17,29)(17,30) (18,30)(19,30)(20,30)(21,30)(22,30)(23,30)(24,30)(25,30) (25,29)(26,29)(27,29)(27,30)(28,30)(28,29)(28,28)(28,27)	16982	381

From the results of my investigation, I also found that on smaller/medium mazes (e.g. that have dimensions of 10x15) with only one solution, Breadth-First Search and my algorithm are the two quickest. However, as I have established (in the paragraph before) that my algorithm may not be reliable at always finding the shortest possible path (in mazes with multiple solutions); it is better to say that Breadth-First Search is best to use on small/medium sized mazes as it is reliable and the quickest (excluding my algorithm). The evidence for this is conveyed in the images on the right, below and on the next page.



processID	algorithm	path	timeTaken	numberOfSquaresTraversed
35	AStarSearch	(1,7)(1,6)(2,6)(2,7)(3,7)(3,8)(3,9)(4,9)(5,9)(5,8)(5,7)(5,6)(6,6) (7,6)(7,7)(7,8)(7,9)(8,9)(8,10)(9,10)(10,10)(10,11)(10,12) (9,12)(9,13)(9,14)(10,14)(10,15)(9,15)(8,15)(7,15)(6,15) (6,14)(6,13)(6,12)(7,12)(7,13)(7,14)(8,14)(8,13)(8,12)(8,11) (7,11)(7,10)(6,10)(6,11)(5,11)(5,12)(5,13)(5,14)(5,15)(4,15) (4,14)(3,14)(3,15)(2,15)(2,14)(2,13)(3,13)(4,13)(4,12)(3,12) (3,11)(4,11)(4,10)(5,10)	1002	81
36	BreadthFirstSearch	(1,7)(1,6)(2,6)(2,7)(3,7)(3,8)(3,9)(4,9)(5,9)(5,8)(5,7)(5,6)(6,6) (7,6)(7,7)(7,8)(7,9)(8,9)(8,10)(9,10)(10,10)(10,11)(10,12) (9,12)(9,13)(9,14)(10,14)(10,15)(9,15)(8,15)(7,15)(6,15) (6,14)(6,13)(6,12)(7,12)(7,13)(7,14)(8,14)(8,13)(8,12)(8,11) (7,11)(7,10)(6,10)(6,11)(5,11)(5,12)(5,13)(5,14)(5,15)(4,15) (4,14)(3,14)(3,15)(2,15)(2,14)(2,13)(3,13)(4,13)(4,12)(3,12) (3,11)(4,11)(4,10)(5,10)	998	136



37	DijkstrasAlgorithm	(1,7)(1,6)(2,6)(2,7)(3,7)(3,8)(3,9)(4,9)(5,9)(5,8)(5,7)(5,6)(6,6) (7,6)(7,7)(7,8)(7,9)(8,9)(8,10)(9,10)(10,10)(10,11)(10,12) (9,12)(9,13)(9,14)(10,14)(10,15)(9,15)(8,15)(7,15)(6,15) (6,14)(6,13)(6,12)(7,12)(7,13)(7,14)(8,14)(8,13)(8,12)(8,11) (7,11)(7,10)(6,10)(6,11)(5,11)(5,12)(5,13)(5,14)(5,15)(4,15) (4,14)(3,14)(3,15)(2,15)(2,14)(2,13)(3,13)(4,13)(4,12)(3,12) (3,11)(4,11)(4,10)(5,10)	1024	136
38	MyAlgorithm	(1,7)(1,6)(2,6)(2,7)(3,7)(3,8)(3,9)(4,9)(5,9)(5,8)(5,7)(5,6)(6,6) (7,6)(7,7)(7,8)(7,9)(8,9)(8,10)(9,10)(10,10)(10,11)(10,12) (9,12)(9,13)(9,14)(10,14)(10,15)(9,15)(8,15)(7,15)(6,15) (6,14)(6,13)(6,12)(7,12)(7,13)(7,14)(8,14)(8,13)(8,12)(8,11) (7,11)(7,10)(6,10)(6,11)(5,11)(5,12)(5,13)(5,14)(5,15)(4,15) (4,14)(3,14)(3,15)(2,15)(2,14)(2,13)(3,13)(4,13)(4,12)(3,12) (3,11)(4,11)(4,10)(5,10)	996	135

In conclusion, after doing my investigation I have found out that on mazes with one solution my algorithm is the quickest at successfully finding the shortest path between two points. However, as my algorithm may not be reliable as suggested in my findings it may not always find the shortest path on a maze with multiple solutions. As a result of it likely being unreliable, it is unwise to assume/say that my algorithm is the fastest. Therefore, from my investigation I found out that on larger mazes the A* Search is the best to use as it is the quickest (excluding my algorithm). Whereas on smaller/median sized mazes Breadth-First Search is the best to use as it runs the quickest (excluding my algorithm).

5.2 Evaluation Of Objectives

The pathfinding comparison program that I created fulfilled all of the objectives I created. However, it didn't fulfill all of the extension objectives. Below in the table I have evaluated how well each objective (groups) have been fulfilled and stated any improvements that could be made. The **evidence** that the objectives have been fulfilled can be found in the **testing section** of the document.

Objective Number	Objective	How well its met, 1 (badly) - 5 (best)	Improvements that could be made to fulfill objective(s)
1.1.	1.1. The user can should be able to choose from the following pathfinding algorithms: 1.1.1. A* Search 1.1.2. Breadth-First Search 1.1.3. Dijkstra's algorithm *1.1.4. Extension: An algorithm that I create*	5	N/A The objective(s) is(/are) met extremely well, as the program outputs all the required possible choices on buttons that can be pressed by the user to run the selected algorithm on a maze. Furthermore not only do each of the algorithms run, they also run the intended algorithm, that they are meant to run, properly. Therefore these key objectives have definitely been fulfilled. Evidence: tests 13-17.
1.2.	The possible algorithms should be displayed clearly in the console *(or on an interface)* so that the user knows the possibilities.	5	N/A The objective is met extremely well, as all the possible options the user can choose from are displayed clearly on the interface. Evidence: test 13.
2.1. 2.1.1.	2.1. The user should be able to choose from different ways of creating/generating/accessing a maze to represent roads: 2.1.1. The option to randomly generate a maze.	5	N/A The objectives are met extremely well, as on the main menu screen the user can choose to generate or import a maze. Evidence: test 18.
2.1.1.1.	User should enter a name for the maze	5	Currently the maze's name only has one purpose which is if the user forgets the ID of a maze, they can manually find the ID from the SQL database (mazeinfo) by looking at mazes with the name they gave to the maze (as the name given isn't unique only the mazeID is). To improve the program the name of a maze could be given more of a purpose, e.g. the user could search for mazes with a specific name within the

			program to get their ID. However, the objective itself is still fulfilled as the program prompts the user to enter a name for the maze (tests 19 & 20).
2.1.1.2. 2.1.1.3.	2.1.1.2. The user should be asked to enter the dimensions of the maze they want to create. 2.1.1.3. There will be a size limit so that the maze cannot have a size which causes the program to crash or not work. 2.1.1.3.1. The maze will have a limit to its dimensions which will be: 2.1.1.3.1.1. A minimum of 2x2 (so that the maze being generated won't just consist of the start and end squares). 2.1.1.3.1.2. A maximum of 30 x 30 squares (to avoid a stack overflow error from occurring and causing the program to crash).	5	N/A The objectives are met extremely well, as the program prompts the user to enter the dimensions for the maze. Also, the objective is well met due to the error handling I implemented as conveyed in the tests 21, 22, 23 (in the testing section of the document).
2.1.2.	2.1.2. The option to import previously saved mazes from a SQL database. 2.1.2.1. The user should be asked to enter the ID of the maze they want to import from a SQL database. 2.1.2.2. If the file doesn't exist, then the user should be able to go back to the previous options and choose to generate a maze or attempt to import another maze instead.	5	N/A The objective(s) is(/are) met extremely well, as the user can choose to import a maze, if they do they are prompted to enter the ID of the maze in order to import it from the SQL database. The program will continue to allow the user to enter another ID. However, if they are unable/don't choose to enter a valid ID they can press the back button and choose to generate a maze instead, so they won't get stuck on that screen. Evidence: tests 24 & 25.
2.2.	Mazes will be represented as an adjacency matrix. But they will also be represented/displayed as a grid (2D array), where each coordinate of a grid will represent a node/square/wall in the maze.	5	N/A The objective is met extremely well, as the maze is represented as both an adjacency matrix and as a 2D array by the program. There is no visual evidence for the program representing the maze as an adjacency matrix as it is only needed to run the algorithms on mazes. Whereas the 2D array (grid) is displayed to the user interface to show the user what the maze looks like. Evidence for this in test 26.

2.3. 2.3.1.	2.3. The user should be able to save mazes so that the same / a different algorithm can be run on the same maze. 2.3.1. The maze and its information should be saved in an SQL database, the maze's walls' location will be saved through using a list of 1s and 0s. The walls of the maze will be represented as a 1 and empty spaces/traversable squares as 0s.	5	N/A The objectives are met extremely well, as the information about the maze is stored in the mazeinfo table within the SQL database. Also, the mazes' walls are stored using 1s and 0s as stated/required by the objective. Evidence: tests 27 & 28.
2.3.2.	Once a maze is saved the maze's ID should be outputted to the user so that they can load the maze	5	N/A The objective is met extremely well, as the maze ID is outputted clearly using a pop-up (message box) which requires a user interaction to close, so the user can't accidently miss the outputted ID. The user can use the outputted ID to import/load the same maze that they generated. Evidence: test 29.
2.3.3.	The maze should load in less than 10 seconds, in order to keep the program relatively efficient.	5	N/A The objective is met extremely well, as the program loads a maze quite quickly and in less than 10 seconds. Evidence: test 30.
2.4.	2.4. The user should be able to enter coordinates for the …(insert from below)... to determine their position within the maze (this would be before the maze is generated) (they cannot have the same coordinates): 2.4.1. Start square/node 2.4.2. Target/End square/node	5	N/A The objective(s) is(/are) met extremely well, as the user can enter (x & y) coordinates for both the start and end square into clearly labelled textboxes. In addition, I have implemented error handling that allows the program to only accept valid inputs for the coordinates. Furthermore, I created a custom exception "SameCoordinates" that is thrown if the user inputs the same coordinates for both the start and square. Evidence for objective being fulfilled is shown in test 31.
2.5.	*2.5. Extension: The maze will be visually displayed* *2.5.1. Extension: The maze itself, the start and target/end square/nodes and the walls will be visually displayed.* *2.5.2. Extension: How the pathfinding algorithm traverses each square/node will be visually displayed.*	3	Not all the objectives here were completed, but as they are extension objectives it isn't an issue. However, the program could be improved by completing the objectives: (2.5.2.), (2.5.3.). The evidence to show that the extension objective (2.5.1.) has

	2.5.3. Extension: The shortest route from the starting square/node to the end/target square/node will be visually displayed.		been successfully fulfilled can be found in test 32.
2.6.	*Extension: The user can choose whether they want the maze to have multiple solutions or just one.*	1	This extension objective wasn't fulfilled but if it was it would have allowed me to investigate whether the amount of solutions affects which algorithm is the fastest to use to get the shortest path.
3.1. 3.1.1.	3.1. The database should have three tables: 3.1.1. A table that stores information to load a maze, should store the following information: 3.1.1.1. mazeID 3.1.1.2. mazeName 3.1.1.3. mazeWidth 3.1.1.4. mazeHeight 3.1.1.5. startSquareX (the x value of the start square) 3.1.1.6. startSquareY (the y value of the start square) 3.1.1.7. endSquareX (the x value of the end square) 3.1.1.8. endSquareY (the y value of the end square) 3.1.1.9. wallStatus (will store information about whether each square in the maze is a wall or not, 1 if wall and 0 if not)	5	N/A The objectives are met extremely well, as the database has three tables. The mazeinfo table stores the required information stated in the objectives. Evidence: tests 33 & 34.
3.1.2.	3.1.2. A table that maps each pathfinding algorithm to an associated ID, should store the following information: 3.1.2.1. algorithmID 3.1.2.2. algorithmName	5	N/A The objectives are met extremely well, as the table typesofalgorithms stores the required information stated in the objectives. Evidence: test 35.
3.1.3.	3.1.3. A table that stores information about the maze when an algorithm was run on it should store the following information: 3.1.3.1. processID 3.1.3.2. mazeID 3.1.3.3. algorithmID 3.1.3.4. timeTakenToRun - The time taken for the pathfinding algorithm to be complete (in milliseconds). 3.1.3.5. pathToEnd - The shortest route (using coordinates of squares), from the starting square/node to the end/target square/node. 3.1.3.6. numberOfSquaresTraversed, which is the number of squares explored in the grid	4	The objectives are met quite well, as the table algorithmsrun stores the required information stated in the objectives. Evidence: test 36. However, currently the measured timeTakenToRun doesn't seem to be extremely reliable as sometimes "0" microseconds is recorded. Therefore to completely fulfill the objective (3.1.3.4.), the timeTakenToRun would need to be made more reliable.

	(which represents a maze) before the end/target square/node was found. *3.1.3.7. Extension: The amount of memory used before the pathfinding algorithm has finished running.*		
3.2.	3.2. The information should be stored in the following format: 3.2.1. In a database	5	N/A The objectives are met extremely well, because the database is stored using an SQL database as required in the objectives. Evidence: test 37.
3.3.	3.3. The information/content should be accessible from: 3.3.1. Within the program (only the data: 3.1.3.4. , 3.1.3.5. , 3.1.3.6.) 3.3.2. From the database it is stored in	5	N/A The objective(s) is(/are) met extremely well, as the required data can be accessed by users from both within the program and manually from the SQL database. Evidence for this can be found in the tests 38 & 39.
4.1.	The program should have an instructions screen that summarises important information on how to use the program.	5	N/A The objective is met extremely well, as the user can click on the “Instructions” button (on the main menu screen) to get a short description of how the program works and what they can do with it. The evidence for this can be found in the 40th test of the testing section of the document.

5.3 Expert Feedback/Evaluation

In order to help determine the success and any improvements that can be made to the program I created for my investigation, I contacted my expert (Nyki) to gain individual feedback.

Feedback/Evaluation from Nyki (the expert):
Me: in italics
Nyki: in bold italics

Nyki please can you watch the following video that shows how my (NEA) program works and how it allows for the comparison of pathfinding algorithms: https://youtu.be/mggdXntCx_4

1. *In your opinion, what improvements could I make to the program if I had time?*
2. *Do you think that my program successfully allows for the comparison of pathfinding algorithms? If so/not, why(/ not)?*

Please can you also write a critical evaluation of my pathfinding algorithm comparison program.

Firstly, to answer the two questions:

1. ***When you select each algorithm on an imported maze, it would be good to have a pop-up window to say it has been completed - currently it is only once you compare them that you can see it has been successful. A visual image of the path found in the maze picture would also be a nice additional feature***
2. ***I believe that this program gives all the comparison information you might want in terms of speed of solving and number of tiles traversed. You could go one step further and consider the complexity of the algorithm when programming and if that factor makes any difference to the choice.***

Evaluation:

The program GUI is very easy to use and all the possible user errors have been considered. There is no ambiguity when a user is asked for an input and this makes it suitable for all users when generating a maze. However, in the comparison stage there is no explanation of what 'tiles traversed' means and so this makes interpreting the results difficult for a user without knowledge of algorithmic terminology. A simple fix here could be a description or key when using this aspect of the interface. In addition, it would be helpful to have a condition set for the optimal algorithm based on the results, which could then inform the user about which one is recommended.

Overall, the program is well structured and allows multiple users to access it's features regardless of prior knowledge. It certainly achieves what it was designed for and uses various programming techniques.

From the feedback I have gained from my expert (Nyki), I have learnt there are lots of possible improvements that could be made to improve my program. Nyki said that it would have been beneficial if my program gave some kind of visual prompt when my program successfully runs an algorithm on a

maze. Nyki suggested the idea of using a pop-up (message box) which I used to output the maze ID of a maze as a visual prompt. She also said that I could have visually shown the shortest route/path found, this itself would have been enough of a visual prompt to convey that an algorithm ran successfully on a maze. Originally I did plan on giving a visual prompt to the user when an algorithm finishes running successfully on a maze, however as it was an extension objective **(2.5.3.)** and I ran out of time, so I didn't add it as a feature to my program. If I was to implement the feature of having pop-up, it would make running an algorithm time-consuming and tedious for the user because they would need to close it/click OK each time they ran an algorithm. So if I was to continue improving my program I would avoid using too many pop-ups. As a result, I would likely implement the program to output the shortest path found on the maze **(2.5.3.)**, as despite it giving a visual prompt the user isn't required to verify that they have seen the visual prompt through interacting with the windows form(/a pop-up).

In the feedback, Nyki said that I have all the required comparison information that I need to compare algorithms in terms of speed **(3,1.3.4.)** or the number of squares traversed **(3,1.3.6.)**, but she suggested that I should consider the complexity of each algorithm, and whether it might be a factor in my investigation. Despite my investigation being mainly about which algorithm can find the shortest route between two points quicker, the complexity of an algorithm could affect the speed of an algorithm, therefore I will take the complexity of the algorithms into consideration when creating a summary of my investigation.

In her critical evaluation of my investigation/project Nyki said that the program's GUI is easy to use and clearly conveys what the user is required to input. I also believe that it is clear to use, as I created textboxes (input boxes) for each required input and stated what had to be inputted in it next to it. Furthermore, Nyki stated that the error handling of my program is thorough as I have considered "all" possible user errors (according to Nyki). After conducting a number of tests (that can be found in the testing section of the document), I couldn't find any errors in my program that I didn't handle and could have caused the program to crash, therefore I also believe that the error handling of my program is thorough.

On the other hand, Nyki said that the comparison stage lacks explanation about how it works therefore it wouldn't be understood by users who don't know the algorithmic terminology. So as a solution to there being a lack of explanation, she said it could be overcome through simply adding an explanation of what 'numberOfSquaresTraversed' means on the windows form when it outputs the data about processes (making the data easier to interpret). I also believe that it would be beneficial to add an explanation to help users with little/no understanding of algorithmic terminology. However, the main purpose of my program is to allow me to conduct an investigation therefore as long I understand not having an explanation isn't too much of an issue. But if I was to continue my programming I would still add a short explanation, as it would be easy to make the change and it would improve my program.

In order to improve my program, Nyki suggested that having a condition set for an optimal algorithm using data gathered from processes, would be helpful. The reason it would be helpful/beneficial, is because then the program would work out and output the quickest/best algorithm for a particular maze without the user needing to manually compare data themselves. If I had more time to work on my program I would listen to Nyki's feedback on how to improve my program, and I would have improved my program. Currently, the users of my program may struggle to compare algorithms manually from the data the program outputs due to their lack of understanding about algorithmic terminology, so by creating a feature that automatically selects and outputs the algorithm that is the best to use on a specific maze would be beneficial. This feature would also be beneficial for me as it would allow me to quickly see a pattern of what algorithms work best on certain types of mazes, and therefore speed up my investigation.

In her evaluation of my program, Nyki also said that my program is well structured and can be accessed by users even if they don't have much knowledge of pathfinding algorithms. In addition to that, she said that my program achieves what it is designed to do through using a variety of programming techniques, which is important as it means that I have successfully fulfilled the main objectives (of my program) and I can use my program to complete my investigation.

5.4 System Improvements

If I had more time or I was to repeat the project, there are a few improvements I could make in order to improve both my program and my investigation. The main thing I would do is to make the measurement of the timeTakenToRun always reliable; I would achieve this by fixing the logic error in my code (likely in the "Maze" class as it is where the timeTaken is measured). Furthermore, as Nyki (my expert) suggested, I would add a short explanation of what 'numberOfSquaresTraversed' means on the windows form when it outputs the data about processes. The explanation for the 'numberOfSquaresTraversed' would say 'the number of squares the pathfinding algorithm has checked to see whether they are the end square'. In addition to that, I would add the units ('microseconds') next to the words 'timeTaken' (on the same windows form screen) to make the user aware of what the timeTaken is being measured in.

After evaluating how effectively objectives have been fulfilled, I realised that despite objective **(2.1.1.1.)** being met extremely well, it has only a little purpose. Therefore, if I was to redo my project/I had more time I would give the name of the maze (which is stored) more of a purpose. At the moment, the maze's name only has one purpose which is if the user forgets the ID of a maze, they can manually find the ID from the SQL database (mazeinfo) by looking at mazes with the name they gave to the maze (as the name given isn't unique only the mazeID is). So if I had time, I could give the name of a maze more of a purpose by implementing a feature that allows users to search for mazes with a specific name within the program in order to get their ID.

As Nyki suggested in her critical evaluation, if I had more time I would implement the feature of outputting the route of the shortest path on the maze displayed on the screen where the user can choose to run algorithms on a maze. Originally I did plan on giving a visual prompt to the user when an algorithm finishes running successfully on a maze, however as I ran out of time and it was an extension objective **(2.5.3.)**, I never added it as a feature to my program.

Currently the program allows users to manually compare data collected from running algorithms on mazes, but according to Nyki it would be beneficial to allow the program to automatically work out which algorithm is the best to run on a specific maze. Nyki suggested that I could make the program decide which algorithm is the best for a maze through using some sort of condition, like which algorithm had the lowest timeTaken. If I was to continue improving my program, the feature that Nyki suggested would be beneficial to include, it could be implemented by making changes to the "Import_Maze_Screen" and "Maze" classes.

After assessing the effectiveness of my program I now know that if I was to redo my project to improve my investigation, I would implement the feature of letting the user choose to create a maze with one or multiple solutions. In order to implement this feature I would need to make changes to the "Maze" and "Maze_Generation_Screen" classes. The reason I didn't implement this feature in my project already is

because it was only an extension objective **(2.6.)** to add this feature to my program, and as I ran out of time I didn't add/implement it. At the time when I created the objectives, I didn't realise how much it could affect the outcome of my investigation. In real life a vehicle can travel from point A to point B using a variety of paths (most of time), therefore my investigation would have been more realistic if I had taken the effect of multiple solutions of a maze into consideration before. If the algorithm I created always successfully finds the shortest path between two points even in a maze with multiple solutions then it is the quickest algorithm out of the four algorithms I implemented in my program. However, as I couldn't test out whether that was the case, to be safe I excluded my algorithm from my findings. As a result, the A* Search algorithm was the quickest at running on large mazes whereas the Breadth-First Search algorithm was the quickest at running on smaller/medium sized mazes.

5.5 Conclusion

Despite the fact that my program & investigation could be improved in a variety of ways, using my program I have still found out that to decrease the timeTaken as much as possible by using a different pathfinding algorithm, depending on the size of a maze(/distance between two places). Therefore, from conducting the investigation of how the arrival speed of vehicles between two points, can be made quicker through investigating pathfinding algorithms and seeing which can find the shortest route in the shortest amount of time, I have found that on small/medium mazes (/distances) Breadth-First Search is the best to use and on large mazes (/distances) A* Search is the best to use.

---

6 Appendix

6.1 Code for Prototype

Prototype Code:

```
Module Module1
    Sub Main() 'error catching/handling needs to be put in place for the entire program !!! , !!! means that I will need to look at / make changes to the code when coding the actual technical solution, ??? that I may need to make changes

        '======== Prompting user to enter information in order to generate a maze ========'

        '=== Maze width & height ==='
        Dim width As Integer
        Dim height As Integer

        Console.WriteLine("Enter the width you want the maze to be") 'prompts the user to enter a width for the maze
        width = Console.ReadLine()
        Console.WriteLine("Enter the height you want the maze to be") 'prompts the user to enter a height for the maze
        height = Console.ReadLine()
        '=========================='

        '=== Coordinates for start & end square ==='
        Dim xValue As Integer 'x value cannot be the same for start square and end square (error handling needs to be put in place)
        Dim yValue As Integer 'y value cannot be the same for start square and end square (error handling needs to be put in place)

        Console.WriteLine("Enter x value for start square") 'input will be required to be between 1 and the maze width (squares with a x coordinate of 0 or a width(/height) + 1 are walls/borders)
        xValue = Console.ReadLine()
        Console.WriteLine("Enter y value for start square") 'input will be required to be between 1 and the maze width (squares with a y coordinate of 0 or a width(/height) + 1 are walls/borders)
        yValue = Console.ReadLine()
        Dim startSqu As New StartSquare(xValue, yValue) 'creating start square

        Console.WriteLine("Enter x value for end square") 'input will be required to be between 1 and the maze width (squares with a x coordinate of 0 or a width(/height) + 1 are walls/borders)
        xValue = Console.ReadLine()
        Console.WriteLine("Enter y value for end square") 'input will be required to be between 1 and the maze width (squares with a y coordinate of 0 or a width(/height) + 1 are walls/borders)
        yValue = Console.ReadLine()
        Dim endSqu As New EndSquare(xValue, yValue) 'creating end square
```

```
        '========================================='

        '=== Name for maze (and its binary file, etc) ==='
        Dim name As String
        Console.WriteLine("Enter a name for the maze") 'prompts the user to enter a name for maze, a check will need to be performed to avoid mazes having the same name !!!
        name = Console.ReadLine()
        '=========================================='


'=============================================================================================='

        '======== Generation of maze ========'
        Dim maze1 As New Maze(name, width, height, startSqu, endSqu) 'creates the maze using the inputs
        maze1.OutputMaze() 'outputs maze visually to the console from 2D array (grid) format for the user to see
        'code for the creation of a binary file and database will likely go in this section !!!
        '==================================='

        maze1.runAlgorthim("aStar") 'runs the aStar search on the maze1 aka the previously generated maze
        'enum for algorithm types, algorithm type (input into separate algorithm type), depending on input asks for required inputs using another procedure???


        Console.ReadKey() 'prevents the program from closing/stopping until the user presses a button

    End Sub

End Module

Public Class Maze

    Private name As String
    Private grid(,) As Square
    Private startSqu As StartSquare
    Private endSqu As EndSquare
    Private distanceToEnd As Integer 'from startSqu to endSqu (estimation)
    Private noOfOperationsToGenerate As Integer 'may remove this variable ???
    Private endFound As Boolean

    Public Sub New(ByVal inName As String, ByVal width As Integer, ByVal height As Integer, ByVal inStartSqu As StartSquare, ByVal inEndSqu As EndSquare) 'creates new Maze (object)

        name = inName '& .txt (binary file for maze will be created after this line)
        ReDim grid(width * 2, height * 2) 'transformations so that walls can be stored in the grid variable
        inStartSqu.ChangeXCord(((inStartSqu.GetCoordinates.xValue) * 2) - 1)
```

```
        inStartSqu.ChangeYCord(((inStartSqu.GetCoordinates.yValue) * 2) - 1)
        startSqu = inStartSqu

        inEndSqu.ChangeXCord(((inEndSqu.GetCoordinates.xValue) * 2) - 1)
        inEndSqu.ChangeYCord(((inEndSqu.GetCoordinates.yValue) * 2) - 1)
        endSqu = inEndSqu

        GenerateMaze()

    End Sub

    Public Sub ResetVisit() 'resets each square to make all of them unvisited
        For i = 0 To grid.GetLength(1) - 1
            For k = 0 To grid.GetLength(0) - 1
                grid(k, i).ResetSquare()
            Next
        Next
    End Sub

    Public Sub ResetVisit(ByVal hCost As Integer) 'resets each square to make all of them unvisited, but
also resets hCost (to parameter input) and gCost to 0
        For i = 0 To grid.GetLength(1) - 1
            For k = 0 To grid.GetLength(0) - 1
                grid(k, i).ResetSquare(hCost)
            Next
        Next
    End Sub

    Public Function GetName() As String 'returns the name of the maze
        Return name
    End Function

    Public Function GetStartSquare() As StartSquare 'returns the start square of the maze
        Return startSqu
    End Function

    Public Function GetEndSquare() As EndSquare 'returns the end square of the maze
        Return endSqu
    End Function

    Sub GenerateWalls(ByVal tempSquare As Square, ByRef approxDistanceToEndCounter As Integer)
'generates walls for the maze, approxDistanceToEndCounter as byref or byval ???
        Randomize() 'needed for random number generation
        Dim randomPosition As Integer

        Dim tempVisitArray(3) As Boolean

        For i = 0 To tempVisitArray.Length - 1 'makes each each member in the array tempVisitArray store
the boolean False
```

```
            tempVisitArray(i) = False
        Next

        tempSquare.Visit() 'visits tempSquare

        If tempSquare.GetCoordinates.xValue = endSqu.GetCoordinates.xValue And
tempSquare.GetCoordinates.yValue = endSqu.GetCoordinates.yValue Then 'if the tempSquare has the
same coordinates as the endSqu then the following code happens:
            distanceToEnd = approxDistanceToEndCounter 'saves the current approxDistanceToEndCounter
as the distanceToEnd
        End If

        'approxDistanceToEndCounter may not work correctly (if more solutions from start to end square
then check if it works correctly) !!! ???
        tempSquare.ChangeGCost(approxDistanceToEndCounter)
        tempSquare.ChangeHCost(approxDistanceToEndCounter)

        For i = 1 To 4 'to reach all possible traversable squares

            Do 'randomly chooses a position from 1, 2, 3, 4 where there will not be a wall
                randomPosition = (Int(Rnd() * (4))) + 1
            Loop Until tempVisitArray(randomPosition - 1) = False 'loops until the program randomly selects
a direction that hasn't been visited yet (the -1 is there as array's start with an index of 0).

            Select Case randomPosition 'different things happens based on which number was randomly
generated
                Case 1 'up
                    If tempSquare.GetCoordinates.yValue - 2 > 0 Then
                        If grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue -
2).GetVisitedStatus = False Then 'new random number if true, backtracking, counter to see if any
squares left
                            grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue -
1).ChangeWallStatus(False)
                            grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue -
2).Visit()
                            approxDistanceToEndCounter += 1
                            GenerateWalls(grid(tempSquare.GetCoordinates.xValue,
tempSquare.GetCoordinates.yValue - 2), approxDistanceToEndCounter)
                        End If
                    End If
                Case 2 'right
                    If tempSquare.GetCoordinates.xValue + 2 < grid.GetLength(0) - 1 Then
                        If grid(tempSquare.GetCoordinates.xValue + 2,
tempSquare.GetCoordinates.yValue).GetVisitedStatus = False Then
                            grid(tempSquare.GetCoordinates.xValue + 1,
tempSquare.GetCoordinates.yValue).ChangeWallStatus(False)
                            grid(tempSquare.GetCoordinates.xValue + 2,
tempSquare.GetCoordinates.yValue).Visit()
                            approxDistanceToEndCounter += 1
```

```
                    GenerateWalls(grid(tempSquare.GetCoordinates.xValue + 2,
tempSquare.GetCoordinates.yValue), approxDistanceToEndCounter)
                End If
            End If
        Case 3 'down
            If tempSquare.GetCoordinates.yValue + 2 < grid.GetLength(1) - 1 Then
                If grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue +
2).GetVisitedStatus = False Then
                    grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue +
1).ChangeWallStatus(False)
                    grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue +
2).Visit()
                    approxDistanceToEndCounter += 1
                    GenerateWalls(grid(tempSquare.GetCoordinates.xValue,
tempSquare.GetCoordinates.yValue + 2), approxDistanceToEndCounter)
                End If
            End If
        Case 4 'left
            If tempSquare.GetCoordinates.xValue - 2 > 0 Then
                If grid(tempSquare.GetCoordinates.xValue - 2,
tempSquare.GetCoordinates.yValue).GetVisitedStatus = False Then
                    grid(tempSquare.GetCoordinates.xValue - 1,
tempSquare.GetCoordinates.yValue).ChangeWallStatus(False)
                    grid(tempSquare.GetCoordinates.xValue - 2,
tempSquare.GetCoordinates.yValue).Visit()
                    approxDistanceToEndCounter += 1
                    GenerateWalls(grid(tempSquare.GetCoordinates.xValue - 2,
tempSquare.GetCoordinates.yValue), approxDistanceToEndCounter)
                End If
            End If
    End Select

    tempVisitArray(randomPosition - 1) = True 'this means that the square in that array with that
position has been visited (when generating the maze)

    noOfOperationsToGenerate += 1 'keeps track of how many this loop was run (so the amount of
times this for loop looped due to recursion + the number of times it looped within each recursion)

  Next

End Sub

Sub AddingValues(ByVal tempX As Integer, ByVal tempY As Integer) 'Adds/saves the start and end
square to the correct coordinates within the maze/grid. It makes squares that should be walls as a
default, Wall, it also makes the border Walls in the grid borders so that the program won't make the wall
boolean false for them later in the program
  Dim tempSquare As New Square(tempX, tempY)
  Dim tempWall As New Wall(tempX, tempY)
```

```
        If tempX = startSqu.GetCoordinates.xValue And tempY = startSqu.GetCoordinates.yValue Then 'compares the x and y values of the temp square and the start square's coordinates to see if they are the same, if so then the square with those coordinates becomes the start square
            grid(tempX, tempY) = startSqu

        ElseIf tempX = endSqu.GetCoordinates.xValue And tempY = endSqu.GetCoordinates.yValue Then 'compares the x and y values of the temp square and the end square's coordinates to see if they are the same, if so then the square with those coordinates becomes the end square
            grid(tempX, tempY) = endSqu

        ElseIf tempY Mod 2 = 0 Or tempX Mod 2 = 0 Then 'makes squares that should be walls as a default, Walls
            If tempY = 0 Or tempX = 0 Or tempY = (grid.GetLength(1) - 1) Or tempX = (grid.GetLength(0) - 1) Then 'sees whether the Wall needs to be a border or not, (when doing .GetLength on a 2D array 0 for x length and 1 for y length)
                tempWall.ChangeBorderStatus(True)
            End If
            grid(tempX, tempY) = tempWall
        Else
            grid(tempX, tempY) = tempSquare
        End If

        If tempX = grid.GetLength(0) - 1 Then 'custom loop, sees whether the tempX integer is equal to the width of the maze/grid (the 2D array)
            tempX = 0 'if so resets the tempX to 0 so that the next row can be generated where tempY increases by one and tempX is 0 again
            If tempY <> grid.GetLength(1) - 1 Then  'checks whether tempY is equal to the height of the maze/grid (the 2D array), if so the generation is complete
                tempY += 1 'if it isn't equal to the height then tempY increments by 1
                AddingValues(tempX, tempY) 'recursion until the generation is complete
            End If
        Else 'else tempX increases by one and recursion until the generation is complete
            tempX += 1
            AddingValues(tempX, tempY)
        End If

    End Sub

    Public Sub GenerateMaze() 'generates the maze so that is ready to have pathfinding algorithms run on it, error handling needed for unsuitable dimensions

        '======== Generation of Start & End Square, Default Walls/Borders ========'

        AddingValues(0, 0)

        '========================================================================='

        Dim approxDistanceToEndCounter As Integer
```

```
        GenerateWalls(startSqu, approxDistanceToEndCounter) 'attempts to randomly generate a path from the start square to the end square

        For i = 0 To grid.GetLength(1) - 1 'uses the distanceToEnd to make hCost for each square correct
            For k = 0 To grid.GetLength(0) - 1
                If grid(k, i).GetNodeStatus = True Then 'not GetWallStatus as walls can become spaces, squares that can be traversed have node = True
                    grid(k, i).ChangeHCost(grid(k, i).GetHCost - distanceToEnd)
                    If grid(k, i).GetHCost < 0 Then
                        grid(k, i).ChangeHCost(grid(k, i).GetHCost * -1)
                    End If
                Else
                    grid(k, i).ChangeGCost(-1)
                    grid(k, i).ChangeHCost(-1)
                End If
                grid(k, i).UpdateSquare()
            Next
        Next

    End Sub

    Public Sub OutputMaze() 'outputs maze visually in the console

        For i = 0 To grid.GetLength(1) - 1
            For k = 0 To grid.GetLength(0) - 1
                grid(k, i).OutputString()
            Next
            Console.WriteLine("")
        Next

    End Sub

    Public Function GetEndDistance() As Integer 'returns an estimation of the distanceToEnd (integer) from the start square to the end square
        Return distanceToEnd
    End Function

    Public Sub ResetForSearch() 'resets all squares in the maze/grid and all other required information that needs to be reset
        For i = 0 To grid.GetLength(1) - 1
            For k = 0 To grid.GetLength(0) - 1
                grid(k, i).ResetSquare()
            Next
        Next
        endFound = False
        ResetAllPaths()
    End Sub
```

```
    Public Sub ResetForSearch(ByVal hCost As Integer) 'resets all squares in the maze/grid and all other required information that needs to be reset, this reset option is used for resetting everything after an A* Search it takes in a parameter value which is used as a reset hCost for all the squares
        'in the future may require to adjust this subroutine as it may not reset everything correctly after running e.g. A* Search once, then running this subroutine and then running the A* Search again. !!! ???
        For i = 0 To grid.GetLength(1) - 1
            For k = 0 To grid.GetLength(0) - 1
                grid(k, i).ResetSquare(hCost)
            Next
        Next
        endFound = False
        ResetAllPaths()
    End Sub

    Public Function CheckNumInList(ByVal list As List(Of Integer), ByVal number As Integer) As Integer 'checks whether the inputted parameter value 'number' is in the inputted parameter value 'list', if so returns True
        For i = 0 To list.Count - 1
            If list(i) = number Then
                Return True
            End If
        Next
        Return False
    End Function

    Public Sub runAlgorthim(ByVal type As String) 'depending on the parameter input runs a certain algorithm
        Dim tempSquare As New Square(-1, -1)
        If type = "aStar" Then 'planning on putting in place enum choices or similar, therefore will need to change !!!
            ResetForSearch()
            aStar(startSqu, tempSquare)
        End If
    End Sub

    Public Structure listHCost 'structure to make one variable (that can be put in e.g. a list) store two pieces information, better solution ???
        Dim contents As Square
        Dim index As Integer
    End Structure

    Public Function BubbleSort(ByRef orderCost() As Square, ByVal fType As Boolean) As Square() 'runs bubble sort on array in parameter input, orders it based on fCost (lowest to highest) if fType = True and on based on hCost (lowest to highest) if ftype = False
        Randomize() 'allows the generation of random number

        Dim temp As Square
        Dim tempList As New List(Of listHCost)
        Dim tempListHCost As listHCost
```

```
    Dim fCostMax As Integer = -1
    Dim randomNumber As Integer
    Dim hCostMax As Integer


    If fType = True Then
        While Not (orderCost(0).GetFCost <= orderCost(1).GetFCost And orderCost(1).GetFCost <=
orderCost(2).GetFCost And orderCost(2).GetFCost <= orderCost(3).GetFCost)
            'error catching needed through the entire program !!! (needed here) !!!
            For i = 1 To orderCost.Length - 1
                If orderCost(i - 1).GetFCost > orderCost(i).GetFCost Then
                    temp = orderCost(i - 1)
                    orderCost(i - 1) = orderCost(i)
                    orderCost(i) = temp
                End If
            Next
        End While

    Else 'this code runs if fType = False, so that subroutine was called to determine which member of
the list that have the same fCost have the lowest hCost, if they both have the same hCost then the
program will randomly choose one of them as the next square
        While Not (orderCost(0).GetHCost <= orderCost(1).GetHCost And orderCost(1).GetHCost <=
orderCost(2).GetHCost And orderCost(2).GetHCost <= orderCost(3).GetHCost)
            'error catching needed through the entire program !!! (needed here) !!!
            For i = 1 To orderCost.Length - 1
                If orderCost(i - 1).GetHCost > orderCost(i).GetHCost Then
                    temp = orderCost(i - 1)
                    orderCost(i - 1) = orderCost(i)
                    orderCost(i) = temp
                End If
            Next
        End While

        For i = 0 To orderCost.Length - 1
            If orderCost(i).GetFCost <> -1 And fCostMax = -1 Then 'only runs when fCostMax = -1 as
fCost is in order from lowest to highest so anything after -1 (the default or the fCost of invalid squares)
will be the lowest fCost, error handling needed for when orderCost(i).GetFCost = -1 !!!
                fCostMax = orderCost(i).GetFCost
                hCostMax = orderCost(i).GetHCost
            ElseIf orderCost(i).GetFCost = fCostMax And orderCost(i).GetHCost < hCostMax Then 'if a
member of orderCost has the same fCost but has lower hCost the following code is run
                hCostMax = orderCost(i).GetHCost
            End If
        Next

        For i = 0 To orderCost.Length - 1 'members of orderCost with the same fCost (lowest) and hCost
(lowest) are put into a list
            If orderCost(i).GetFCost = fCostMax And orderCost(i).GetHCost = hCostMax Then
                tempListHCost.contents = orderCost(i)
```

```
                tempListHCost.index = i
                tempList.Add(tempListHCost)
            End If
        Next

        If tempList.Count > 0 Then 'if two numbers have the same fCost and hCost this code randomly chooses which one should be visited first and puts it at the front of the orderCost array, the positions are swapped if needed
            Dim randomNumberList As New List(Of Integer) 'stores previous random numbers that have been used
            For i = 0 To tempList.Count - 1
                If randomNumberList.Count = 0 Then 'cannot check if a randomNumber is in randomNumberList if the list is still empty
                    randomNumber = (Int(Rnd() * (tempList.Count))) 'chooses random index, if there two numbers same F and H cost then the index 0 or 1 can be chosen randomly
                Else
                    Do
                        randomNumber = (Int(Rnd() * (tempList.Count)))
                    Loop Until CheckNumInList(randomNumberList, randomNumber) = False 'sees whether that index has been used
                End If
                randomNumberList.Add(randomNumber)
                temp = orderCost(i) 'what we want to replace it with
                orderCost(i) = tempList.Item(randomNumber).contents 'stores it in orderCost in position 'i'

                orderCost(tempList.Item(randomNumber).index) = temp
            Next
        End If

    End If
    Return orderCost
End Function

Public Sub ResetAllPaths() 'resets path status of all traversable squares in the maze to false
    For i = 0 To grid.GetLength(1) - 1
        For k = 0 To grid.GetLength(0) - 1
            grid(k, i).ChangePathStatus(False)
        Next
    Next
End Sub

Public Function PreviousSquareCheck(ByVal previousSquare As Square, ByVal nextSquare As Square) As Boolean 'checks whether the chosen nextSquare is the same as the previousSquare
    If previousSquare.GetCoordinates.xValue = nextSquare.GetCoordinates.xValue And previousSquare.GetCoordinates.yValue = nextSquare.GetCoordinates.yValue Then
        Return True
    Else
        Return False
    End If
```

```
    End Function

    Public Sub aStar(ByVal currentSquare As Square, ByVal previousSquare As Square) 'subroutine to run A* Search on maze. Need to modify for mazes with multiple solutions and update square costs if shorter route with less cost available !!! ??? , check if program follows required steps of an A* Search by doing more research!!!
        Dim nextSquare As New Square(-1, -1)
        Dim arrayOfVisited() As Boolean = {False, False, False, False}
        Dim index As Integer = -1

        Dim up As New Square(-1, -1)
        up.ChangeFCost(-1)
        Dim right As New Square(-1, -1)
        right.ChangeFCost(-1)
        Dim down As New Square(-1, -1)
        down.ChangeFCost(-1)
        Dim left As New Square(-1, -1)
        left.ChangeFCost(-1)
        Dim temp As New Square(-1, -1)

        Console.WriteLine("
GCost: " & currentSquare.GetGCost & "
HCost: " & currentSquare.GetHCost & "
FCost: " & currentSquare.GetFCost & "
Coords: " & currentSquare.GetCoordinatesActual.xValue & " " &
currentSquare.GetCoordinatesActual.yValue) 'should it output actual coordinates or grid coordinates? ??? !!!

        currentSquare.ChangePathStatus(True) 'changes path status, could be add to list/array of what square/nodes were traversed ??? !!!

        If currentSquare.GetCoordinates.xValue = endSqu.GetCoordinates.xValue And
currentSquare.GetCoordinates.yValue = endSqu.GetCoordinates.yValue Then 'runs if end square is reached
            endFound = True
            Console.Write("Found end")
        Else
            '========================================================== Checks which direction next node is
            Try 'up
                temp = grid(currentSquare.GetCoordinates.xValue, currentSquare.GetCoordinates.yValue - 2) 'up is subtraction as the smaller the number, the higher in the 2D array / grid

                If grid(currentSquare.GetCoordinates.xValue, currentSquare.GetCoordinates.yValue - 1).GetWallStatus = True Or PreviousSquareCheck(previousSquare, temp) = True Then 'sees whether it can go up, if it cannot it throws an exception, else (when it can go that direction) it saves the temp square as that direction
                    Throw New Exception 'create custom exception here !!!
                Else
```

```
            up = grid(currentSquare.GetCoordinates.xValue, currentSquare.GetCoordinates.yValue - 2)
        End If

    Catch ex As Exception 'change error handling so that it handles specific error differently !!!
        up.ChangeFCost(-1)
        arrayOfVisited(0) = True
    End Try

    Try 'right
        temp = grid(currentSquare.GetCoordinates.xValue + 2, currentSquare.GetCoordinates.yValue)
        If grid(currentSquare.GetCoordinates.xValue + 1,
currentSquare.GetCoordinates.yValue).GetWallStatus = True Or PreviousSquareCheck(previousSquare,
temp) = True Then 'sees whether it can go right, if it cannot it throws an exception, else (when it can go
that direction) it saves the temp square as that direction
            Throw New Exception 'create custom exception here !!!
        Else
            right = grid(currentSquare.GetCoordinates.xValue + 2,
currentSquare.GetCoordinates.yValue)
        End If
    Catch ex As Exception 'change error handling so that it handles specific error differently !!!
        right.ChangeFCost(-1)
        arrayOfVisited(1) = True
    End Try

    Try 'down
        temp = grid(currentSquare.GetCoordinates.xValue, currentSquare.GetCoordinates.yValue + 2)
        If grid(currentSquare.GetCoordinates.xValue, currentSquare.GetCoordinates.yValue +
1).GetWallStatus = True Or PreviousSquareCheck(previousSquare, temp) = True Then 'sees whether it
can go down, if it cannot it throws an exception, else (when it can go that direction) it saves the temp
square as that direction
            Throw New Exception 'create custom exception here !!!
        Else
            down = grid(currentSquare.GetCoordinates.xValue, currentSquare.GetCoordinates.yValue +
2)
        End If
    Catch ex As Exception 'change error handling so that it handles specific error differently !!!
        down.ChangeFCost(-1)
        arrayOfVisited(2) = True
    End Try

    Try 'left
        temp = grid(currentSquare.GetCoordinates.xValue - 2, currentSquare.GetCoordinates.yValue)
        If grid(currentSquare.GetCoordinates.xValue - 1,
currentSquare.GetCoordinates.yValue).GetWallStatus = True Or PreviousSquareCheck(previousSquare,
temp) = True Then 'sees whether it can go left, if it cannot it throws an exception, else (when it can go
that direction) it saves the temp square as that direction
            Throw New Exception 'create custom exception here !!!
        Else
            left = grid(currentSquare.GetCoordinates.xValue - 2, currentSquare.GetCoordinates.yValue)
```

```
            End If
        Catch ex As Exception 'change error handling so that it handles specific error differently !!!
            left.ChangeFCost(-1)
            arrayOfVisited(3) = True
        End Try

        Dim orderCost() As Square = {up, right, down, left} 'each square adjacent to the current square is put in an array

        orderCost = BubbleSort(orderCost, True) 'a bubble sort is run on the array to order the array based on each square's fCost (the ones with a lowest fCost at the front)

        'due to some directions e.g. left not being traversable they have a fCost of -1, so the following section of code is run so that the program chooses the shortest valid direction (as the invalid directions would be at the start of the array as they have the lowest fCost), (may need to clean/neaten this section ??? )
        If orderCost(0).GetFCost < 0 Then
            If orderCost(1).GetFCost < 0 Then
                If orderCost(2).GetFCost < 0 Then
                    If orderCost(3).GetFCost <> -1 Then
                        nextSquare = orderCost(3)
                    End If
                ElseIf orderCost(2).GetFCost = orderCost(3).GetFCost Then 'sees whether these two squares or more members in the array have the same fCost
                    orderCost = BubbleSort(orderCost, False) 'runs bubble sort and orders the squares based on hCost (lowest number to highest), (only the ones with the same lowest fCost)
                    nextSquare = orderCost(0)
                Else
                    nextSquare = orderCost(2)
                End If
            ElseIf orderCost(1).GetFCost = orderCost(2).GetFCost Then 'sees whether these two squares or more members in the array have the same fCost
                orderCost = BubbleSort(orderCost, False) 'runs bubble sort and orders the squares based on hCost (lowest number to highest), (only the ones with the same lowest fCost)
                nextSquare = orderCost(0)
            Else
                nextSquare = orderCost(1)
            End If
        ElseIf orderCost(0).GetFCost = orderCost(1).GetFCost Then 'sees whether these two squares or more members in the array have the same fCost
            orderCost = BubbleSort(orderCost, False) 'runs bubble sort and orders the squares based on hCost (lowest number to highest), (only the ones with the same lowest fCost)
            nextSquare = orderCost(0)
        Else
            nextSquare = orderCost(0)
            index = 0
        End If

        For i = 0 To orderCost.Length - 1 'is this needed ???
```

```
            If orderCost(i).GetFCost <> -1 Then
                arrayOfVisited(i) = False
            End If
        Next

        For i = 0 To orderCost.Length - 1
            If orderCost(i).GetCoordinates.xValue = nextSquare.GetCoordinates.xValue And
orderCost(i).GetCoordinates.yValue = nextSquare.GetCoordinates.yValue Then
                If index = -1 Then
                    index = i
                End If
            End If
        Next

        If nextSquare.GetCoordinates.xValue <> -1 Then 'invalid squares have x & y coordinates of -1
(this section only runs if the next square is valid, if it isn't then there are no more valid squares left)
            While (Not (arrayOfVisited(0) = True And arrayOfVisited(1) = True And arrayOfVisited(2) = True
And arrayOfVisited(3) = True)) And endFound <> True And index <> 4 'unavailable paths are true,
recursion happens until all paths are explored, until the end is not True (False) and the index is not 4, (<>
= not equal to)
                previousSquare = currentSquare
                arrayOfVisited(index) = True
                aStar(nextSquare, previousSquare) 'where the recursion occurs
                index += 1
                If endFound = False And index < 3 Then
                    nextSquare = orderCost(index)
                End If
            End While
        Else 'runs if not path/solution from the start square to the end square is found
            Console.WriteLine("")
            Console.WriteLine("No path found")
            ResetAllPaths()
        End If
    End If
End Sub

End Class

Public Class Square
    Protected gCost As Integer
    Protected hCost As Integer
    Protected fCost As Integer

    Protected visited As Boolean = False
    Protected wall As Boolean = False
    Protected path As Boolean = False
    Protected node As Boolean = True 'all traversable square have node = True, Wall (squares) have
node = False . The reason there is a separate variable called 'node', is so that the program can store
whether a square was originally a transversable Square or a Wall. As you cannot just use the wall
```

```
boolean to determine this as once a Wall (Square) becomes crossable (so there is no longer a wall between two squares), the wall boolean which was originally True becomes False. !!! This variable could be removed if e.g. the program instead checked the type of Square instead of checking whether node = True. ??? (change code)

    Protected startSquare As Boolean = False 'boolean to store whether the square is the start square (if so then True)
    Protected endSquare As Boolean = False 'boolean to store whether the end is the start square (if so then True)

    Public Structure coord 'a structure that stores two integers: xValue and yValue, change this to protected instead of public ???
        Dim xValue As Integer
        Dim yValue As Integer
    End Structure

    'the difference between coordinate and coordinateActual is that coordinate is the coordinates of square in the 2D array (grid) when the wall squares are included and coordinateActual is the actual coordinates of the square when the walls are excluded
    Protected coordinate As New coord
    Protected coordinateActual As New coord

    Public Sub New(ByVal xValue As Integer, ByVal yValue As Integer) 'creates a new square using inputs
        coordinate.xValue = xValue
        coordinate.yValue = yValue
        coordinateActual.xValue = (xValue + 1) / 2
        coordinateActual.yValue = (yValue + 1) / 2
    End Sub

    Public Function GetCoordinates() As coord 'returns coordinates for square in coord structure form
        Return coordinate
    End Function

    Public Sub ChangeXCord(ByVal xValue As Integer) 'changes x cord of a square to input, then applies required transformations
        coordinate.xValue = xValue
        coordinateActual.xValue = (xValue + 1) / 2
    End Sub

    Public Sub ChangeYCord(ByVal yValue As Integer) 'changes y cord of a square to input, then applies required transformations
        coordinate.yValue = yValue
        coordinateActual.yValue = (yValue + 1) / 2
    End Sub

    Public Function GetCoordinatesActual() As coord 'return the actual coordinates for square in coord structure form
        Return coordinateActual
```

```
    End Function

    Public Sub UpdateSquare() 'updates fCost (using new gCost and hCost)
        fCost = gCost + hCost
    End Sub

    Public Sub ResetSquare() 'resets square so that it isn't visited
        visited = False
    End Sub

    Public Sub ResetSquare(ByVal inHCost) 'resets square so that it isn't visited, but all resets hCost to
input hCost and gCost
        visited = False
        hCost = inHCost
        gCost = 0
        UpdateSquare()
    End Sub

    Public Overridable Sub ResetInfo() 'resets information about square but remains visited (is this
needed?)
        startSquare = False
        endSquare = False
    End Sub

    Public Function GetVisitedStatus() As Boolean 'returns the visited status/boolean of square
        Return visited
    End Function

    Public Sub Visit() 'visits the square, represented by changing the visited square to true
        visited = True
    End Sub

    Public Function GetWallStatus() As Boolean 'return whether the square is a wall or not
        Return wall
    End Function

    Public Overridable Sub ChangeWallStatus(ByVal inWall As Boolean) 'changes the wall status of the
square (if overridden)

    End Sub

    Public Overridable Sub OutputString() 'outputs associated string of square,(space outputted if not
overridable)
        Console.Write(" ")
    End Sub

    Public Function GetGCost() As Integer 'return the gCost of associated square
        Return gCost
    End Function
```

```
    Public Function GetFCost() As Integer 'return the fCost of associated square
        Return fCost
    End Function

    Public Function GetHCost() As Integer 'return the hCost of associated square
        Return hCost
    End Function

    Public Sub ChangeGCost(ByVal inGCost As Integer) 'changes gCost to inputted value
        gCost = inGCost
    End Sub

    Public Sub ChangeFCost(ByVal inFCost As Integer) 'changes fCost to inputted value
        fCost = inFCost
    End Sub

    Public Sub ChangeHCost(ByVal inHCost As Integer) 'changes hCost to inputted value
        hCost = inHCost
    End Sub

    Public Function GetPathStatues() As Boolean 'returns whether square is a traversable square
        Return path
    End Function

    Public Sub ChangePathStatus(ByVal inPath As Boolean) 'changes path staus of the associated
square
        path = inPath
    End Sub

    Public Function GetNodeStatus() As Boolean 'get node status of associated square
        Return node
    End Function

End Class

Public Class StartSquare
    Inherits Square 'inherits from Square class

    Public Sub New(ByVal xValue As Integer, ByVal yValue As Integer) 'creates new StartSquare using
parameter inputs
        MyBase.New(xValue, yValue) 'runs the 'New' subroutine in the Square class that creates a new
square
        startSquare = True
    End Sub

    Public Overrides Sub OutputString() 'overrides subroutine in Square class and runs this subroutine
instead
        Console.Write("S")
```

```
    End Sub

End Class



Public Class EndSquare
    Inherits Square 'inherits from Square class

    Public Sub New(ByVal xValue As Integer, ByVal yValue As Integer) 'creates new EndSquare using parameter inputs
        MyBase.New(xValue, yValue) 'runs the 'New' subroutine in the Square class that creates a new square
        endSquare = True
    End Sub

    Public Overrides Sub OutputString() 'overrides subroutine in Square class and runs this subroutine instead
        Console.Write("E")
    End Sub

End Class

Public Class Wall
    Inherits Square 'inherits from Square class

    Protected border As Boolean = False

    Public Sub New(ByVal xValue As Integer, ByVal yValue As Integer) 'creates new Wall using parameter inputs
        MyBase.New(xValue, yValue) 'runs the 'New' subroutine in the Square class that creates a new square
        wall = True
        node = False
    End Sub
    Public Function IsBorder() As Boolean 'returns whether the wall is a border. If it is, then it cannot become a space between two traversable squares that can be crossed
        Return border
    End Function
    Public Sub ChangeBorderStatus(ByVal inBorder As Boolean) 'changes the border status of wall (square) to inputted status
        border = inBorder
    End Sub
    Public Overrides Sub ChangeWallStatus(ByVal inWall As Boolean) 'changes wall status of wall (allowing it to be crossed)
        wall = inWall
    End Sub
```

```
    Public Overrides Sub ResetInfo() 'resets information about associated Wall
        MyBase.ResetInfo()
        wall = True
        border = False
    End Sub

    Public Overrides Sub OutputString() 'overrides subroutine in Square class and outputs a specific string depending on whether the Wall's wall boolean is true or false
        If wall = True Then
            Console.Write("#") 'wall that cannot be crossed
        Else
            Console.Write(" ") 'wall that can be crossed
        End If
    End Sub

End Class
```

6.2 Code for Technical Solution

Technical Solution Code:

6.2.1 Algorithm Class

```
Public Class Algorithm 'stores a collection of different algorithms

    Protected queueOfSquaresToVisit As Queue
    Protected startSqu As StartSquare
    Protected maze As Maze

    Public Enum Algorithms 'algorithm types
        AStar
        BreadthFirst
        Dijkstras
        MyAlgorithm
    End Enum

    Public Sub New(ByVal inputMaze As Maze, ByVal parameterStartSqu As StartSquare) 'creates a new Algorithm object
        startSqu = parameterStartSqu 'stores the parameter input in the startSqu variable
        maze = inputMaze
        queueOfSquaresToVisit = New Queue(maze.GetCollectionOfNodes.Length - 1) 'creates a new queue
    End Sub

    Public Function StartAStarSearch() As List(Of Square) 'a function that returns the shortest path from the start node/square to the end node/square which has been calculated using the A* Search
        queueOfSquaresToVisit.Enqueue(startSqu) 'adds the start square to the queue
        startSqu.Discover() 'makes the start square discovered
```

```
        AStarSearch(0) 'starts the A* Search on the maze
        Return GetPath(maze.GetEndSquare) 'returns the path from the start square to the end square
    End Function

    Public Sub AStarSearch(ByVal distanceFromStart As Integer) 'subroutine that runs an A* Search on the maze
        Dim currentSquare As Square
        Dim neighbours As List(Of Square) 'a list which will store a list of neighbouring/adjacent squares of the current square


        If queueOfSquaresToVisit.IsEmpty = False And maze.IsEndFound = False Then 'sees whether the queue is empty or the end has been found
            currentSquare = queueOfSquaresToVisit.Dequeue() 'gets the square at the front of the priority queue
            currentSquare.Traverse() 'marks the square as traversed

            maze.IncreaseNumberOfSquaresTraversed() 'increments the counter that keeps track of the numberOfSquaresTraversed

            If currentSquare.IsEndSquare = True Then 'checks whether the current square is the end square
                maze.ChangeEndFoundStatus(True) 'the endFound variable is made to be TRUE in the Maze class, to indicate that end square has been found

            Else

                neighbours = maze.GetNeighbours(currentSquare) 'gets a list of the current square's neighbouring/adjacent squares

                distanceFromStart += 1 'increments the counter that stores the distance from the start which is used to calculate the gCost of a square

                For Each square In neighbours 'every square (item) in the neighbours list, has the code in the FOR loop run on it

                    If square.GetTraversedStatus = False And square.GetDiscoveredStatus = False Then 'if GetDiscoveredStatus = True, it means the square is/has already been in queueofSquaresToVisit. GetTraversedStatus = True , means that the square has been fully explored
                        queueOfSquaresToVisit.Enqueue(square) 'adds the neighbouring/adjacent square of current square being traversed to the queue
                        square.Discover() 'marks the neighbouring/adjacent square as discovered, so that they cannot be added to the queue again
                        square.ChangePreviousSquare(currentSquare) 'makes the current square of the neighbour square the previous/parent square
                        square.ChangeGCost(distanceFromStart) 'changes the gCost
                        square.ChangeHCost(maze.GetEndSquare) 'changes the fCost, square is updated automatically once a cost is changed
                    ElseIf square.GetTraversedStatus = False Then 'if a square has been discovered (so found) but not traversed then the following code is run
```

```
                Dim tempSquare As Square 'a tempSquare is used so that if the newly calculated fCost isn't lower than the previous one, nothing is changed. Another reason it is used, is so that the two fCosts can be compared
                tempSquare = square
                tempSquare.ChangePreviousSquare(currentSquare)
                tempSquare.ChangeGCost(distanceFromStart)
                tempSquare.ChangeHCost(maze.GetEndSquare)
                If square.GetFCost > tempSquare.GetFCost Then 'sees whether the new fCost is lower than the current one, if it is lower than it is made to be the fCost of the square by making the square the tempSquare
                    square = tempSquare
                End If
            End If
        Next

        queueOfSquaresToVisit.BubbleSortQueue(Queue.sortByCost.fCost) 'a bubble sort is run on the queue based on the fCost
        AStarSearch(distanceFromStart) 'recursion occurs

      End If

    End If

  End Sub

  Public Function StartBreadthFirstSearch() As List(Of Square) 'a function that returns the shortest path from the start node/square to the end node/square which has been calculated using Breadth First Search
    queueOfSquaresToVisit.Enqueue(startSqu) 'adds the start square to the queue
    startSqu.Discover() 'makes the start square discovered
    BreadthFirstSearch() 'starts Breadth First Search on the maze
    Return GetPath(maze.GetEndSquare) 'returns the path from the start square to the end square
  End Function

  Public Sub BreadthFirstSearch() 'subroutine that runs Breadth First Search on the maze, uses a normal queue as the bubblesort procedure is never run on the queue (which is what makes the queue a priority queue)
    Dim currentSquare As Square
    Dim neighbours As List(Of Square) 'a list which will store a list of neighbouring/adjacent squares of the current square


    If queueOfSquaresToVisit.IsEmpty = False And maze.IsEndFound = False Then 'sees whether the queue is empty, if it isn't than the following code is run

      currentSquare = queueOfSquaresToVisit.Dequeue() 'gets the square at the front of the queue
      'no need to mark a square as traversed as once it is discovered, it is added to the queue and there is no rearrangement of the queue
```

```
        maze.IncreaseNumberOfSquaresTraversed() 'increments the counter that keeps track of the numberOfSquaresTraversed

        If currentSquare.IsEndSquare = True Then 'checks whether the current square is the end square
            maze.ChangeEndFoundStatus(True) 'the endFound variable is made to be TRUE in the Maze class, to indicate that end square has been found

        Else

            neighbours = maze.GetNeighbours(currentSquare) 'gets a list of the current square's neighbouring/adjacent squares

            For Each square In neighbours 'every square (item) in the neighbours list, has the code in the FOR loop run on it
                If square.GetDiscoveredStatus = False Then 'if GetDiscoveredStatus = True, it means the square is in queueofSquaresToVisit
                    queueOfSquaresToVisit.Enqueue(square) 'adds the neighbouring/adjacent square of current square being traversed to the queue
                    square.Discover() 'marks the neighbouring/adjacent square as discovered, so that they cannot be added to the queue again
                    square.ChangePreviousSquare(currentSquare) 'makes the current square of the neighbour square the previous/parent square
                End If
            Next

            BreadthFirstSearch() 'recursion occurs

        End If

    End If
End Sub

Public Function StartDijkstrasAlgorithm() As List(Of Square) 'a function that returns the shortest path from the start node/square to the end node/square which has been calculated using Dijkstra's Algorithm
    queueOfSquaresToVisit.Enqueue(startSqu) 'adds the start square to the queue
    startSqu.Discover() 'makes the start square discovered
    DijkstrasAlgorithm(0) 'starts Dijkstra's Algorithm on the maze
    Return GetPath(maze.GetEndSquare) 'returns the path from the start square to the end square
End Function

Public Sub DijkstrasAlgorithm(ByVal distanceFromStart As Integer) 'subroutine that runs Dijkstra's Algorithm on the maze
    Dim currentSquare As Square
    Dim neighbours As List(Of Square) 'a list which will store a list of neighbouring/adjacent squares of the current square


    If queueOfSquaresToVisit.IsEmpty = False And maze.IsEndFound = False Then
```

```
        currentSquare = queueOfSquaresToVisit.Dequeue() 'gets the square at the front of the priority queue
        currentSquare.Traverse() 'marks the square as traversed

        maze.IncreaseNumberOfSquaresTraversed()  'increments the counter that keeps track of the numberOfSquaresTraversed

        If currentSquare.IsEndSquare = True Then 'checks whether the current square is the end square
            maze.ChangeEndFoundStatus(True) 'the endFound variable is made to be TRUE in the Maze class, to indicate that end square has been found

        Else

            neighbours = maze.GetNeighbours(currentSquare) 'gets a list of the current square's neighbouring/adjacent squares

            distanceFromStart += 1 'increments the counter that stores the distance from the start which is used to calculate the gCost of a square

            For Each square In neighbours 'every square (item) in the neighbours list, has the code in the FOR loop run on it
                If square.GetTraversedStatus = False And square.GetDiscoveredStatus = False Then 'if GetDiscoveredStatuss = True, it means the square is/has already been in queueofSquaresToVisit. GetTraversedStatus = True , means that the square has been fully explored
                    queueOfSquaresToVisit.Enqueue(square) 'adds the neighbouring/adjacent square of current square being traversed to the queue
                    square.Discover() 'marks the neighbouring/adjacent square as discovered, so that they cannot be added to the queue again
                    square.ChangePreviousSquare(currentSquare) 'makes the current square of the neighbour square the previous/parent square
                    square.ChangeGCost(distanceFromStart) 'gCost is the distance from the start, changes the associated gCost of the square

                ElseIf square.GetTraversedStatus = False Then 'if a square has been discovered (so found) but not traversed then the following code is run
                    Dim tempSquare As Square 'a tempSquare is used so that if the newly calculated gCost isn't lower than the previous one, nothing is changed. Another reason it is used, is so that the two gCosts can be compared
                    tempSquare = square
                    tempSquare.ChangePreviousSquare(currentSquare)
                    tempSquare.ChangeGCost(distanceFromStart)

                    If square.GetGCost > tempSquare.GetGCost Then 'sees whether the new gCost is lower than the current one, if it is lower than it is made to be the gCost of the square by making the square the tempSquare
                        square = tempSquare
                    End If

                End If
```

```
            Next

            queueOfSquaresToVisit.BubbleSortQueue(Queue.sortByCost.gCost) 'a bubble sort is run on the queue based on the gCost
            DijkstrasAlgorithm(distanceFromStart) 'recursion occurs

        End If

    End If
End Sub

Public Function StartMyAlgorithm() As List(Of Square) 'a function that returns the shortest path from the start node/square to the end node/square which has been calculated using My Algorithm
    queueOfSquaresToVisit.Enqueue(startSqu) 'adds the start square to the queue
    startSqu.Discover() 'makes the start square discovered
    MyAlgorithm(startSqu) 'starts My Algorithm on the maze
    Return GetPath(maze.GetEndSquare) 'returns the path from the start square to the end square
End Function

Public Sub MyAlgorithm(ByVal previousSquare As Square) 'extension, subroutine runs My Algorithm on the maze
    Dim currentSquare As Square
    Dim neighbours As List(Of Square)

    If queueOfSquaresToVisit.IsEmpty = False And maze.IsEndFound = False Then
        currentSquare = queueOfSquaresToVisit.Dequeue() 'gets the square at the front of the priority queue
        currentSquare.ChangePreviousSquare(previousSquare) 'makes the current square of the neighbour square the previous/parent square
        maze.IncreaseNumberOfSquaresTraversed() 'increments the counter that keeps track of the numberOfSquaresTraversed

        If currentSquare.IsEndSquare = True Then
            maze.ChangeEndFoundStatus(True) 'the endFound variable is made to be TRUE in the Maze class, to indicate that end square has been found
        Else

            neighbours = maze.GetNeighbours(currentSquare) 'gets a list of the current square's neighbouring/adjacent squares


            For Each square In neighbours 'every square (item) in the neighbours list, has the code in the FOR loop run on it
                If square.GetDiscoveredStatus = False Then 'if GetDiscoveredStatus = True, it means the square is in queueofSquaresToVisit
                    queueOfSquaresToVisit.Enqueue(square) 'adds the neighbouring/adjacent square of current square being traversed to the queue
```

```
                square.Discover() 'marks the neighbouring/adjacent square as discovered, so that they cannot be added to the queue again
                MyAlgorithm(currentSquare) 'recursion occurs
            End If
        Next


        End If

    End If
End Sub

Public Sub ResetInformation() 'empties the queue as some elements may still be in it
    While queueOfSquaresToVisit.IsEmpty = False 'until the queue is empty
        queueOfSquaresToVisit.Dequeue() 'every element is dequeued (removed)
    End While
End Sub

Function GetPath(ByVal lastSquare As Square) As List(Of Square) 'returns the path from the start square to the end square (including them) found by the pathfinding algorithm
    Dim path As New List(Of Square)
    Dim pathStack As New StackOfSquares(900) 'I chose to use a stack as it is first in last out so the start square which will be the last on, will be the first off. The maximum possible size is 900 (as the max size of a maze is 30 * 30 (traversable squares) = 900)

    Do
        pathStack.Push(lastSquare) 'the lastSquare is added to the stack
        lastSquare = lastSquare.GetPreviousSquare 'the previous/parent square of the lastSquare is retrieved and made the lastSquare
    Loop Until lastSquare.GetCoordinatesActual.xValue = startSqu.GetCoordinatesActual.xValue And lastSquare.GetCoordinatesActual.yValue = startSqu.GetCoordinatesActual.yValue 'loops until the start square is reached
    pathStack.Push(lastSquare) 'the start square is also added to the stack

    While pathStack.IsEmpty = False 'until the stack is empty
        path.Add(pathStack.Pop()) 'each element in the stack is popped off (removed from the top of the stack) and added to the path list
    End While

    Return path 'the path (list) is returned
End Function

End Class
```

6.2.2 Maze Class

```
Public Class Maze
```

```
    Dim connection As New System.Data.Odbc.OdbcConnection("DRIVER={MySQL ODBC 5.3 ANSI
Driver};SERVER=localhost;PORT=3306;DATABASE=pathfindingdatabase;USER=root;PASSWORD=roo
t;OPTION=3;")

    Private id As Integer
    Private name As String
    Private grid(,) As Square
    Private adjacencyMatrix(,) As Boolean 'default boolean is false
    Private collectionOfNodeSquares() As Square
    Private algorithmVariable As Algorithm

    Private startSqu As StartSquare
    Private endSqu As EndSquare

    Private numberOfSquaresTraversed As Integer 'from startSqu to endSqu (estimation) (used as a
measurement of the distance between the two points in squares
    Private endFound As Boolean
    Private pathToEnd As New List(Of Square) 'stores shortest path, its length-1 is the number squares
needed to be traversed from the start to end square (including the start and end square)

    Private lengthOfTimeToComplete As Integer 'used to store the time it took for an algorithm to run
    Private startTime As DateTime 'used to store the starting time of when an algorithm is run

    Public Sub New(ByVal inName As String, ByVal width As Integer, ByVal height As Integer, ByVal
inStartSqu As StartSquare, ByVal inEndSqu As EndSquare) 'creates a new Maze (object)

        connection.Open()

        Dim idFromTable As Odbc.OdbcDataReader
        Dim getID As New Odbc.OdbcCommand("SELECT MAX(mazeID) FROM mazeinfo", connection)

        idFromTable = getID.ExecuteReader()

        Try
            If idFromTable.Read = True Then
                id = idFromTable.GetInt32(0) + 1
            End If
        Catch empty As System.InvalidCastException 'if table is empty no data can be retrieved
            id = 1 'so id is made to be 1
        End Try


        name = inName
        ReDim grid(width * 2, height * 2) 'transformations so that walls can be stored in the grid variable
        inStartSqu.ChangeXCord(((inStartSqu.GetCoordinates.xValue) * 2) - 1)
        inStartSqu.ChangeYCord(((inStartSqu.GetCoordinates.yValue) * 2) - 1)
        startSqu = inStartSqu

        inEndSqu.ChangeXCord(((inEndSqu.GetCoordinates.xValue) * 2) - 1)
```

```
        inEndSqu.ChangeYCord(((inEndSqu.GetCoordinates.yValue) * 2) - 1)
        endSqu = inEndSqu

        ReDim collectionOfNodeSquares(((grid.GetLength(0) - 1) / 2) * ((grid.GetLength(1) - 1) / 2)) 'has a length of e.g. 16, but at index 0 it is empty.

        algorithmVariable = New Algorithm(Me, startSqu)

        GenerateMaze()

        connection.Close()

    End Sub

    Public Sub New(ByVal inputMazeID As Integer) 'creates a new Maze (object) by importing a maze from the table mazeinfo using the inputMazeID

        Dim tempWidth As Integer
        Dim tempHeight As Integer

        Dim tempXValue As Integer
        Dim tempYValue As Integer

        Dim wallStatusString As String = ""

        '-----

        id = inputMazeID
        connection.Open()

        '-----

        Dim processInfoFromTable As Odbc.OdbcDataReader
        Dim getProcessInfo As New Odbc.OdbcCommand("SELECT mazeName, mazeWidth, mazeHeight, startSquareX, startSquareY, endSquareX, endSquareY, wallStatus FROM mazeinfo WHERE mazeID = '" & id & "'", connection) 'selects the information needed to load the maze with inputted ID from the SQL table 'mazeinfo' from the database 'pathfindingdatabase'

        processInfoFromTable = getProcessInfo.ExecuteReader()


        While processInfoFromTable.Read = True 'stores the data retrieved from the table in variables
            name = (processInfoFromTable("mazeName"))

            tempWidth = (processInfoFromTable("mazeWidth"))
            tempHeight = (processInfoFromTable("mazeHeight"))
            ReDim grid(tempWidth * 2, tempHeight * 2)

            tempXValue = (processInfoFromTable("startSquareX"))
```

```
            tempYValue = (processInfoFromTable("startSquareY"))
            startSqu = New StartSquare((((tempXValue) * 2) - 1), (((tempYValue) * 2) - 1))

            tempXValue = (processInfoFromTable("endSquareX"))
            tempYValue = (processInfoFromTable("endSquareY"))
            endSqu = New EndSquare((((tempXValue) * 2) - 1), (((tempYValue) * 2) - 1))

            ReDim collectionOfNodeSquares(((grid.GetLength(0) - 1) / 2) * ((grid.GetLength(1) - 1) / 2)) 'has
a length of e.g. 16, but at index 0 it is empty.

            wallStatusString = (processInfoFromTable("wallStatus"))

        End While

        '-----

        AddingValues(0, 0)

        'this double FOR loop uses the wallStatusString to assign whether a wall square is a wall or empty
(so can be crossed)
        For y = 1 To (grid.GetLength(1))
            For x = 1 To (grid.GetLength(0))
                If Mid(wallStatusString, (x) + ((y - 1) * (grid.GetLength(0))), 1) = "1" Then
                    grid(x - 1, y - 1).ChangeWallStatus(True)
                Else
                    grid(x - 1, y - 1).ChangeWallStatus(False)
                End If
            Next
        Next

        GenerateAdjacencyMatrix()

        algorithmVariable = New Algorithm(Me, startSqu)

        connection.Close()
    End Sub

    Public Sub GenerateAdjacencyMatrix() 'creates an adjacency matrix which once created can easily &
quickly be used to determine whether two squares neighbours/adjacent to one another
        ReDim adjacencyMatrix(((grid.GetLength(0) - 1) / 2) * ((grid.GetLength(1) - 1) / 2),
((grid.GetLength(0) - 1) / 2) * ((grid.GetLength(1) - 1) / 2))
        For y = 1 To (grid.GetLength(1) / 2) 'actual coordinates
            For x = 1 To (grid.GetLength(0) / 2)
                If grid((x * 2) - 1, (y * 2) - 1).GetNodeStatus = True Then

                    'looks at whether it is possible to go in the direction:

                    If grid((x * 2) - 1, (y * 2) - 2).GetWallStatus = False Then 'up
```

```
                adjacencyMatrix(grid((x * 2) - 1, (y * 2) - 1).GetNodeNumber, grid((x * 2) - 1, (y * 2) -
3).GetNodeNumber) = True
            End If

            If grid((x * 2), (y * 2) - 1).GetWallStatus = False Then 'right
                adjacencyMatrix(grid((x * 2) - 1, (y * 2) - 1).GetNodeNumber, grid((x * 2) + 1, (y * 2) -
1).GetNodeNumber) = True
            End If

            If grid((x * 2) - 1, (y * 2)).GetWallStatus = False Then 'down
                adjacencyMatrix(grid((x * 2) - 1, (y * 2) - 1).GetNodeNumber, grid((x * 2) - 1, (y * 2) +
1).GetNodeNumber) = True
            End If

            If grid((x * 2) - 2, (y * 2) - 1).GetWallStatus = False Then 'left
                adjacencyMatrix(grid((x * 2) - 1, (y * 2) - 1).GetNodeNumber, grid((x * 2) - 3, (y * 2) -
1).GetNodeNumber) = True
            End If

        End If
      Next
    Next

  End Sub

  Public Function GetID() As String 'returns the id of the maze
    Return id
  End Function

  Public Function GetName() As String 'returns the name of the maze
    Return name
  End Function

  Public Function GetStartSquare() As StartSquare 'returns the start square of the maze
    Return startSqu
  End Function

  Public Function GetEndSquare() As EndSquare 'returns the end square of the maze
    Return endSqu
  End Function

  Sub GenerateWalls(ByVal tempSquare As Square) 'generates walls for the maze, can only be used to
create mazes with one solution
    Randomize() 'needed for random number generation
    Dim randomPosition As Integer

    Dim tempVisitArray(3) As Boolean
```

```
        For i = 0 To tempVisitArray.Length - 1 'makes each each member in the array tempVisitArray store the boolean False
            tempVisitArray(i) = False
        Next

        tempSquare.Traverse() 'traverses tempSquare


        For i = 1 To 4 'to reach all possible traversable squares

            Do 'randomly chooses a position from 1, 2, 3, 4 where there will not be a wall
                randomPosition = (Int(Rnd() * (4))) + 1
            Loop Until tempVisitArray(randomPosition - 1) = False 'loops until the program randomly selects a direction that hasn't been visited yet (the -1 is there as array's start with an index of 0).

            Select Case randomPosition 'based on which number was randomly generated the wall in that direction becomes crossable
                Case 1 'up
                    If tempSquare.GetCoordinates.yValue - 2 > 0 Then
                        If grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue - 2).GetTraversedStatus = False Then 'new random number if true, backtracking, counter to see if any squares left
                            grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue - 1).ChangeWallStatus(False)
                            grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue - 2).Traverse()
                            GenerateWalls(grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue - 2))
                        End If
                    End If
                Case 2 'right
                    If tempSquare.GetCoordinates.xValue + 2 < grid.GetLength(0) - 1 Then
                        If grid(tempSquare.GetCoordinates.xValue + 2, tempSquare.GetCoordinates.yValue).GetTraversedStatus = False Then
                            grid(tempSquare.GetCoordinates.xValue + 1, tempSquare.GetCoordinates.yValue).ChangeWallStatus(False)
                            grid(tempSquare.GetCoordinates.xValue + 2, tempSquare.GetCoordinates.yValue).Traverse()
                            GenerateWalls(grid(tempSquare.GetCoordinates.xValue + 2, tempSquare.GetCoordinates.yValue))
                        End If
                    End If
                Case 3 'down
                    If tempSquare.GetCoordinates.yValue + 2 < grid.GetLength(1) - 1 Then
                        If grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue + 2).GetTraversedStatus = False Then
                            grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue + 1).ChangeWallStatus(False)
```

```
                    grid(tempSquare.GetCoordinates.xValue, tempSquare.GetCoordinates.yValue +
2).Traverse()
                    GenerateWalls(grid(tempSquare.GetCoordinates.xValue,
tempSquare.GetCoordinates.yValue + 2))
                End If
            End If
        Case 4 'left
            If tempSquare.GetCoordinates.xValue - 2 > 0 Then
                If grid(tempSquare.GetCoordinates.xValue - 2,
tempSquare.GetCoordinates.yValue).GetTraversedStatus = False Then
                    grid(tempSquare.GetCoordinates.xValue - 1,
tempSquare.GetCoordinates.yValue).ChangeWallStatus(False)
                    grid(tempSquare.GetCoordinates.xValue - 2,
tempSquare.GetCoordinates.yValue).Traverse()
                    GenerateWalls(grid(tempSquare.GetCoordinates.xValue - 2,
tempSquare.GetCoordinates.yValue))
                End If
            End If
    End Select

    tempVisitArray(randomPosition - 1) = True 'this means that the square in that array with that
position has been traversed (when generating the maze)

    Next

End Sub

Sub AddingValues(ByVal tempX As Integer, ByVal tempY As Integer) 'Adds/saves the start and end
square to the correct coordinates within the maze/grid. The squares that should be walls in the grid are
created using the Wall class. It also makes the border Walls in the grid, have a border status of TRUE so
that the program won't make the wall boolean false for them later in the program
    Dim tempWall As New Wall(tempX, tempY)

    If tempX = startSqu.GetCoordinates.xValue And tempY = startSqu.GetCoordinates.yValue Then
'compares the x and y values of the temp square and the start square's coordinates to see if they are the
same, if so then the square with those coordinates becomes the start square
        grid(tempX, tempY) = startSqu 'adds the start square to the grid (2D array) of squares

        collectionOfNodeSquares(startSqu.GetNodeNumber) = startSqu 'makes the inputted index
(which is the associated node number of the start square) of the array store the start square itself (so
that later the square can be accessed through the associated node number)

    ElseIf tempX = endSqu.GetCoordinates.xValue And tempY = endSqu.GetCoordinates.yValue Then
'compares the x and y values of the temp square and the end square's coordinates to see if they are the
same, if so then the square with those coordinates becomes the end square
        grid(tempX, tempY) = endSqu 'adds the  end square to the grid (2D array) of squares
```

```
        collectionOfNodeSquares(endSqu.GetNodeNumber) = endSqu 'makes the inputted index (which is the associated node number of the end square) of the array store the end square itself (so that later the square can be accessed through the associated node number)

    ElseIf tempY Mod 2 = 0 Or tempX Mod 2 = 0 Then 'makes squares that should be walls as a default, Walls
        If tempY = 0 Or tempX = 0 Or tempY = (grid.GetLength(1) - 1) Or tempX = (grid.GetLength(0) - 1) Then 'sees whether the Wall needs to be a border or not, (when doing .GetLength on a 2D array 0 for x length and 1 for y length)
            tempWall.ChangeBorderStatus(True) 'makes the border status of the Wall TRUE
        End If
        grid(tempX, tempY) = tempWall 'adds the tempWall to the grid (2D array) of squares (& wall as Wall inherits from the Square class)
    Else
        Dim tempSquare As New NodeSquare(tempX, tempY) 'inside if statement as if outside then nodeNumber increases in the Square Class despite the square being a temp square

        grid(tempX, tempY) = tempSquare 'adds the tempSquare to the grid (2D array) of squares

        collectionOfNodeSquares(tempSquare.GetNodeNumber) = tempSquare 'makes the inputted index (which is the associated node number of the square) of the array store the square itself (so that later the square can be accessed through the associated node number)

    End If



    If tempX = grid.GetLength(0) - 1 Then 'custom loop, sees whether the tempX integer is equal to the width of the maze/grid (the 2D array)
        tempX = 0 'if so resets the tempX to 0 so that the next row can be generated where tempY increases by one and tempX is 0 again
        If tempY <> grid.GetLength(1) - 1 Then  'checks whether tempY is equal to the height of the maze/grid (the 2D array), if so the generation is complete
            tempY += 1 'if it isn't equal to the height then tempY increments by 1
            AddingValues(tempX, tempY) 'recursion until the generation is complete
        End If
    Else 'else tempX increases by one and recursion until the generation is complete
        tempX += 1
        AddingValues(tempX, tempY)
    End If

End Sub

Public Sub GenerateMaze() 'generates the maze so that is ready to have pathfinding algorithms run on it, error handling needed for unsuitable dimensions

    '======== Generation of Start & End Square, Walls/Borders & Adjacency Matrix ========'

    AddingValues(0, 0)
```

```
        GenerateWalls(startSqu) 'attempts to randomly generate a path from the start square to the end square

        GenerateAdjacencyMatrix()


'================================================================================================='


    End Sub

    Public Function OutputMaze() As String 'returns the string representation of the maze
        Dim mazeString As String = ""
        For i = 0 To grid.GetLength(1) - 1
            For k = 0 To grid.GetLength(0) - 1
                mazeString &= grid(k, i).OutputString()
            Next
            mazeString &= vbCrLf
        Next
        Return mazeString
    End Function

    Public Sub SaveMaze() 'saves the maze into the mazeinfo table in the pathfindingdatabase

        connection.Open()

        Dim wallStatusString As String = "" 'stored as a string of 1s and 0s

        For y = 0 To grid.GetLength(1) - 1
            For x = 0 To grid.GetLength(0) - 1

                If grid(x, y).GetWallStatus = True Then
                    wallStatusString &= "1" 'represents the wall status being TRUE
                Else
                    wallStatusString &= "0" 'represents the wall status being FALSE
                End If

            Next
        Next

        Dim sql As New Odbc.OdbcCommand("INSERT INTO mazeinfo(mazeID,mazeName,mazeWidth,mazeHeight,startSquareX,startSquareY,endSquareX,endSquareY,wallStatus) VALUES ('" & id & "', '" & name & "', '" & ((grid.GetLength(0) - 1) / 2) & "', '" & ((grid.GetLength(1) - 1) / 2) & "', '" & startSqu.GetCoordinatesActual.xValue & "', '" & startSqu.GetCoordinatesActual.yValue & "', '" & endSqu.GetCoordinatesActual.xValue & "', '" & endSqu.GetCoordinatesActual.yValue & "', '" & wallStatusString & "')", connection) 'data is stored in the SQL table mazeinfo
        sql.ExecuteNonQuery() 'executes the query above
```

```
        connection.Close()

    End Sub

    Public Sub ResetMaze() 'resets all squares in the maze/grid and all other required information that needs to be reset before creating a new maze

        Dim tempSquare As New Square(-1, -1)

        tempSquare.ResetNodeNumber()


    End Sub

    Public Sub ResetForSearch() 'resets all squares in the maze/grid and all other required information that needs to be reset

        For i = 0 To grid.GetLength(1) - 1
            For k = 0 To grid.GetLength(0) - 1
                grid(k, i).ResetSquare()
            Next
        Next

        numberOfSquaresTraversed = 0
        ChangeEndFoundStatus(False)
        algorithmVariable.ResetInformation()

    End Sub

    Public Sub RunAlgorithm(ByVal type As Algorithm.Algorithms) 'depending on the parameter input runs a certain algorithm

        ResetForSearch() 'as visit nodes were used previously when generating the maze

        If type = Algorithm.Algorithms.AStar Then
            StartTimer() 'timer is started here separately to make the time measured as accurate as possible
            pathToEnd = algorithmVariable.StartAStarSearch() 'once the algorithm is run, the path from the start square to the end square is returned and stored in the variable pathToEnd
            EndTimer() 'timer is ended here separately to make the time measured as accurate as possible

        ElseIf type = Algorithm.Algorithms.BreadthFirst Then
            StartTimer() 'timer is started here separately to make the time measured as accurate as possible
            pathToEnd = algorithmVariable.StartBreadthFirstSearch() 'once the algorithm is run, the path from the start square to the end square is returned and stored in the variable pathToEnd
            EndTimer() 'timer is ended here separately to make the time measured as accurate as possible

        ElseIf type = Algorithm.Algorithms.Dijkstras Then
            StartTimer() 'timer is started here separately to make the time measured as accurate as possible
```

```
        pathToEnd = algorithmVariable.StartDijkstrasAlgorithm() 'once the algorithm is run, the path from the start square to the end square is returned and stored in the variable pathToEnd
        EndTimer() 'timer is ended here separately to make the time measured as accurate as possible

    ElseIf type = Algorithm.Algorithms.MyAlgorithm Then
        StartTimer() 'timer is started here separately to make the time measured as accurate as possible
        pathToEnd = algorithmVariable.StartMyAlgorithm() 'once the algorithm is run, the path from the start square to the end square is returned and stored in the variable pathToEnd
        EndTimer() 'timer is ended here separately to make the time measured as accurate as possible

    End If



    SaveProcessInfo(type)

End Sub

Public Sub SaveProcessInfo(ByVal algorithmType As Algorithm.Algorithms) 'saves the current information stored in variables about a process in the SQL table algorithmsrun

    Dim algorithmID As Integer

    Dim processID As Integer
    Dim processIDFromTable As Odbc.OdbcDataReader
    Dim getprocessID As New Odbc.OdbcCommand("SELECT MAX(processID) FROM algorithmsrun", connection) 'gets the maximum processID from the algorithmsrun table, this is done so that the new process can have the next possible unique processID

    Dim pathString As String = ""

    connection.Open()

    If algorithmType = Algorithm.Algorithms.AStar Then
        algorithmID = 1 'the associated algorithmID of AStar
    ElseIf algorithmType = Algorithm.Algorithms.BreadthFirst Then
        algorithmID = 2 'the associated algorithmID of BreadthFirst
    ElseIf algorithmType = Algorithm.Algorithms.Dijkstras Then
        algorithmID = 3 'the associated algorithmID of Dijkstras
    ElseIf algorithmType = Algorithm.Algorithms.MyAlgorithm Then
        algorithmID = 4 'the associated algorithmID of MyAlgorithm
    End If


    processIDFromTable = getprocessID.ExecuteReader() 'gets the current maximum process ID

    Try
        If processIDFromTable.Read = True Then
            processID = processIDFromTable.GetInt32(0) + 1
```

```
            End If
        Catch empty As System.InvalidCastException 'if table is empty no data can be retrieved
            processID = 1 'so processID is made to 1
        End Try

        For i = 0 To pathToEnd.Count - 1 'gets the node number of each square in the list "pathToEnd" and puts them into a string where they are each separated by commas (",")
            If i <> pathToEnd.Count - 1 Then
                pathString = pathString & pathToEnd.Item(i).GetNodeNumber & ","
            Else
                pathString = pathString & pathToEnd.Item(i).GetNodeNumber
            End If
        Next

        Dim sql As New Odbc.OdbcCommand("INSERT INTO algorithmsrun(processID,mazeID,algorithmID,pathToEnd,timeTakenToRun,numberOfSquaresTraversed) VALUES ('" & processID & "', '" & id & "', '" & algorithmID & "', '" & pathString & "', '" & lengthOfTimeToComplete & "', '" & numberOfSquaresTraversed & "')", connection) 'inputs the values stored in the variables into the SQL table algorithmsrun
        sql.ExecuteNonQuery() 'executes the SQL statement above ^

        connection.Close()

    End Sub

    Public Function LoadProcess() As List(Of String()) 'loads information about processes (when an algorithm is run on a maze) from the pathfindingdatabase

        Dim results As New List(Of String()) 'a list as the number elements/members which will be stored in it is unknown
        Dim record(4) As String
        Dim emptyRecord(4) As String
        Dim pathString As String
        Dim nodeArray() As String

        connection.Open()

        Dim processInfoFromTable As Odbc.OdbcDataReader
        Dim getProcessInfo As New Odbc.OdbcCommand("SELECT processID, algorithmName, pathToEnd, timeTakenToRun, numberOfSquaresTraversed FROM algorithmsrun, typesofalgorithms WHERE algorithmsrun.mazeID = '" & id & "' AND typesofalgorithms.algorithmID = algorithmsrun.algorithmID", connection) 'allow user to import info based on processID

        processInfoFromTable = getProcessInfo.ExecuteReader()


        While processInfoFromTable.Read = True
            record = emptyRecord.Clone() 'the array "record" is updated to be a clone of "emptyRecord" (so that there is a new object reference associated with the array "record")
```

```
        record(0) = (processInfoFromTable("processID")) 'the retrieved processID of the current record/selection from the SQL table is put into the array record at index 0
        record(1) = (processInfoFromTable("algorithmName")) 'the retrieved algorithmName of the current record/selection from the SQL table is put into the array record at index 1

        nodeArray = Split((processInfoFromTable("pathToEnd")), ",") 'the retrieved pathToEnd (which is a list of node numbers that represent the path from the start square to the end square) of the current record/selection from the SQL table is split at comma (",") and put into the array nodeArray
        For Each nodeNumber In nodeArray
            pathString = pathString & "(" & GetSquareUsingNodeNumber(nodeNumber).GetCoordinatesActual.xValue & "," & GetSquareUsingNodeNumber(nodeNumber).GetCoordinatesActual.yValue & ")" 'using the node number of a square, the square's actual coordinates are retrieved, and add to a string of coordinates
        Next
        record(2) = pathString 'the string of coordinates (that show the path from the start square to the end square) is stored in the array "record" at the index 2
        pathString = "" 'pathString is made empty again for the record/selection from the SQL database (as multiple records/results may have been selected using the one query)
        record(3) = (processInfoFromTable("timeTakenToRun")) 'the retrieved timeTakenToRun (in microseconds) is put into the array 'record' at the index 3
        record(4) = (processInfoFromTable("numberOfSquaresTraversed")) 'the retrieved numberOfSquaresTraversed is put into the array 'record' at the index 4

        results.Add(record) 'the current record array is add to the list "results"

    End While

    connection.Close()

    Return results

  End Function

  Public Function GetIfAdjacentSquares(ByVal currentSquare As Square, ByVal targetSquare As Square) As Boolean 'returns whether the two squares are adjacent/neighbours or not
    If adjacencyMatrix(currentSquare.GetNodeNumber, targetSquare.GetNodeNumber) = True Then 'or other way round (but checking both is time consuming)
        Return True
    Else
        Return False
    End If
  End Function

  Public Function GetNeighbours(ByVal currentSquare As Square) As List(Of Square) 'returns a list of the neighbours/adjacent squares of the square inputted into the parameter
    Dim neighbours As New List(Of Square)
    For i = 1 To adjacencyMatrix.GetLength(0) - 1
        If adjacencyMatrix(currentSquare.GetNodeNumber, i) = True Then
            neighbours.Add(GetSquareUsingNodeNumber(i))
```

```
            End If
        Next
        Return neighbours
    End Function

    Public Function GetSquareUsingNodeNumber(ByVal nodeNumber As Integer) As Square 'returns the square with the node number inputted into the parameter
        Return collectionOfNodeSquares(nodeNumber)
    End Function

    Public Function IsEndFound() As Boolean 'returns the boolean of endFound, if it is TRUE then the end square has been found, if it is FALSE then the end square hasn't been found
        Return endFound
    End Function

    Public Sub ChangeEndFoundStatus(ByVal status As Boolean) 'changes the endFound status to the boolean parameter input
        endFound = status
    End Sub

    Public Sub IncreaseNumberOfSquaresTraversed() 'increments the counter which stores the numberOfSquaresTraversed
        numberOfSquaresTraversed += 1
    End Sub

    Public Function GetPath() As List(Of Square) 'returns the list of squares which must be traversed to get from the start square to the end square
        Return pathToEnd
    End Function

    Public Function GetCurrentTime() As Date 'return the current time
        Return Date.Now
    End Function

    Public Sub StartTimer() 'used to assign the current time as the startTime (of when the algorithm started running)
        startTime = GetCurrentTime()
    End Sub

    Public Function CalculateMicroseconds(ByVal endTime As Date) As Integer 'returns the time in microseconds (but using integer data type)
        Return (endTime.Ticks - startTime.Ticks) / 10 ' calculates the time it took in microseconds for an algorithm to be run on a maze
    End Function

    Public Sub EndTimer() 'used to get the lengthOfTimeToComplete (the time it took for an algorithm to be run on the maze) through using the startTime and getting the current time (endTime)
        Dim endTime As New Date
        Dim newStart As New Date 'used to reset startTime
```

```
        Try
            endTime = GetCurrentTime() 'to stop the 'timer' here so that the following processes are not
timed as well
            lengthOfTimeToComplete = CalculateMicroseconds(endTime) 'calculates the
lengthOfTimeToComplete in microseconds
            startTime = newStart 'resets the startTime to default
        Catch tooLong As OverflowException
            lengthOfTimeToComplete = -1 'stored if it takes too long to run an algorithm on a maze
        End Try
    End Sub

    Public Function GetTimeLength() As Integer 'returns the time taken to run an algorithm in
microseconds (using integer data type)
        Return lengthOfTimeToComplete
    End Function

    Public Function GetCollectionOfNodes() As Square() 'returns the array of squares where the index of
each square is its node number
        Return collectionOfNodeSquares
    End Function

End Class
```

6.2.3 Queue Class

```
Public Class Queue ' a circular priority queue

    Private maxSize As Integer 'the maxSize that the queue can be
    Private size As Integer = 0 'keeps track of the current size of the queue
    Private squareQueue() As Square 'an array to store the contents of the queue, as array starts at 0
    Private front As Integer = 0 'keeps track of the front of the queue
    Private rear As Integer = -1 'keeps track of the rear of the queue

    Enum sortByCost 'the types of costs that the BubbleSortQueue subroutine can order the queue by
        fCost
        hCost
        gCost
    End Enum

    Public Structure listHCost 'structure to make one variable (that can be put in e.g. a list) store two
pieces information
        Dim contents As Square
        Dim index As Integer
    End Structure

    Public Sub New(ByVal inMaxSize As Integer) 'creates a new queue
        maxSize = inMaxSize
        ReDim squareQueue(maxSize)
```

```
    End Sub
    Public Sub Enqueue(ByVal inSquare As Square) 'adds the square passed into the parameter to the
back of the queue
        If IsFull() = False Then
            rear += 1
            size += 1
            LoopRound()
            squareQueue(rear) = inSquare
        End If

    End Sub

    Public Function Dequeue() As Square 'returns the member/square at the front of the queue and
removes it from the queue
        Dim originalFront As Integer = front
        If IsEmpty() = False Then
            front += 1
            size -= 1
            LoopRound()
            Return squareQueue(originalFront) 'return removed square
        End If

    End Function

    Public Function Peek() As Square 'returns the member/square at the front of the queue
        If IsEmpty() = False Then
            Return squareQueue(front)
        End If
    End Function

    Public Function IsFull() As Boolean 'returns whether the queue is full or not (returns True if it is and
False if it isn't)
        If size = maxSize Then
            Return True
        Else
            Return False
        End If
    End Function

    Public Function IsEmpty() As Boolean 'returns whether the queue is empty or not (returns True if it is
and False if it isn't)
        If size = 0 Then
            Return True
        Else
            Return False
        End If
    End Function

    Public Function GetMaxSize() As Integer 'returns the maxSize of the queue
```

```
        Return maxSize
    End Function

    Public Sub LoopRound() 'loops queue back to start if its front/rear is greater than the maxSize but not full, it makes the queue a circular queue
        If rear > maxSize Then
            rear = (rear Mod maxSize) - 1
        End If
        If front > maxSize Then
            front = (front Mod maxSize) - 1
        End If
    End Sub

    Public Function CheckNumInList(ByVal list As List(Of Integer), ByVal number As Integer) As Integer 'checks whether the inputted parameter value 'number' is in the inputted parameter value 'list', if so returns True
        For i = 0 To list.Count - 1
            If list(i) = number Then
                Return True
            End If
        Next
        Return False
    End Function

    Public Sub BubbleSortQueue(ByVal costType As sortByCost) 'runs bubble sort on queue (to make it a priority queue) and orders it based on the costType passed into the parameter
        Randomize() 'allows the generation of random number

        Dim orderCostList As New List(Of Square)

        While IsEmpty() = False 'moves the contents of queue into a list
            orderCostList.Add(Dequeue)
        End While

        Dim orderCost(orderCostList.Count - 1) As Square

        For i = 0 To orderCost.Length - 1 'moves the contents of list into an array, the reason it was made a list first is because a list can change in size whereas an array cannot, by putting the squares into a list I can calculate the length which the array must be
            orderCost(i) = orderCostList.Item(i) 'they are put into an array as it is easier to change the order of items in an array
        Next

        Dim temp As Square
        Dim tempList As New List(Of listHCost)
        Dim tempListHCost As listHCost

        Dim fCostMin As Integer = -1
        Dim hCostMin As Integer = -1
```

```
        Dim randomNumber As Integer

        Dim change As Boolean
        Dim length As Integer = orderCost.Length - 1

        If orderCost.Length > 1 Then 'as no need to run the following code if there is only one element

            Do 'this DO loop performs a bubble sort on the contents of the array based in a certain costType
                change = False 'used to prevent the bubble sort from carrying on infinitely
                For i = 1 To length
                    If orderCost(i - 1).GetCost(costType) > orderCost(i).GetCost(costType) Then
                        temp = orderCost(i - 1)
                        orderCost(i - 1) = orderCost(i)
                        orderCost(i) = temp
                        change = True 'used to prevent the bubble sort from carrying on infinitely
                    End If
                Next
                length -= 1 'used to minimise the times the number FOR loops (as after every iteration of the DO loop the largest costType will be in the correct position)
            Loop Until change = False Or length = 0

            If costType = sortByCost.hCost Then

                For i = 0 To orderCost.Length - 1
                    If orderCost(i).GetFCost <> -1 And fCostMin = -1 Then 'only runs when fCostMin = -1 as fCost is in order from lowest to highest so anything after -1 (the default or the fCost of invalid squares) will be the lowest fCost
                        fCostMin = orderCost(i).GetFCost 'sets the F Cost of the current square (a member of orderCost) to be the minimum fCost
                        hCostMin = orderCost(i).GetHCost 'sets the H Cost of the current square (a member of orderCost) to be the minimum hCost
                    ElseIf orderCost(i).GetFCost = fCostMin And orderCost(i).GetHCost < hCostMin Then 'if a member of orderCost has the same fCost but has lower hCost as the square which set the previous minimum fCost & hCost, the following code is run
                        hCostMin = orderCost(i).GetHCost
                    End If
                Next

                For i = 0 To orderCost.Length - 1 'members of orderCost with the same fCost (lowest) and hCost (lowest) are put into a list
                    If orderCost(i).GetFCost = fCostMin And orderCost(i).GetHCost = hCostMin Then
                        tempListHCost.contents = orderCost(i)
                        tempListHCost.index = i
                        tempList.Add(tempListHCost)
                    End If
                Next

                randomNumber = (Int(Rnd() * (tempList.Count))) 'chooses random index, if there are e.g. two numbers same F and H cost then the index 0 or 1 can be chosen randomly
```

```
            temp = orderCost(randomNumber) 'as the contents of orderCost(i) is going to be replaced it is stored in a temp variable
            orderCost(randomNumber) = tempList.Item(randomNumber).contents 'the randomly selected member is stored in orderCost at the position 'i'
            orderCost(tempList.Item(randomNumber).index) = temp 'the previous contents of orderCost(i) is stored in the originally location of the randomly selected member




        End If

        If orderCost(0).GetFCost = orderCost(1).GetFCost And costType = sortByCost.fCost Then 'if the fCost of first two members of orderCost are the same, and the subroutine just ordered the queue based on the fCost the following code is run

            For i = 0 To (orderCost.Length - 1) 'puts the all the squares back into the queue
                Enqueue(orderCost(i))
            Next

            BubbleSortQueue(sortByCost.hCost) 'runs the current subroutine again but passes hCost as the sortByCost

        Else

            For i = 0 To (orderCost.Length - 1) 'puts the all the squares back into the queue
                Enqueue(orderCost(i))
            Next

        End If
    Else
        For i = 0 To (orderCost.Length - 1) 'puts the all the squares back into the queue
            Enqueue(orderCost(i))
        Next
    End If

End Sub

End Class
```

6.2.4 StackOfSquares Class

```
Public Class StackOfSquares

    Private topCounter As Integer = -1
    Private maxSize As Integer
    Private squareStack() As Square 'as array starts at 0
```

```
    Public Sub New(ByVal inMaxSize As Integer) 'creates a new stack
        maxSize = inMaxSize
        ReDim squareStack(maxSize - 1)
    End Sub

    Public Sub Push(ByVal inSquare As Square) 'adds/pushes inputted square on top of the stack as long
the stack isn't full
        If IsFull() = False Then
            topCounter += 1
            squareStack(topCounter) = inSquare
        End If
    End Sub

    Public Function Pop() As Square 'removes and returns the square from the top of the stack as long as
the stack isn't empty
        If IsEmpty() = False Then
            topCounter -= 1
            Return squareStack(topCounter + 1)
        End If
        'doesn't return something in all cases but it is still dealt with properly
    End Function

    Public Function Peek() As Square 'looks/returns (but doesn't remove) the square at the top of the
stack as long as the stack isn't empty
        If IsEmpty() = False Then
            Return squareStack(topCounter)
        End If
        'doesn't return something in all cases but it is still dealt with properly
    End Function

    Public Function IsFull() As Boolean 'checks if the stack has exceeded its maximum size, returns true if
it has and false if not
        If topCounter = maxSize Then
            Return True
        Else
            Return False
        End If
    End Function

    Public Function IsEmpty() As Boolean 'checks if the stack is empty, returns true if it has and false if not
        If topCounter = -1 Then
            Return True
        Else
            Return False
        End If
    End Function
End Class
```

6.2.5 Square Class

```
Public Class Square
    Protected gCost As Integer
    Protected hCost As Integer
    Protected fCost As Integer

    Protected previousSquare As Square
    Protected discovered As Boolean = False
    Protected traversed As Boolean = False
    Protected wall As Boolean = False
    Protected node As Boolean = False 'all traversable squares (NodeSquare) have node = True, TempSquares (Square) / Wall (which a type of square) have node = False . The reason there is a separate variable called 'node', is so that the program can store whether a square was originally a traversable Square or a Wall. As you cannot just use the wall boolean to determine this as once a Wall (Square) becomes crossable (so there is no longer a wall between two squares), the wall boolean which was originally True becomes False. TempSquares cannot have a nodeNumer as they are not nodes.
    Protected nodeNumber As Integer = 0
    Protected Shared nextNodeNumber As Integer = 1

    Protected startSquare As Boolean = False 'boolean to store whether the square is the start square (if so then True)
    Protected endSquare As Boolean = False 'boolean to store whether the end is the start square (if so then True)

    Public Structure coord 'a structure that stores two integers: xValue and yValue
        Dim xValue As Integer
        Dim yValue As Integer
    End Structure

    'the difference between coordinate and coordinateActual is that coordinate is the coordinates of square in the 2D array (grid) when the wall squares are included and coordinateActual is the actual coordinates of the square when the walls (between each traversable square) are excluded
    Protected coordinate As New coord
    Protected coordinateActual As New coord

    Public Sub New(ByVal xValue As Integer, ByVal yValue As Integer) 'creates a new square using inputs
        coordinate.xValue = xValue
        coordinate.yValue = yValue
        coordinateActual.xValue = (xValue + 1) / 2
        coordinateActual.yValue = (yValue + 1) / 2
    End Sub

    Public Function GetCoordinates() As coord 'returns coordinates for square in coord structure form
        Return coordinate
    End Function
```

```
    Public Sub ChangeXCord(ByVal xValue As Integer) 'changes x cord of a square to input, then applies required transformations
        coordinate.xValue = xValue
        coordinateActual.xValue = (xValue + 1) / 2
    End Sub

    Public Sub ChangeYCord(ByVal yValue As Integer) 'changes y cord of a square to input, then applies required transformations
        coordinate.yValue = yValue
        coordinateActual.yValue = (yValue + 1) / 2
    End Sub

    Public Function GetCoordinatesActual() As coord 'returns the actual coordinates for square in coord structure form
        Return coordinateActual
    End Function

    Public Sub UpdateSquare() 'updates fCost (using new gCost and hCost)
        fCost = gCost + hCost
    End Sub

    Public Sub ResetSquare() 'resets square so that it isn't discovered or traversed
        Dim tempSquare As Square
        discovered = False
        traversed = False
        ChangeGCost(0)
        ChangeHCost(0)
        ChangePreviousSquare(tempSquare) 'resets previous square to empty
    End Sub

    Public Function GetDiscoveredStatus() As Boolean 'returns the discovered status/boolean of square
        Return discovered
    End Function

    Public Sub Discover() 'discovers the square, represented by changing the discovered boolean of the square to true
        discovered = True
    End Sub

    Public Function GetTraversedStatus() As Boolean 'returns the traveresed status/boolean of square
        Return traversed
    End Function

    Public Sub Traverse() 'traverses the square, represented by changing the traversed boolean of the square to true
        traversed = True
    End Sub

    Public Function GetWallStatus() As Boolean 'return whether the square is a wall or not
```

```
        Return wall
    End Function

    Public Overridable Sub ChangeWallStatus(ByVal inWall As Boolean) 'changes the wall status of the
square (if overridden)

    End Sub

    Public Overridable Function OutputString() As String 'returns associated string of square,(space
outputted if not overridable)
        Return " "
    End Function

    Public Function GetGCost() As Integer 'return the gCost of associated square
        Return gCost
    End Function

    Public Function GetFCost() As Integer 'return the fCost of associated square
        Return fCost
    End Function

    Public Function GetHCost() As Integer 'return the hCost of associated square
        Return hCost
    End Function

    Public Sub ChangeGCost(ByVal inGCost As Integer) 'changes gCost to inputted value
        gCost = inGCost
        UpdateSquare()
    End Sub

    Public Sub ChangeFCost(ByVal inFCost As Integer) 'changes fCost to inputted value
        fCost = inFCost
        UpdateSquare()
    End Sub

    Public Sub ChangeHCost(ByVal inHCost As Integer) 'changes hCost to inputted value
        hCost = inHCost
        UpdateSquare()
    End Sub

    Public Sub ChangeHCost(ByVal endSquare As EndSquare) 'calculates the hCost based on input
        hCost = ((endSquare.GetCoordinatesActual.xValue - Me.GetCoordinatesActual.xValue) *
(endSquare.GetCoordinatesActual.xValue - Me.GetCoordinatesActual.xValue)) +
((endSquare.GetCoordinatesActual.yValue - Me.GetCoordinatesActual.yValue) *
(endSquare.GetCoordinatesActual.yValue - Me.GetCoordinatesActual.yValue))
        UpdateSquare()
    End Sub
```

```
    Public Function GetCost(ByVal costType As Queue.sortByCost) As Integer 'returns the inputted cost
type's associated value
        If costType = Queue.sortByCost.fCost Then
            Return GetFCost()
        ElseIf costType = Queue.sortByCost.hCost Then
            Return GetHCost()
        ElseIf costType = Queue.sortByCost.gCost Then
            Return GetGCost()
        Else
            Return -1
        End If
    End Function

    Public Function GetNodeStatus() As Boolean 'gets node status of associated square
        Return node
    End Function

    Public Function IsStartSquare() As Boolean 'returns whether square is the start square
        Return startSquare
    End Function

    Public Function IsEndSquare() As Boolean 'returns whether square is the end square
        Return endSquare
    End Function

    Public Function GetNodeNumber() As Integer 'returns the associated nodeNumber of the square
        If Me.GetNodeStatus = True Then
            Return nodeNumber
        Else
            Return 0 'returns zero if it isn't a node
        End If
    End Function

    Public Sub ResetNodeNumber() 'resets the Shared variable "nextNodeNumber" back to 1
        nextNodeNumber = 1
    End Sub

    Public Sub ChangePreviousSquare(ByVal inPreviousSquare As Square) 'changes the squares
parent/previous square to the inputted square
        previousSquare = inPreviousSquare
    End Sub

    Public Function GetPreviousSquare() As Square 'returns the previous square of this square
        Return previousSquare
    End Function

End Class
```

6.2.6 StartSquare Class

```
Public Class StartSquare
    Inherits Square 'inherits from Square class

    Public Sub New(ByVal xValue As Integer, ByVal yValue As Integer) 'creates new StartSquare using parameter inputs
        MyBase.New(xValue, yValue) 'runs the 'New' subroutine in the Square class that creates a new square
        startSquare = True
        node = True
        nodeNumber = nextNodeNumber
        nextNodeNumber += 1
    End Sub

    Public Overrides Function OutputString() As String 'overrides function in Square class and runs this subroutine instead
        Return "S"
    End Function

End Class
```

6.2.7 EndSquare Class

```
Public Class EndSquare
    Inherits Square 'inherits from Square class

    Public Sub New(ByVal xValue As Integer, ByVal yValue As Integer) 'creates new EndSquare using parameter inputs
        MyBase.New(xValue, yValue) 'runs the 'New' subroutine in the Square class that creates a new square
        endSquare = True
        node = True
        nodeNumber = nextNodeNumber
        nextNodeNumber += 1
    End Sub

    Public Overrides Function OutputString() As String 'overrides function in Square class and runs this subroutine instead
        Return "E"
    End Function

End Class
```

6.2.8 NodeSquare Class

```
Public Class NodeSquare
    Inherits Square

    Public Sub New(ByVal xValue As Integer, ByVal yValue As Integer) 'creates a new NodeSquare (a square that can be traversed)
        MyBase.New(xValue, yValue)
        node = True
        nodeNumber = nextNodeNumber
        nextNodeNumber += 1
    End Sub

End Class
```

6.2.9 Wall Class

```
Public Class Wall
    Inherits Square 'inherits from Square class

    Protected border As Boolean = False

    Public Sub New(ByVal xValue As Integer, ByVal yValue As Integer) 'creates new Wall using parameter inputs
        MyBase.New(xValue, yValue) 'runs the 'New' subroutine in the Square class that creates a new square
        wall = True
        node = False
    End Sub
    Public Function IsBorder() As Boolean 'returns whether the wall is a border. If it is, then it cannot become a space between two traversable squares that can be crossed
        Return border
    End Function
    Public Sub ChangeBorderStatus(ByVal inBorder As Boolean) 'changes the border status of wall (square) to inputted status
        border = inBorder
    End Sub
    Public Overrides Sub ChangeWallStatus(ByVal inWall As Boolean) 'changes wall status of wall (allowing it to be crossed)
        wall = inWall
    End Sub

    Public Overrides Function OutputString() As String 'overrides function in Square class and outputs a specific string depending on whether the Wall's wall boolean is true or false
        If wall = True Then
            Return "#" 'wall that cannot be crossed
        Else
```

```
            Return " " 'wall that can be crossed
        End If
    End Function

End Class
```

6.2.10 InputOutOfRange Class (Exception)

```
Public Class InputOutOfRange
    Inherits Exception

    Public Sub New()
        MyBase.New("Input isn't in the valid range") 'the associated error message with this error
    End Sub

End Class
```

6.2.11 InputLengthInvalid Class (Exception)

```
Public Class InputLengthInvalid

    Inherits Exception
    Public Sub New()
        MyBase.New("Input's length is too long") 'an associated error message with this error
    End Sub

    Public Sub New(ByVal length As Integer)
        MyBase.New("Input's length is too long, must be under " & length & " character(s) long") 'an
associated error message with this error
    End Sub

End Class
```

6.2.12 EmptyInputBox Class (Exception)

```
Public Class EmptyInputBox
    Inherits Exception
    Public Sub New()
        MyBase.New("This box cannot be left empty") 'the associated error message with this error
    End Sub
End Class
```

6.2.13 SameCoordinates Class (Exception)

```
Public Class SameCoordinates
    Inherits Exception

    Public Sub New()
        MyBase.New("The start and end square cannot have the same position/coordinates") 'the
associated error message with this error
    End Sub

End Class
```

6.2.14 OutOfMazeDimensions Class (Exception)

```
Public Class OutOfMazeDimensions
    Inherits Exception
    Public Sub New(ByVal valueType As Char, ByVal length As Integer)
        MyBase.New(valueType & ": Input isn't inside the given maze dimensions, input must be between 1
& " & length) 'the associated error message with this error
    End Sub
End Class
```

6.2.15 MazeNotFound Class (Exception)

```
Public Class MazeNotFound
    Inherits Exception
    Public Sub New()
        MyBase.New("A maze with that ID cannot be found") 'the associated error message with this error
    End Sub

End Class
```

6.2.16 RoundingOccurred Class (Exception)

```
Public Class RoundingOccurred
    Inherits Exception
    Public Sub New()
        MyBase.New("You cannot enter a decimal, the input must be an integer") 'the associated error
message with this error
    End Sub

End Class
```

6.2.17 Main_Menu_Screen Class

```
Imports System.Windows.Forms
Public Class Main_Menu_Screen

    Private Sub CreateMainMenuScreen(sender As Object, e As EventArgs) Handles MyBase.Load
'creates/loads windows/screen when the program is run

        'variables
        Dim width As Integer = 750
        Dim height As Integer = 600

        Dim optionGroupBox As New GroupBox

        Dim instructionButton As New Button
        Dim generateMazeButton As New Button
        Dim importMazeButton As New Button
        Dim exitButton As New Button


        Me.Size = New Drawing.Size(width, height) 'defines size for window

        '===== Assigning information about objects to be put onto the windows form ====

        'determines information about groupbox (that acts as a container) to contain anything added to it
        optionGroupBox.Name = "optionGroupBox"
        optionGroupBox.Location = New Drawing.Point((width / 2) - 150, (height / 2) - 200) 'determines the
location of the groupbox
        optionGroupBox.Size = New Drawing.Size(300, 350) 'determines the size of the groupbox

        'determines information about button
        instructionButton.Name = "instructionButton"
        instructionButton.Text = "Instructions"
        instructionButton.Location = New Drawing.Point((((width / 2) - 150) / 2) - 60, (((height / 2) - 150) / 2)
- 40)
        instructionButton.Size = New Drawing.Size(200, 50)
        instructionButton.FlatStyle = FlatStyle.Popup
        AddHandler instructionButton.Click, AddressOf Me.InstructionsButtonClick
        optionGroupBox.Controls.Add(instructionButton) 'adds the button to the "optionGroupBox"
groupbox

        'determines information about button
        generateMazeButton.Name = "generateMazeButton"
        generateMazeButton.Text = "Generate Maze"
        generateMazeButton.Location = New Drawing.Point((((width / 2) - 150) / 2) - 60, (((height / 2) - 150)
/ 2) + 35)
        generateMazeButton.Size = New Drawing.Size(200, 50)
```

```
        generateMazeButton.FlatStyle = FlatStyle.Popup
        AddHandler generateMazeButton.Click, AddressOf Me.GenerateMazeButtonClick
        optionGroupBox.Controls.Add(generateMazeButton) 'adds the button to the "optionGroupBox"
groupbox

        'determines information about button
        importMazeButton.Name = "importMazeButton"
        importMazeButton.Text = "Import Maze"
        importMazeButton.Location = New Drawing.Point((((width / 2) - 150) / 2) - 60, (((height / 2) - 150) /
2) + 110)
        importMazeButton.Size = New Drawing.Size(200, 50)
        importMazeButton.FlatStyle = FlatStyle.Popup
        AddHandler importMazeButton.Click, AddressOf Me.ImportMazeButtonClick
        optionGroupBox.Controls.Add(importMazeButton) 'adds the button to the "optionGroupBox"
groupbox

        'determines information about button
        exitButton.Name = "exitButton"
        exitButton.Text = "Exit"
        exitButton.Location = New Drawing.Point((((width / 2) - 150) / 2) - 60, (((height / 2) - 150) / 2) + 185)
        exitButton.Size = New Drawing.Size(200, 50)
        exitButton.FlatStyle = FlatStyle.Popup
        AddHandler exitButton.Click, AddressOf Me.ExitButtonClick
        optionGroupBox.Controls.Add(exitButton) 'adds the button to the "optionGroupBox" groupbox

        Me.Controls.Add(optionGroupBox) 'the groupbox "optionGroupBox" along with its contents is added
to the windows form

    End Sub

    Private Sub InstructionsButtonClick() 'runs when button with the text "Instructions" is clicked
        Dim instructionsScreen As New Instructions_Screen
        instructionsScreen.ShowDialog()
    End Sub

    Private Sub GenerateMazeButtonClick() 'runs when button with the text "Generate Maze" is clicked
        Dim mazeGenerationScreen As New Maze_Generation_Screen
        mazeGenerationScreen.ShowDialog() 'open the generate maze screen
    End Sub

    Private Sub ImportMazeButtonClick() 'runs when button with the text "Import Maze" is clicked
        Dim importMazeScreen As New Import_Maze_Screen
        importMazeScreen.ShowDialog() 'opens the import maze screen
    End Sub

    Private Sub ExitButtonClick() 'closes the entire program
        Me.Close()
    End Sub
```

End Class

6.2.18 Instructions_Screen Class

```
Imports System.Windows.Forms
Public Class Instructions_Screen
    Private Sub LoadInstructions() Handles MyBase.Load 'creates/loads instruction screen which outputs instructions

        Dim screenWidth As Integer = 750
        Dim screenHeight As Integer = 400

        Dim instructionsGroupBox As New GroupBox
        Dim groupBoxWidth As Integer = 500
        Dim groupBoxHeight As Integer = 300

        Dim instructions As New Label
        Dim labelWidth As Integer = 400
        Dim labelHeight As Integer = 250

        Dim backButton As New Button
        Dim backButtonWidth As Integer = 120
        Dim backButtonHeight As Integer = 75

        Me.Size = New Drawing.Size(screenWidth, screenHeight) 'defines size for window

        instructionsGroupBox.Name = "instructionsGroupBox"
        instructionsGroupBox.Location = New Drawing.Point(20, 20)
        instructionsGroupBox.Size = New Drawing.Size(groupBoxWidth, groupBoxHeight)


        instructions.Name = "instructions"

        instructions.Text = "
Instructions:

- Click the ‘Generate Maze’ button in order to generate a maze, you will be required to enter information about the maze you want to randomly generate into the input boxes provided on the screen. Once you have entered the required information press ‘Accept’, and then using the information a maze will be randomly generated and displayed for you. If you wish to save the generated maze just press ‘Save’, you will then be outputted the associated ID for the maze which you must remember.

- Click the ‘Import Maze’ button to import a maze (you must have first generated a maze to import one). To import a maze just input the maze’s associated ID. After that you can choose from a selection of algorithms to run on the maze. In addition to that, you can press ‘Compare Data’ to compare data about the maze such as the time taken for an specific algorithm to be run on it compared to another algorithm (or the same).
```

```
- Click the ‘Exit’ button to close the program.”

        instructions.Location = New Drawing.Point(25, 25)
        instructions.Size = New Drawing.Size(labelWidth, labelHeight)
        instructions.TextAlign = Drawing.ContentAlignment.TopLeft
        instructionsGroupBox.Controls.Add(instructions)

        Me.Controls.Add(instructionsGroupBox)


        backButton.Name = "backButton"
        backButton.Text = "Back"
        backButton.Location = New Drawing.Point(570, 25)
        backButton.Size = New Drawing.Size(backButtonWidth, backButtonHeight)
        backButton.FlatStyle = FlatStyle.Popup
        AddHandler backButton.Click, AddressOf Me.BackButtonClick
        Me.Controls.Add(backButton)

    End Sub

    Private Sub BackButtonClick() 'closes the instructions screen
        Me.Close()
    End Sub

End Class
```

6.2.19 Maze_Generation_Screen Class

```
Imports System.Drawing
Imports System.Windows.Forms

Public Class Maze_Generation_Screen

    'textboxes
    Private nameInputBox As New TextBox 'an object that can be written in on the windows form

    Private widthInputBox As New TextBox
    Private heightInputBox As New TextBox

    Private startSquareXInputBox As New TextBox
    Private startSquareYInputBox As New TextBox

    Private endSquareXInputBox As New TextBox
    Private endSquareYInputBox As New TextBox


    'Error Messages
    Private errorMName As New Label
```

```
    Private errorMWidth As New Label
    Private errorMHeight As New Label
    Private errorMStartSquX As New Label
    Private errorMStartSquY As New Label
    Private errorMEndSquX As New Label
    Private errorMEndSquY As New Label

    Private errorMessageArray() As Label = {errorMName, errorMWidth, errorMHeight, errorMStartSquX,
errorMStartSquY, errorMEndSquX, errorMEndSquY}

    'Maze
    Private generatedMaze As Maze

    Private Sub LoadScreen(sender As Object, e As EventArgs) Handles MyBase.Load 'when the window
loads this subroutine runs

        CreateMazeGenerationScreen() 'creates/loads the maze generation screen, the reason the code for
it is in a separate procedure and not within the LoadScreen subroutine is because it can be called upon
by a different procedure as well

    End Sub

    Public Sub CreateMazeGenerationScreen() 'creates/loads the maze generation screen where the user
generate a maze

        Dim nameGroupBox As New GroupBox '0 (position in groupBoxList)
        Dim mazeSizeGroupBox As New GroupBox '1 (position in groupBoxList), keeps information about
size on interface grouped together
        Dim coordinatesGroupBox As New GroupBox '2 (position in groupBoxList)
        Dim acceptInputsButton As New Button
        Dim backButton As New Button

        'Label (normal text)
        Dim nameLabel As New Label
        Dim widthLabel As New Label
        Dim heightLabel As New Label
        Dim startSquareLabel As New Label
        Dim endSquareLabel As New Label

        Me.Size = New Drawing.Size(750, 600) 'defines size for window

        '===== objects in the name input section =====

        nameGroupBox.Name = "nameGroupBox"
        nameGroupBox.Location = New Drawing.Point(10, 10)
        nameGroupBox.Size = New Drawing.Size(420, 100)

        nameLabel.Name = "nameLabel" 'identifier for the object (for windows forms)
        nameLabel.Text = "Name for maze:" 'text displayed by label
```

```
        nameLabel.Location = New Drawing.Point(25, 45) 'location of label
        nameGroupBox.Controls.Add(nameLabel) 'adds the label into the groupbox

        nameInputBox.Name = "nameInputBox" 'class variable
        nameInputBox.Location = New Drawing.Point(125, 42)
        nameGroupBox.Controls.Add(nameInputBox) 'adds the label into the groupbox

        errorMName.Name = "errorMName" 'class variable
        errorMName.Text = ""
        errorMName.Location = New Drawing.Point(25, 70)
        errorMName.Size = New Drawing.Point(390, 15) 'size of label
        errorMName.ForeColor = Drawing.Color.Red 'makes the colour the colour of the text red
        nameGroupBox.Controls.Add(errorMName) 'adds the label into the groupbox

        Me.Controls.Add(nameGroupBox) 'adds the nameGroupBox and everything it contains onto the
screen


        '===== objects in the size input section =====

        mazeSizeGroupBox.Name = "mazeSizeGroupBox"
        mazeSizeGroupBox.Location = New Drawing.Point(10, 100)
        mazeSizeGroupBox.Size = New Drawing.Size(420, 200)


        widthLabel.Name = "widthLabel"
        widthLabel.Text = "Width:"
        widthLabel.Location = New Drawing.Point(25, 70)
        mazeSizeGroupBox.Controls.Add(widthLabel)

        widthInputBox.Name = "widthInputBox"
        widthInputBox.Location = New Drawing.Point(125, 67)
        mazeSizeGroupBox.Controls.Add(widthInputBox)

        errorMWidth.Name = "errorMWidth"
        errorMWidth.Text = ""
        errorMWidth.Location = New Drawing.Point(25, 95)
        errorMWidth.Size = New Drawing.Point(390, 15)
        errorMWidth.ForeColor = Drawing.Color.Red
        mazeSizeGroupBox.Controls.Add(errorMWidth)


        heightLabel.Name = "heightLabel"
        heightLabel.Text = "Height:"
        heightLabel.Location = New Drawing.Point(25, 120)
        mazeSizeGroupBox.Controls.Add(heightLabel)

        heightInputBox.Name = "heightInputBox"
        heightInputBox.Location = New Drawing.Point(125, 117)
```

```
mazeSizeGroupBox.Controls.Add(heightInputBox)
Me.Controls.Add(mazeSizeGroupBox)

errorMHeight.Name = "errorMHeight"
errorMHeight.Text = ""
errorMHeight.Location = New Drawing.Point(25, 145)
errorMHeight.Size = New Drawing.Point(390, 15)
errorMHeight.ForeColor = Drawing.Color.Red
mazeSizeGroupBox.Controls.Add(errorMHeight)

'===== objects in the coordinate input section =====

coordinatesGroupBox.Name = "coordinatesGroupBox"
coordinatesGroupBox.Location = New Drawing.Point(10, 290)
coordinatesGroupBox.Size = New Drawing.Size(420, 250)


startSquareLabel.Name = "startSquareLabel"
startSquareLabel.Text = "Start Square Coordinates (x|y):"
startSquareLabel.Location = New Drawing.Point(25, 60)
startSquareLabel.Size = New Drawing.Point(100, 30)
coordinatesGroupBox.Controls.Add(startSquareLabel)

startSquareXInputBox.Name = "startSquareXInputBox"
startSquareXInputBox.Location = New Drawing.Point(125, 65)
startSquareXInputBox.Size = New Drawing.Point(30, 20)
coordinatesGroupBox.Controls.Add(startSquareXInputBox)

errorMStartSquX.Name = "errorMStartSquX"
errorMStartSquX.Text = ""
errorMStartSquX.Location = New Drawing.Point(25, 90)
errorMStartSquX.Size = New Drawing.Point(390, 15)
errorMStartSquX.ForeColor = Drawing.Color.Red
coordinatesGroupBox.Controls.Add(errorMStartSquX)

startSquareYInputBox.Name = "startSquareYInputBox"
startSquareYInputBox.Location = New Drawing.Point(175, 65)
startSquareYInputBox.Size = New Drawing.Point(30, 20)
coordinatesGroupBox.Controls.Add(startSquareYInputBox)

errorMStartSquY.Name = "errorMStartSquY"
errorMStartSquY.Text = ""
errorMStartSquY.Location = New Drawing.Point(25, 105)
errorMStartSquY.Size = New Drawing.Point(390, 15)
errorMStartSquY.ForeColor = Drawing.Color.Red
coordinatesGroupBox.Controls.Add(errorMStartSquY)


endSquareLabel.Name = "endSquareLabel"
```

```
endSquareLabel.Text = "End Square Coordinates (x|y):"
endSquareLabel.Location = New Drawing.Point(25, 130)
endSquareLabel.Size = New Drawing.Point(100, 30)
coordinatesGroupBox.Controls.Add(endSquareLabel)

endSquareXInputBox.Name = "endSquareXInputBox"
endSquareXInputBox.Location = New Drawing.Point(125, 135)
endSquareXInputBox.Size = New Drawing.Point(30, 20)
coordinatesGroupBox.Controls.Add(endSquareXInputBox)

errorMEndSquX.Name = "errorMEndSquX"
errorMEndSquX.Text = ""
errorMEndSquX.Location = New Drawing.Point(25, 160)
errorMEndSquX.Size = New Drawing.Point(390, 15)
errorMEndSquX.ForeColor = Drawing.Color.Red
coordinatesGroupBox.Controls.Add(errorMEndSquX)

endSquareYInputBox.Name = "endSquareYInputBox"
endSquareYInputBox.Location = New Drawing.Point(175, 135)
endSquareYInputBox.Size = New Drawing.Point(30, 20)
coordinatesGroupBox.Controls.Add(endSquareYInputBox)

errorMEndSquY.Name = "errorMEndSquy"
errorMEndSquY.Text = ""
errorMEndSquY.Location = New Drawing.Point(25, 175)
errorMEndSquY.Size = New Drawing.Point(390, 15)
errorMEndSquY.ForeColor = Drawing.Color.Red
coordinatesGroupBox.Controls.Add(errorMEndSquY)

Me.Controls.Add(coordinatesGroupBox)

'===== modification of the accept button =====

acceptInputsButton.Name = "acceptInputsButton"
acceptInputsButton.Text = "Accept"
acceptInputsButton.Location = New Drawing.Point(480, 125)
acceptInputsButton.Size = New Drawing.Size(200, 100)
acceptInputsButton.FlatStyle = FlatStyle.Popup
AddHandler acceptInputsButton.Click, AddressOf Me.AcceptInputsClick
Me.Controls.Add(acceptInputsButton)


backButton.Name = "backButton"
backButton.Text = "Back"
backButton.Location = New Drawing.Point(480, 275)
backButton.Size = New Drawing.Size(200, 100)
backButton.FlatStyle = FlatStyle.Popup
AddHandler backButton.Click, AddressOf Me.BackButtonClickInputScreen
Me.Controls.Add(backButton)
```

```
End Sub

Private Sub AcceptInputsClick() 'when the "Accept" button is pressed, this subroutine is run, it checks whether all the inputs are valid

    Dim validValues As Boolean = True


    Dim nameInput As String = ""
    Dim nameMaxLength As Integer = 20

    Dim widthInput As Integer
    Dim heightInput As Integer

    Dim startSquareXInput As Integer
    Dim startSquareYInput As Integer

    Dim endSquareXInput As Integer
    Dim endSquareYInput As Integer


    Dim rangeMin As Integer = 2
    Dim rangeMax As Integer = 30

    For i = 0 To errorMessageArray.Length - 1
        errorMessageArray(i).Text = ""
    Next

    Try
        nameInput = nameInputBox.Text
        If nameInput = "" Then
            Throw New EmptyInputBox
        ElseIf nameInput.Length > nameMaxLength Then
            Throw New InputLengthInvalid(nameMaxLength)
        End If
    Catch empty As EmptyInputBox
        validValues = False
        OutputError(nameInputBox, errorMName, empty)
    Catch tooLong As InputLengthInvalid
        validValues = False
        OutputError(nameInputBox, errorMName, tooLong)
    Catch invalidDataType As System.InvalidCastException
        validValues = False
        OutputError(nameInputBox, errorMName, invalidDataType)
    Catch otherError As Exception
        validValues = False
        OutputError(nameInputBox, errorMName, otherError)
```

```
End Try


Try
    widthInput = widthInputBox.Text

    If Convert.ToDouble(widthInput) <> Convert.ToDouble(widthInputBox.Text) Then
        Throw New RoundingOccurred
    ElseIf widthInput > rangeMax Or widthInput < rangeMin Then
        Throw New InputOutOfRange
    ElseIf widthInput = vbEmpty Then
        Throw New EmptyInputBox
    End If
Catch empty As EmptyInputBox
    validValues = False
    OutputError(widthInputBox, errorMWidth, empty)
Catch invalidDataType As System.InvalidCastException
    validValues = False
    OutputError(widthInputBox, errorMWidth, invalidDataType)
Catch outOfRange As InputOutOfRange
    validValues = False
    OutputError(widthInputBox, errorMWidth, outOfRange)
Catch decimalInput As RoundingOccurred
    OutputError(widthInputBox, errorMWidth, decimalInput)
Catch otherError As Exception
    validValues = False
    OutputError(widthInputBox, errorMWidth, otherError)
End Try


Try
    heightInput = heightInputBox.Text

    If Convert.ToDouble(heightInput) <> Convert.ToDouble(heightInputBox.Text) Then
        Throw New RoundingOccurred
    ElseIf heightInput > rangeMax Or heightInput < rangeMin Then
        Throw New InputOutOfRange
    ElseIf heightInput = vbEmpty Then
        Throw New EmptyInputBox
    End If
Catch empty As EmptyInputBox
    validValues = False
    OutputError(heightInputBox, errorMHeight, empty)
Catch invalidDataType As System.InvalidCastException
    validValues = False
    OutputError(heightInputBox, errorMHeight, invalidDataType)
Catch outOfRange As InputOutOfRange
    validValues = False
    OutputError(heightInputBox, errorMHeight, outOfRange)
```

```
Catch decimalInput As RoundingOccurred
    OutputError(heightInputBox, errorMHeight, decimalInput)
Catch otherError As Exception
    validValues = False
    OutputError(heightInputBox, errorMHeight, otherError)
End Try


Try
    startSquareXInput = startSquareXInputBox.Text

    If Convert.ToDouble(startSquareXInput) <> Convert.ToDouble(startSquareXInputBox.Text) Then
        Throw New RoundingOccurred
    ElseIf startSquareXInput > widthInput Or startSquareXInput < 1 Then
        Throw New OutOfMazeDimensions("X", widthInput)
    ElseIf startSquareXInput = vbEmpty Then
        Throw New EmptyInputBox
    End If
Catch empty As EmptyInputBox
    validValues = False
    OutputError(startSquareXInputBox, errorMStartSquX, empty)
Catch invalidDataType As System.InvalidCastException
    validValues = False
    OutputError(startSquareXInputBox, errorMStartSquX, invalidDataType)
Catch notInMaze As OutOfMazeDimensions
    validValues = False
    OutputError(startSquareXInputBox, errorMStartSquX, notInMaze)
Catch decimalInput As RoundingOccurred
    OutputError(startSquareXInputBox, errorMStartSquX, decimalInput)
Catch otherError As Exception
    validValues = False
    OutputError(startSquareXInputBox, errorMStartSquX, otherError)
End Try

Try
    startSquareYInput = startSquareYInputBox.Text

    If Convert.ToDouble(startSquareYInput) <> Convert.ToDouble(startSquareYInputBox.Text) Then
        Throw New RoundingOccurred
    ElseIf startSquareYInput > heightInput Or startSquareYInput < 1 Then
        Throw New OutOfMazeDimensions("Y", heightInput)
    ElseIf startSquareYInput = vbEmpty Then
        Throw New EmptyInputBox
    End If
Catch decimalInput As RoundingOccurred
    OutputError(startSquareYInputBox, errorMStartSquY, decimalInput)
Catch empty As EmptyInputBox
    validValues = False
    OutputError(startSquareYInputBox, errorMStartSquY, empty)
```

```
Catch invalidDataType As System.InvalidCastException
    validValues = False
    OutputError(startSquareYInputBox, errorMStartSquY, invalidDataType)
Catch notInMaze As OutOfMazeDimensions
    validValues = False
    OutputError(startSquareYInputBox, errorMStartSquY, notInMaze)
Catch otherError As Exception
    validValues = False
    OutputError(startSquareYInputBox, errorMStartSquY, otherError)
End Try


Try
    endSquareXInput = endSquareXInputBox.Text

    If Convert.ToDouble(endSquareXInput) <> Convert.ToDouble(endSquareXInputBox.Text) Then
        Throw New RoundingOccurred
    ElseIf endSquareXInput > widthInput Or endSquareXInput < 1 Then
        Throw New OutOfMazeDimensions("X", widthInput)
    ElseIf endSquareXInput = vbEmpty Then
        Throw New EmptyInputBox
    End If
Catch empty As EmptyInputBox
    validValues = False
    OutputError(endSquareXInputBox, errorMEndSquX, empty)
Catch invalidDataType As System.InvalidCastException
    validValues = False
    OutputError(endSquareXInputBox, errorMEndSquX, invalidDataType)
Catch notInMaze As OutOfMazeDimensions
    validValues = False
    OutputError(endSquareXInputBox, errorMEndSquX, notInMaze)
Catch decimalInput As RoundingOccurred
    OutputError(endSquareXInputBox, errorMEndSquX, decimalInput)
Catch otherError As Exception
    validValues = False
    OutputError(endSquareXInputBox, errorMEndSquX, otherError)
End Try

Try
    endSquareYInput = endSquareYInputBox.Text

    If Convert.ToDouble(endSquareYInput) <> Convert.ToDouble(endSquareYInputBox.Text) Then
        Throw New RoundingOccurred
    ElseIf endSquareYInput > heightInput Or endSquareYInput < 1 Then
        Throw New OutOfMazeDimensions("Y", heightInput)
    ElseIf endSquareYInput = vbEmpty Then
    ElseIf endSquareYInput = vbEmpty Then
        Throw New EmptyInputBox
    End If
```

```
        Catch empty As EmptyInputBox
            validValues = False
            OutputError(endSquareYInputBox, errorMEndSquY, empty)
        Catch invalidDataType As System.InvalidCastException
            validValues = False
            OutputError(endSquareYInputBox, errorMEndSquY, invalidDataType)
        Catch notInMaze As OutOfMazeDimensions
            validValues = False
            OutputError(endSquareYInputBox, errorMEndSquY, notInMaze)
        Catch decimalInput As RoundingOccurred
            OutputError(endSquareYInputBox, errorMEndSquY, decimalInput)
        Catch otherError As Exception
            validValues = False
            OutputError(endSquareYInputBox, errorMEndSquY, otherError)
        End Try

        If validValues = True Then 'to prevent the program from attempting to compare coordinates if they
are invalid

            Try
                If startSquareXInput = endSquareXInput And startSquareYInput = endSquareYInput Then
                    Throw New SameCoordinates
                End If
            Catch samePosition As SameCoordinates
                validValues = False
                OutputError(startSquareXInputBox, errorMStartSquX, samePosition)
                OutputError(startSquareYInputBox, errorMStartSquY, samePosition)
                OutputError(endSquareXInputBox, errorMEndSquX, samePosition)
                OutputError(endSquareYInputBox, errorMEndSquY, samePosition)
            Catch otherError As Exception
                validValues = False
                OutputError(startSquareXInputBox, errorMStartSquX, otherError)
                OutputError(startSquareYInputBox, errorMStartSquY, otherError)
                OutputError(endSquareXInputBox, errorMEndSquX, otherError)
                OutputError(endSquareYInputBox, errorMEndSquY, otherError)
            End Try
        End If

        If validValues = True Then
            GeneratingMaze(nameInput, widthInput, heightInput, startSquareXInput, startSquareYInput,
endSquareXInput, endSquareYInput)
        End If


    End Sub

    Public Sub SaveScreen() 'creates/loads the save screen where the user can choose to save the maze
after seeing what it looks like
```

```
        Dim saveScreenGroupBox As New GroupBox
        Dim mazeOutputBox As New Label
        Dim saveInputsButton As New Button
        Dim backButton As New Button

        Me.Controls.Clear()
        Me.Size = New Drawing.Size(800, 700)

        saveScreenGroupBox.Name = "saveScreenGroupBox"
        saveScreenGroupBox.Location = New Drawing.Point(1, 1)
        saveScreenGroupBox.Size = New Drawing.Size(Me.Width - 50, Me.Height - 10)

        mazeOutputBox.Name = "mazeOutputBox"
        mazeOutputBox.Text = generatedMaze.OutputMaze()
        mazeOutputBox.Location = New Drawing.Point(7, 7)
        mazeOutputBox.Size = New Drawing.Size(450, 650)
        mazeOutputBox.Font = New Font("Lucida Console", 8, style:=FontStyle.Regular) 'font that has
character's which have a fixed-width/monospaced (important when displaying maze)
        saveScreenGroupBox.Controls.Add(mazeOutputBox)


        saveInputsButton.Name = "saveInputsButton"
        saveInputsButton.Text = "Save"
        saveInputsButton.Location = New Drawing.Point(500, 100)
        saveInputsButton.Size = New Drawing.Size(200, 100)
        saveInputsButton.FlatStyle = FlatStyle.Popup
        AddHandler saveInputsButton.Click, AddressOf Me.SaveMaze
        saveScreenGroupBox.Controls.Add(saveInputsButton)

        backButton.Name = "backButton"
        backButton.Text = "Back"
        backButton.Location = New Drawing.Point(500, 250)
        backButton.Size = New Drawing.Size(200, 100)
        backButton.FlatStyle = FlatStyle.Popup
        AddHandler backButton.Click, AddressOf Me.BackButtonClickSaveScreen
        saveScreenGroupBox.Controls.Add(backButton)

        Me.Controls.Add(saveScreenGroupBox)

    End Sub

    Private Sub BackButtonClickInputScreen() 'if this subroutine is called upon (by pressing the back
button on the generate maze screen) it clears and closes the screen
        Me.Controls.Clear()
        Me.Close()
    End Sub

    Private Sub BackButtonClickSaveScreen() 'if this subroutine is called upon (by pressing the back
button on the save maze screen) it clears the screen and loads the maze generation screen
```

```
        Me.Controls.Clear()
        Me.CreateMazeGenerationScreen()
    End Sub


    Private Sub OutputError(ByVal textBox As TextBox, ByVal labelToChange As Label, ByVal errorType
As Exception) 'outputs/deals with errors
        textBox.Clear() 'clears invalid input
        labelToChange.Text = errorType.Message 'outputs new error message
    End Sub

    Public Sub GeneratingMaze(ByVal name As String, ByVal width As Integer, ByVal height As Integer,
ByVal startSquareX As Integer, ByVal startSquareY As Integer, ByVal endSquareX As Integer, ByVal
endSquareY As Integer) 'generates the maze

        Dim tempSquare As New Square(-1, -1)
        tempSquare.ResetNodeNumber()

        Dim startSqu As New StartSquare(startSquareX, startSquareY) 'creating start square
        Dim endSqu As New EndSquare(endSquareX, endSquareY) 'creating end square

        '======== Generation of maze ========'
        generatedMaze = New Maze(name, width, height, startSqu, endSqu) 'creates the maze using the
inputs
        SaveScreen()
        '====================================='
    End Sub

    Private Sub SaveMaze() 'saves the generated maze and outputs its ID in a MsgBox

        generatedMaze.SaveMaze() 'saves the maze
        MsgBox("The generated maze has an ID of: " & generatedMaze.GetID) 'outputs the ID of the maze
        Me.Close() 'closes the screen

    End Sub

End Class
```

6.2.20 Import_Maze_Screen Class

```
Imports System.Drawing
Imports System.Windows.Forms
Public Class Import_Maze_Screen
    Dim connection As New System.Data.Odbc.OdbcConnection("DRIVER={MySQL ODBC 5.3 ANSI
Driver};SERVER=localhost;PORT=3306;DATABASE=pathfindingdatabase;USER=root;PASSWORD=roo
t;OPTION=3;")

    Private idInputBox As New TextBox
```

```
    Private errorMId As New Label
    Private importedMaze As Maze 'the maze imported is assigned this variable

    Private Sub CreateImportMazeScreen(sender As Object, e As EventArgs) Handles MyBase.Load
'when this class loaded this subroutine is run, this subroutine creates/loads the screen that allows the
user to import a maze

        Dim idGroupBox As New GroupBox 'groupboxes are used to group together objects
        Dim idLabel As New Label

        Dim searchButton As New Button
        Dim backButton As New Button

        Me.Size = New Drawing.Size(750, 600) 'defines size for window

        idGroupBox.Name = "idGroupBox"
        idGroupBox.Location = New Drawing.Point(10, 10)
        idGroupBox.Size = New Drawing.Size(420, 100)

        idLabel.Name = "idLabel"
        idLabel.Text = "Enter ID of maze to import it:"
        idLabel.Location = New Drawing.Point(25, 45)
        idLabel.Size = New Drawing.Size(150, 25)
        idGroupBox.Controls.Add(idLabel)

        idInputBox.Name = "idInputBox"
        idInputBox.Location = New Drawing.Point(200, 42)
        idInputBox.Size = New Drawing.Size(75, 50)
        idGroupBox.Controls.Add(idInputBox)

        errorMId.Name = "errorMID"
        errorMId.Text = ""
        errorMId.Location = New Drawing.Point(25, 70)
        errorMId.Size = New Drawing.Point(390, 15)
        errorMId.ForeColor = Drawing.Color.Red
        idGroupBox.Controls.Add(errorMId)

        Me.Controls.Add(idGroupBox)

        searchButton.Name = "searchButton"
        searchButton.Text = "Search"
        searchButton.Location = New Drawing.Point(480, 125)
        searchButton.Size = New Drawing.Size(200, 100)
        searchButton.FlatStyle = FlatStyle.Popup
        AddHandler searchButton.Click, AddressOf Me.SearchButtonClick
        Me.Controls.Add(searchButton)


        backButton.Name = "backButton"
```

```
        backButton.Text = "Back"
        backButton.Location = New Drawing.Point(480, 275)
        backButton.Size = New Drawing.Size(200, 100)
        backButton.FlatStyle = FlatStyle.Popup
        AddHandler backButton.Click, AddressOf Me.BackButtonMenuScreen
        Me.Controls.Add(backButton)

    End Sub

    Private Sub SearchButtonClick() 'this subroutine is run when the search button is clicked, it attempts to search for the maze's ID inputted into the textbox in the pathfindingdatabase
        Dim id As Integer 'must save as ID, because the user may change the input within the input box while the program is running
        Dim tempSquare As New Square(-1, -1) 'used to reset the nodeNumber before start and end squares are created

        Try
            id = idInputBox.Text 'takes the input of the textbox and saves it in the id variable
            If Convert.ToDouble(id) <> Convert.ToDouble(idInputBox.Text) Then
                Throw New RoundingOccurred
            ElseIf SearchForMaze(id) = True Then 'if maze exists with the ID them it is imported as importedMaze ready to have algorithms run on it
            tempSquare.ResetNodeNumber() 'rests Shared variable to 1 as new maze
                importedMaze = New Maze(id) 'the maze is imported using the ID
                RunAlgorithmsScreen()
            Else
                Throw New MazeNotFound 'error which occurs when the maze with the associated ID cannot be found
            End If
        Catch notFound As MazeNotFound 'occurs when the maze with the associated ID cannot be found
            OutputError(idInputBox, errorMId, notFound)
        Catch invalidDataType As InvalidCastException 'occurs when the input is not an integer (so an invalid data type)
            OutputError(idInputBox, errorMId, invalidDataType)
        Catch otherError As Exception 'catches any other errors that occur
            OutputError(idInputBox, errorMId, otherError)
        End Try

    End Sub

    Private Sub BackButtonMenuScreen() 'if this subroutine is called upon (by pressing the back button on the run algorithms screen) it closes the connection as well as this screen
        connection.Close() 'closes connection
        Me.Close() 'closes this screen
    End Sub

    Private Sub BackButtonRunAlgorithmsScreen() 'if this subroutine is called upon (by pressing the back button on the compare data screen) it will load the 'RunAlgorithmsScreen'
        RunAlgorithmsScreen()
```

```
    End Sub

    Private Function SearchForMaze(ByVal id As Integer) 'searches maze with the inputted ID within the
'pathfindingdatabase'
        connection.Open() 'opens connection, to enable data to be retrieved/read from the
'pathfindingdatabase'

        Dim idFromTable As Odbc.OdbcDataReader
        Dim getID As New Odbc.OdbcCommand("SELECT mazeID FROM mazeinfo WHERE mazeID = '"
& id & "'", connection)

        idFromTable = getID.ExecuteReader()

        Try 'try catch to see if a maze with inputted ID exists in the database if it does not or the input is
invalid the code deals with it in the required way and outputs the associated error message
            If idFromTable.Read = True Then
                id = idFromTable.GetInt32(0)
            Else
                Throw New MazeNotFound
            End If
        Catch mazeNotExist As MazeNotFound
            connection.Close() 'closes connection
            Return False
        Catch empty As System.InvalidCastException
            connection.Close() 'closes connection
            Return False
        Catch otherError As Exception
            connection.Close() 'closes connection
            Return False
        End Try

        connection.Close() 'closes connection
        Return True

    End Function

    Private Sub OutputError(ByVal textBox As TextBox, ByVal labelToChange As Label, ByVal errorType
As Exception) 'outputs/deals with errors
        textBox.Clear() 'clears invalid input
        labelToChange.Text = errorType.Message 'outputs new error message
    End Sub

    Private Sub RunAlgorithmsScreen() 'loads the screen that allows the user to run algorithms on the
imported maze

        Dim optionsGroupBox As New GroupBox

        Dim mazeOutputBox As New Label
```

```
    Dim aStarSearchButton As New Button
    Dim breadthFirstSearchButton As New Button
    Dim dijkstrasAlgorithmButton As New Button
    Dim myAlgorithmButton As New Button '(extension)

    Dim compareDataButton As New Button

    Dim backButton As New Button

    Me.Controls.Clear()
    Me.Size = New Drawing.Size(830, 700) 'changes the size of the window/screen


    mazeOutputBox.Name = "mazeOutputBox"
    mazeOutputBox.Text = importedMaze.OutputMaze() 'the maze as a string is made to be the text of
the label
    mazeOutputBox.Location = New Drawing.Point(27, 7)
    mazeOutputBox.Size = New Drawing.Size(450, 650)
    mazeOutputBox.Font = New Font("Lucida Console", 8, style:=FontStyle.Regular) 'I choose this font,
as it has characters which have a fixed-width/monospaced (which is important when displaying maze)
    Me.Controls.Add(mazeOutputBox)


    optionsGroupBox.Name = "optionsGroupBox"
    optionsGroupBox.Location = New Drawing.Point(500, 25)
    optionsGroupBox.Size = New Drawing.Size(250, 600)


    aStarSearchButton.Name = "AStarSearchButton"
    aStarSearchButton.Text = "A Star Search"
    aStarSearchButton.Location = New Drawing.Point(50, 25)
    aStarSearchButton.Size = New Drawing.Size(150, 50)
    aStarSearchButton.FlatStyle = FlatStyle.Popup
    AddHandler aStarSearchButton.Click, AddressOf Me.RunAStar
    optionsGroupBox.Controls.Add(aStarSearchButton)

    breadthFirstSearchButton.Name = "BreadthFirstSearchButton"
    breadthFirstSearchButton.Text = "Breadth First Search"
    breadthFirstSearchButton.Location = New Drawing.Point(50, 125)
    breadthFirstSearchButton.Size = New Drawing.Size(150, 50)
    breadthFirstSearchButton.FlatStyle = FlatStyle.Popup
    AddHandler breadthFirstSearchButton.Click, AddressOf Me.RunBreadthFirst
    optionsGroupBox.Controls.Add(breadthFirstSearchButton)

    dijkstrasAlgorithmButton.Name = "DijkstrasAlgorithmButton"
    dijkstrasAlgorithmButton.Text = "Dijkstra's Algorithm"
    dijkstrasAlgorithmButton.Location = New Drawing.Point(50, 225)
    dijkstrasAlgorithmButton.Size = New Drawing.Size(150, 50)
    dijkstrasAlgorithmButton.FlatStyle = FlatStyle.Popup
```

```
    AddHandler dijkstrasAlgorithmButton.Click, AddressOf Me.RunDijkstras
    optionsGroupBox.Controls.Add(dijkstrasAlgorithmButton)

    myAlgorithmButton.Name = "myAlgorithmButton" 'extension
    myAlgorithmButton.Text = "My Algorithm"
    myAlgorithmButton.Location = New Drawing.Point(50, 325)
    myAlgorithmButton.Size = New Drawing.Size(150, 50)
    myAlgorithmButton.FlatStyle = FlatStyle.Popup
    AddHandler myAlgorithmButton.Click, AddressOf Me.RunMyAlgorithm
    optionsGroupBox.Controls.Add(myAlgorithmButton)


    compareDataButton.Name = "compareDataButton"
    compareDataButton.Text = "Compare Data"
    compareDataButton.Location = New Drawing.Point(50, 425)
    compareDataButton.Size = New Drawing.Size(150, 50)
    compareDataButton.FlatStyle = FlatStyle.Popup
    AddHandler compareDataButton.Click, AddressOf Me.CompareDataScreen
    optionsGroupBox.Controls.Add(compareDataButton)


    backButton.Name = "backButton"
    backButton.Text = "Back"
    backButton.Location = New Drawing.Point(50, 525)
    backButton.Size = New Drawing.Size(150, 50)
    backButton.FlatStyle = FlatStyle.Popup
    AddHandler backButton.Click, AddressOf Me.BackButtonMenuScreen
    optionsGroupBox.Controls.Add(backButton)


    Me.Controls.Add(optionsGroupBox)

End Sub

Private Sub RunAStar() 'runs the AStar search on the maze
    importedMaze.RunAlgorithm(Algorithm.Algorithms.AStar)
End Sub

Private Sub RunBreadthFirst() 'runs the BreadthFirst search on the maze
    importedMaze.RunAlgorithm(Algorithm.Algorithms.BreadthFirst)
End Sub

Private Sub RunDijkstras() 'runs the Dijkstras algorithm on the maze
    importedMaze.RunAlgorithm(Algorithm.Algorithms.Dijkstras)
End Sub

Private Sub RunMyAlgorithm() 'runs myAlgorithm search on the maze
    importedMaze.RunAlgorithm(Algorithm.Algorithms.MyAlgorithm)
End Sub
```

```
    Private Function CreateLabel(ByVal text As String) As Label 'returns a label that was created using the parameter input
        Dim newLabel As New Label
        Dim newGroup As New GroupBox
        newLabel.Name = (text & "Label") 'given name for label
        newLabel.Text = text 'assigning the parameter input as the associated text of the label
        newLabel.AutoSize = True 'the label can change size if it needs to in order to fit the text into the label
        newLabel.Width = 120 'the original width of the label
        newLabel.Height = 150 'the original height of the label
        Return newLabel 'returns the created label
    End Function

    Private Function CreateTextBox(ByVal text As String) As TextBox 'returns a textbox that was created using the parameter input. A textbox is created so that the output inside it can be scrolled as textboxes can have scrollbars whereas labels cannot
        Dim newTextBox As New TextBox 'modified textbox so that behaviour is similar to a label's
        newTextBox.Name = (text & "TextBox") 'given name for textbox
        newTextBox.Text = text 'assigns the parameter input to be the text outputted in the textbox
        newTextBox.Multiline = True 'lets the text in the textbox be on multiple lines and not just one
        newTextBox.AutoSize = True 'enables the textbox to change size to fit the required data
        newTextBox.ReadOnly = True 'so no data can be inputted, only outputted (this is what makes it behave similar to a label)
        newTextBox.Width = 300 'the original width of the textbox
        newTextBox.Height = 150 'the original height of the textbox
        newTextBox.ScrollBars = ScrollBars.Vertical 'enables the vertical scrollbar within the textbox
        newTextBox.MaximumSize = New Size(400, 10000) 'maxiumum size of textbox
        Return newTextBox 'returns the created textbox
    End Function

    Private Sub CompareDataScreen() 'loads the screen that allows the user to compare data about process previously run on the imported maze
        Dim processData As List(Of String())
        Dim tableForProcessData As New TableLayoutPanel
        Dim backButton As New Button
        Dim columnNumber As Integer = 5

        Me.Controls.Clear() 'clears the window/screen

        tableForProcessData.Name = "tableForProcessData"
        tableForProcessData.Location = New Drawing.Point(10, 10)
        tableForProcessData.Size = New Size(775, 500)
        tableForProcessData.AutoScrollMinSize = New Size(0, 0)
        tableForProcessData.BorderStyle = BorderStyle.FixedSingle
        tableForProcessData.ColumnCount = columnNumber
        For i = 0 To columnNumber - 1
            tableForProcessData.ColumnStyles.Add(New ColumnStyle(SizeType.AutoSize))
        Next
```

```
        tableForProcessData.CellBorderStyle = BorderStyle.FixedSingle
        tableForProcessData.GrowStyle = TableLayoutPanelGrowStyle.AddRows

        backButton.Name = "backButton"
        backButton.Text = "Back"
        backButton.Location = New Drawing.Point(600, 550)
        backButton.Size = New Drawing.Size(150, 75)
        backButton.FlatStyle = FlatStyle.Popup
        AddHandler backButton.Click, AddressOf Me.BackButtonRunAlgorithmsScreen
        Me.Controls.Add(backButton)


        processData = importedMaze.LoadProcess()
        tableForProcessData.RowStyles.Add(New RowStyle(SizeType.AutoSize)) 'adds a new row
        tableForProcessData.RowCount += 1

        tableForProcessData.Controls.Add(CreateLabel("processID"), 0, tableForProcessData.RowCount -
1)
        tableForProcessData.Controls.Add(CreateLabel("algorithm"), 1, tableForProcessData.RowCount -
1)
        tableForProcessData.Controls.Add(CreateLabel("path"), 2, tableForProcessData.RowCount - 1)
        tableForProcessData.Controls.Add(CreateLabel("timeTaken"), 3, tableForProcessData.RowCount -
1)
        tableForProcessData.Controls.Add(CreateLabel("numberOfSquaresTraversed"), 4,
tableForProcessData.RowCount - 1)

        For i = 0 To (processData.Count - 1)
            tableForProcessData.RowCount += 1 'the numerical row count is increased
            For k = 0 To processData(i).Length - 1
                If k = 2 Then 'when k=2 the current information being dealt with is the path from the start
square to the end square, as this is long it is outputted in a textbox
                    tableForProcessData.Controls.Add(CreateTextBox((processData(i))(k)), k,
tableForProcessData.RowCount - 1) 'the data is outputted in a textbox which is put into a cell within the
table
                Else
                    tableForProcessData.Controls.Add(CreateLabel((processData(i))(k)), k,
tableForProcessData.RowCount - 1) 'the data is outputted in a label which is put into a cell within the
table
                End If
                tableForProcessData.RowStyles.Add(New RowStyle(SizeType.AutoSize)) 'the row count for
row style
            Next
        Next

        'scrollbars
        tableForProcessData.AutoScroll = True
        tableForProcessData.AutoScrollPosition = New Point(0,
tableForProcessData.VerticalScroll.Maximum)
        tableForProcessData.VerticalScroll.Enabled = True
```

```
        Me.Controls.Add(tableForProcessData) 'adds the table to the screen

    End Sub

End Class
```

6.3 Code to create SQL Database

SQL Database creation code:

Creation of the database (pathfindingdatabase)

```
CREATE DATABASE `pathfindingdatabase`
```

Creation of the mazeinfo table:

```
CREATE TABLE `mazeinfo` (
  `mazeID` int(3),
  `mazeName` varchar(20),
  `mazeWidth` int(2),
  `mazeHeight` int(2),
  `startSquareX` int(2),
  `startSquareY` int(2),
  `endSquareX` int(2),
  `endSquareY` int(2),
  `wallStatus` varchar(3721)
)
```

Creation of the typesofalgorithms table:

```
CREATE TABLE `typesofalgorithms` (
  `algorithmID` int(2) NOT NULL,
  `algorithmName` varchar(20) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  PRIMARY KEY (`algorithmID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Creation of the algorithmsrun table:

```
CREATE TABLE `algorithmsrun` (
  `processID` int(20),
  `mazeID` int(20),
  `algorithmID` int(2),
```

```
  `pathToEnd` varchar(3600),
  `timeTakenToRun` varchar(20),
  `numberOfSquaresTraversed` int(3),
  PRIMARY KEY (`processID`)
)
```