1 Analysis Section

1.1 Introduction

is a PHD student at Sunderland University who also owns a business called All Programming Limited. aims to become a computer science university lecturer after he has completed his PHD. He will also maintain his business. would like a range of educational tools to use as part of his future teaching and foresees teaching the topic of 'shortest path' as part of his duties. He wants his future students to appreciate that there are many searching algorithms already in existence, some are dedicated shortest path algorithms whilst others are not but can be tailored to show the shortest path. would like a program to demonstrate three of the existing searching algorithms including one that is a dedicated shortest path algorithm. He would like the user to be able to specify a target, seeker and closed nodes and for the program to then show the nodes that have been visited in each search. This will allow them to determine which is the most efficient under particular circumstances by the amount of nodes visited to find the target. An acceptable limitation will be actually showing the shortest path. I will need to research different algorithms in order to select the three that the program will demonstrate.

I foresee the system using a grid, whereby the user can place targets and seekers in particular co-ordinates and add closed sections where traversal cannot take place. The system will show all nodes visited in order to arrive at the target. From this, the user can see which is the most efficient under those conditions. There is no current system in place as in does not have a program that does this. In terms of algorithms, though there are many and three of these will be utilised in the new program. I will need to research these.

1.2 Research

As part of my A Level Computer Science course, I have already studied breadth-first and depth-first algorithms. From research I have found another which is known A*. These are the three, which I have decided to include. I did initially think of including Dijkstras but I think A* is a more efficient and interesting algorithm. My client agreed with my choices. So therefore, the different algorithms I am going to use are A* search, Breadth First Search and Depth First Search. I think these are the best algorithms for me to use compared to others as it will give me the ability to program them and find the shortest path more fast and efficiently and my client agrees with the use of them.

1.2.1 Breadth First Search

A Breadth first search[1] works by starting at the Root node then moving from left to right whilst working down the graph. A standard algorithm for this is:

```
FUNCTION bfs(graph, vertex)
BEGIN
   queue ← []
   visited ← []
   enqueue vertex
   WHILE queue NOT EMPTY DO
        dequeue item and put in currentNode
        set colour of currentNode to "dark blue"
        append currentNode to visited
     FOR each neighbour of currentNode DO
        IF colour of neighbour = "white" THEN
                enqueue neighbour
                set colour of neighbour to "pale blue"
```

Explanation

1. Set the Queue and Visited as empty.
2. Put the current vertex in the queue.
3. While the queue is not empty keep doing this.
4. Take the Next vertex in line in the queue and put it in the CurrentNode set the current node to a certain colour.
5. Set current node as visited.
6. For each neighbour do if the neighbour is white then put it on the queue and change colour to pale blue

[1] PowerPoint Slides

END IF
END FOR
END WHILE
RETURN visited
END FUNCTION
	7. End all statements and after end of while statement return visited list.

I have made these diagrams to demonstrate the first level search. Yellow is used as the node visited.

Step 1

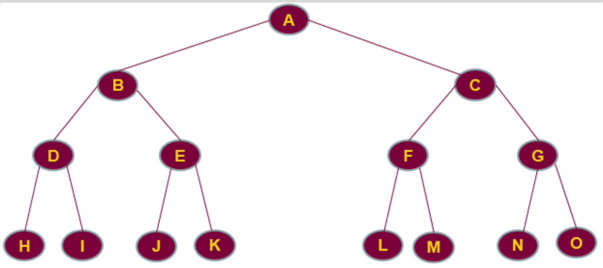


Figure 1:1st Step

This would be the first step. Nothing has been traversed as of yet but (A) is the root node.

Step 2

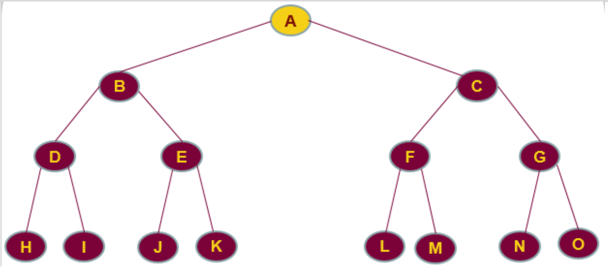


Figure 2: Second Step

The root node has now been traversed.

Step 3

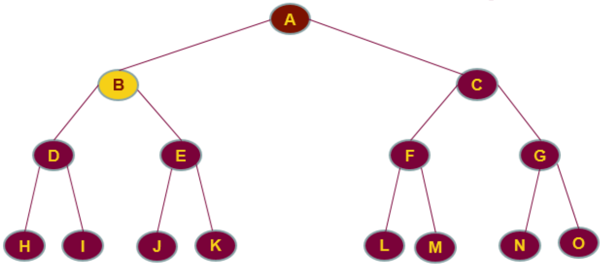


Figure 3: 3rd Step

The next node in the left sub-tree is traversed

Step 4

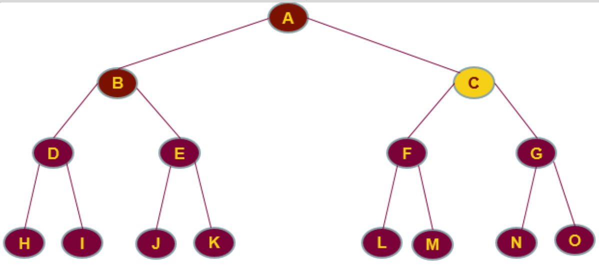


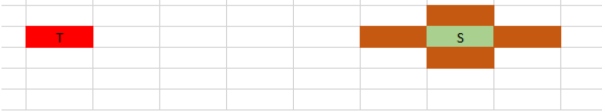
Figure 4: 4th Step

And then it moves to the next node in the right sub-tree and so on.

Once the entire traversal has taken place the order in which they would have been visited is: (A,B,C,D,E,F,G,H,I,J,K,L,M,N,O).

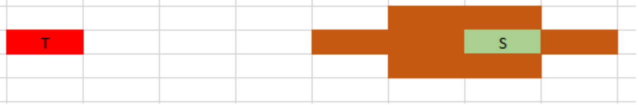
On a grid, it would look like a diamond spreading out from the seeker, as it will work across from the left first then up then bottom then right and repeat whilst moving in this order. I have created a spreadsheet to demonstrate this:

Step 1

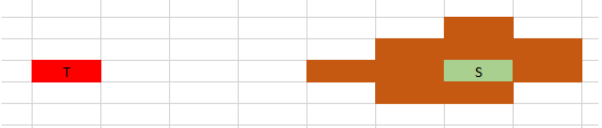
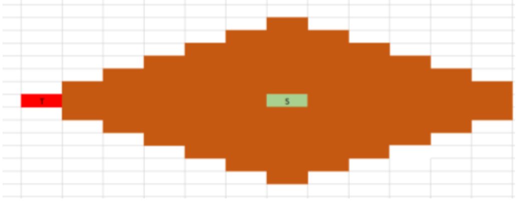


In this step, it has checked left, right, up and down. Giving the diamond pattern shape.

Step 2



Notice how it has moved left and repeated the steps of checking Left, Up, Down and Right. It will now move back to the up one above the Seeker we checked.

Step 3	Step 4
 When it has gone to the block that has already been checked it will then proceed to check again Left, Up, Down and Right. It will continue to follow this suit until it has found the Target.	 You will notice that the Diamond shape is not absolutely perfect as it is missing a bit on the right. This is because it has already Found the Target before having to check there.

I think this search will be a good search to include in the program though the algorithm will need to be tailored in order to be suitable for the program. My program will need to resemble the spreadsheet screen prints I included as opposed to simply changing the colour of a vertex. Once the search is completed, the nodes visited will be shown and the user will need to be determine which is the shortest path.

1.2.2 Depth First Search

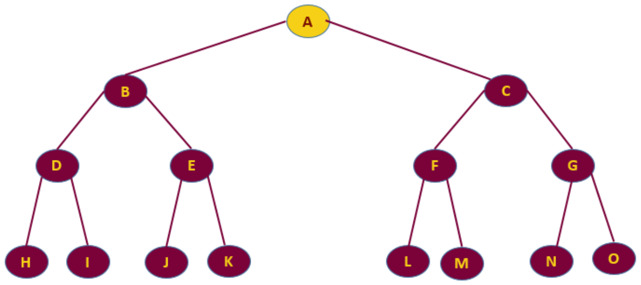
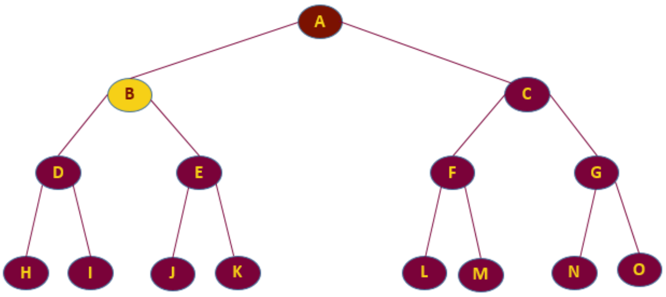
A Depth First Search[2] works by starting at the Root node working down first then moving left to right. A standard algorithm for this is:

```
visitedList = [ ]
FUNCTION dfs(graph, currentVertex, visited)
Begin
        FOR vertex IN  graph[currentVertex] DO
                IF vertex NOT IN visited THEN
                        dfs(graph, vertex, visited)
                END IF
        END FOR
        Return visited
END FUNCTION
Traversal = dfs(GRAPH, "A", visitedList)
OUTPUT "Nodes visited in this order: ",traversal
```

Explanation

1. an empty list of visited nodes (vertices) is created
2. the Function dfs is called with parameters being passed of the graph, the current vertex and the visited list
3. Each vertex in the graph is checked to see if if it is in the visited list. If it is not the current Vertex is appended to the list visited nodes and the neighbours checked
4. The nodes visited are then displayed in order.

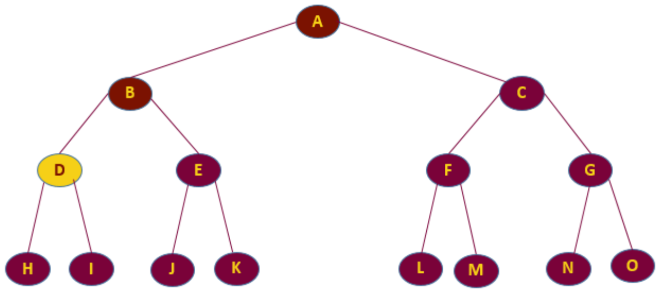
This demonstrates part of the search. Yellow is used as the node visited.

Step 1	Step 2
 A B C D E F G H I J K L M N O This is the first node being traversed	 A B C D E F G H I J K L M N O

[2] PowerPoint Slides

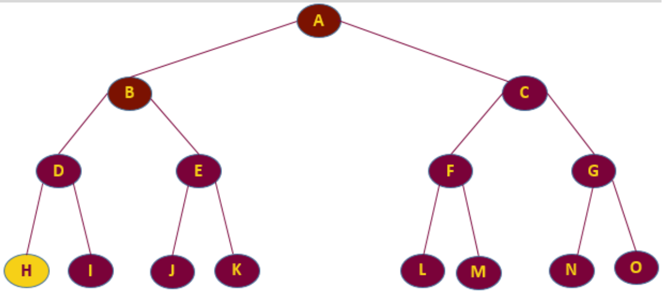
At step 2 it still looks like the same traversal as depth-first in that the next node in the left sub-tree is examined.

Step 3



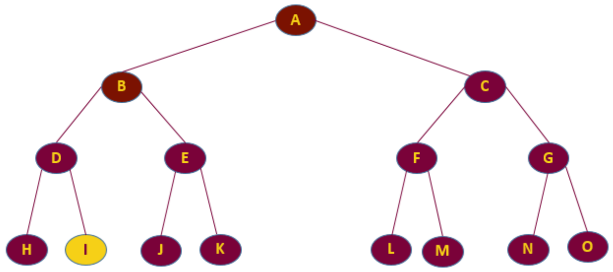
However, this is where it now differs; it continues to traverse the left sub-trees.

Step 4



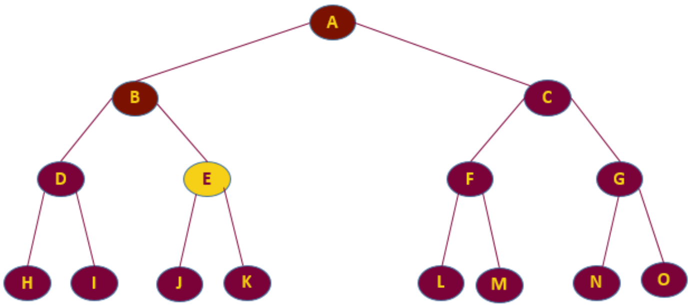
This is the final left sub-tree it can traverse from D

Step 5



So it now moves the right sub-tree from D

Step 6

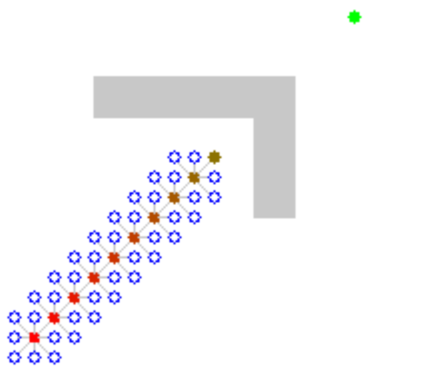
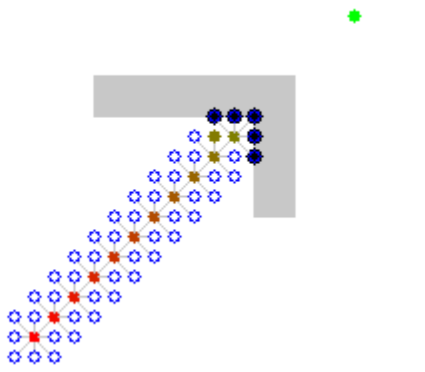
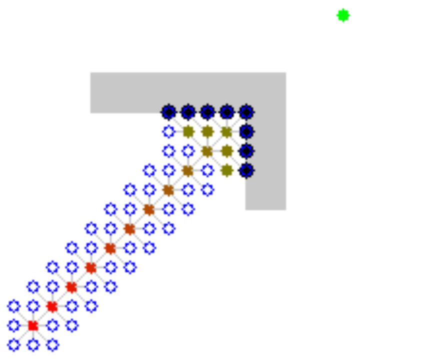
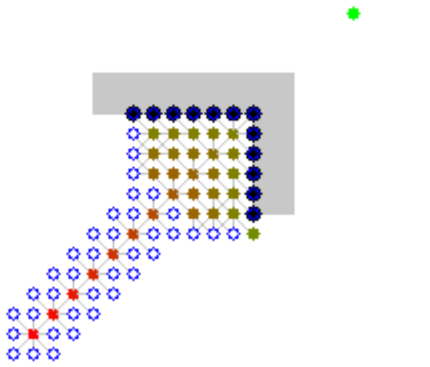
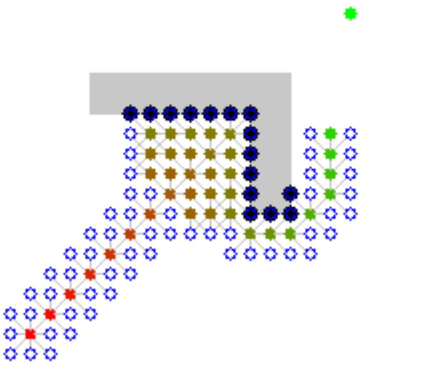
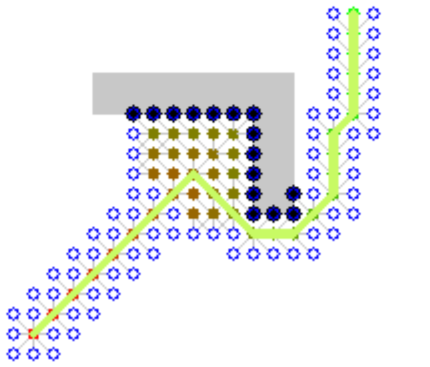


D has now been fully explored so it moves back to B. The left sub-tree of B has now been fully explored so it moves to the right sub-tree of B and so on.

So the order that these would be traversed and the output would be (A,B,D,H,I,E,J,K,C,F,L,M,G,N,O). I also think this will be a good search to include in the program though, again, it will need customising in order to fit the program. My client agreed that this would be a good search to include.

1.2.3 A* Search – More Complex Algorithm

The A* algorithm was invented by Nils Nilsson in 1964. He invented an algorithm called A1, which increased the speed of Dijkstra's algorithm. Following that, Bertram Raphael made improvements calling this search A2. A man named Peter Hart argued and proved that A2 was better than A1 naming it A* to show that it includes any search algorithm beginning with A no matter what number followed it[3]. This is an example[4] of an A* search being carried out.

Step 1	Step 2	Step 3
 It works from the root node and looks at the possible ways it could go and decides to go the way that ends up closer to the target.	 It then hits the wall so back tracks the nodes it has visited.	 It keeps doing this until it can get around the blockage.
Step 4	**Step 5**	**Step 6**
 Once it has found a way around it will then carry on as before.	 It still recognises that there is a blockage but still carry's on rather than backtracking because it is still shortening the distance between the two by carrying on.	 Once it has got to the target it then backtracks to the original starting node using the nodes it has already visited to mark the shortest path.

This is the pseudocode[5]

```
    // A*
    initialize the open list
    initialize the closed list
    put the starting node on the open list (you can leave its f at zero)

    while the open list is not empty
        find the node with the least f on the open list, call it "q"
```

[3] http://stackoverflow.com/questions/29470253/astar-explanation-of-name

[4] https://www.reddit.com/r/programming/comments/1cylmb/pathfinding_algorithm_visually_explained/

[5] http://web.mit.edu/eranki/www/tutorials/search/

```
    pop q off the open list
    generate q's 8 successors and set their parents to q
    for each successor
       if successor is the goal, stop the search
        successor.g = q.g + distance between successor and q
        successor.h = distance from goal to successor
        successor.f = successor.g + successor.h

        if a node with the same position as successor is in the OPEN list \
            which has a lower f than successor, skip this successor
        if a node with the same position as successor is in the CLOSED list \
            which has a lower f than successor, skip this successor
        otherwise, add the node to the open list
    end
    push q on the closed list
end
```

The main differences seem to be that A* does not necessarily need to work on a graph. It does need to know the distance away from the target at every node and the cost of going down that path. This is very different to breadth and depth as it means a judgement can be made on which is the most efficient way to go. I think a stack could possibly be used to represent the search with the cost of the path worked out as f=g + h. g = the cost it took to get to the current node and h = the guess of the cost to get to the goal from the current node. I think this algorithm will be a good one to include, as it is very different from the other two. My client agreed that this would be a good search to include.

1.3 Analysis Data Dictionary

The only attributes that I picture being required in the main program will be discussed here. Properties of classes will be discussed within the classes themselves.

Attribute Name	Attribute Purpose	Attribute Type	Example Data	Validation
X size	To store the x axis number input by the user	Integer	5	Must between 4 and 10
Y size	To store the y axis number input by the user	Integer	5	Must between 4 and 10
Seeker	To make sure one and only one seeker has been selected	Boolean	True	Must be only one seeker
Target	To make sure one and only one target has been selected	Boolean	True	Must be only one target
stopChange	Stop the user from changing the grid items if the save has already been started	Boolean	False	n/a
ConnectionsFrom	To hold the connections from other grid items to the current grid item	List of string	0,1	n/a

ConnectionsTo	To hold the connections to other grid items to the current grid item	List of string	1,1	n/a
GridItemDictionary	Dictionary to hold connections and the current state of the grid item (seeker, target, open, closed or visited). Also to hold the position of x and the position y of the current grid item. Will interact with the TGridItemClass	string	Parent:0,1 Left:1,1 Right:1,2 Up:2,2 Down:1,3 Seeker Open Visited Pos x Pos y	n/a

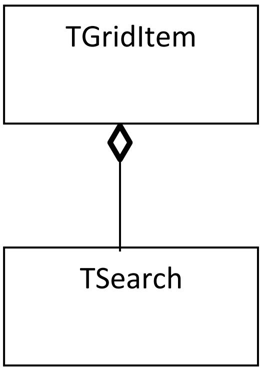
1.4 Object Oriented Planning

1.4.1 Class Diagrams

Initially, I think there will be a need for two classes and they will need to relate to each other through composition:

TGridItem will be a class that stores the data of each object and its connections. I will explain this in more detail below.

TSearch will be a class that creates instances of TGridItem and to then access the methods of that class in order to set and amend its properties. It will also include its own methods in order to carry out each of the three searches.



1.4.2 TGridItem Class

TGridItem
Private
CurrentState : string Posx:Integer Posy:Integer leftConnection : TGridItem rightConnection : TGridItem topConnection : TGridItem bottomConnection : TGridItem parentNode : TGridItem visited : Boolean
Public
GetGridItem(pCurrentState,pPosx,pPosy, pleftConnection,prightConnection,ptopConnection, pbottomConnection,pparentNode,pvisited)

For each of my attributes there will be a straightforward setter and getter and I am not showing in order to save space. I have shown the GetGridItem method as passing the required attributes to store. Current state will be needed to determine what the current node is as in if it has been visited or it may be a seeker, Target, Closed or Open.

Posx will be needed to store how many nodes are in the X axis and Posy is the same except for the y axis instead.

The left, right, top and bottom connections are used to store where the nodes are connected and whether there is a node there at all. The parent node is the node that will store where the move has came from. Visited is needed to look upon when the algorithms check if it has already been visited or not which will then determine if it should then move to that node or not.

1.4.3 TSearch Class

The Search Class will create instances of TGridItem using composition. There will be no requirement for any attributes but there will be a requirement for parameters to be passed to each method to provide each search with the data it needs to be able to run the search.

TSearch
Private
Public
RunBreadthFirstSearch(RootNode : TGridItem) RunDepthFirstSearch(RootNode : TGridItem) RunAStarSearch(rootNode, targetNode : TGridItem)

1.5 Users and User Needs

Michael will use the program to demonstrate searching during his lectures. For example, he could set up the grid and ask what traversal would be carried out by a particular algorithm. The responses could be checked by running the program. The university students will be able to use the program outside of lessons. This will be beneficial aiding the visualisation of search and shortest path algorithms which should help them compete their assignments or exams.

Michael would like the program to

1. Allow the user to specify the size of the grid
2. Allow the user to specify the co-ordinate of the target
3. Allow the user to specify the co-ordinate of the seeker
4. Allow the user to specify closed blocks
5. Allow the user to specify the type of search to be carried out
    a. Breadth-first
    b. Depth-first
    c. A*
6. Show the user the path taken to get from the target to the seeker

Acceptable limitation

7. Show the user the shortest path

1.6 Objectives of New System

1.6.1 System Start up

On the system start-up, it will show all the buttons and labels etc. but will not allow the user to edit them apart from the section for setting up the grid. These are the detailed start-up objectives:

1. The search form should load
2. There should be an empty panel on the form
3. There should be an edit box to enter the x axis of the grid.
4. There should be an edit box to enter the y axis of the grid.
5. There should be a create button
6. There should be a radio group with options of
    a. Target – this will be used to specify whether the grid item is the target
    b. Seeker – this will be used to specify whether the grid item is the seeker
    c. Open – this will be used to specify that the grid item can be traversed and should be selected by default
    d. Closed – this will be used to specify that the grid item cannot be traversed
    e. Instructions telling the user they can only select one target should be visible near the radio group

1.6.2 Create the Grid

This will be the first task for the user as they will need to input what size they would like it I will need to put validation into this task as the user may create a grid size too large or too small to not be used efficiently. It will set all the blocks in the grid as open/empty meaning anything can pass through it. Once it has been set up the user will then be able to select 1 of 4 options at a time between (Open, Target, Seeker and Closed). This will allow the user to place 1 Target and 1 Seeker on the grid as the Seeker will be the root (Starting) node and the Target being the end (Finish) node. There will be an option for and Open block as this means if the user accidently places a different block in the wrong place the user can fix this error by replacing it with an Open Block. The fourth option is the Closed block where users have the ability to block off a curtain path so when the search algorithm is carried out it will not be able to use/cross over this section. This will give the user the ability to test real life scenarios where there will be blockages in the way of a path and will have to take a different route.

7. Validation should be carried out to ensure the x and y axis are numbers and that they are between 4 and 10
8. The grid should be created:
    a. Individual grid item width = panel width/x axis number
    b. Individual grid item height = panel height/y axis number
    c. At run time repeatedly load an image for each grid item (up to x axis * y axis)
    d. Ensure each image is relevant to the type of grid item
        i. Open = white image
        ii. Closed = white background with red cross
        iii. Target = blue background with black T in the centre
        iv. Seeker = green background with black S in the centre

1.6.3 Saving the Grid Locations

Once the user has set up the size of the grid and decided the layout i.e. where they want the target and seeker to be and any closed blocks. There will be a validation in place to make sure both one Seeker and one Target has been placed. Then the ability to click on a *Save Locations* button will become available. They will then press the *Save Locations* button and this will disable the user's ability to change the grid afterwards. This will also establish the connection to each node and what the state of it is whether it is Open, Target, Seeker etc. Once this is complete, it will give the user the ability to select which search algorithm they would like to run whilst disabling the ability to press the *Save Locations* button again.

9. Ensure target grid has been specified and that the user cannot specify more than one target
10. Ensure seeker grid has been specified and that the user cannot specify more than one seeker
11. Determine the neighbours for each open grid item
    a. Start at 0,0
    b. If there is a neighbour above and it is open, add a connection between the two grid items
    c. If there is a neighbour to the left and it is open, add a connection between the two grid items
    d. If there is a neighbour to the right and it is open, add a connection between the two grid items
    e. If there is a neighbour below and it is open, add a connection between the two grid items
    f. Move to 0,1 etc. until all grid items have been explored and connection stored

1.6.4 Running a Search

The user will select which algorithm they would like to run by using a radio group. Whether they have actually selected an algorithm will be validated. By default, the first radio group option will be selected. Once the user has selected which search they want to run they will then have to click on the *Search* button. This will run a Procedure to save all the connections in between the nodes. Once the connections have been saved it will check to see which algorithm the user has selected and will continue to run it.

12. If the user has selected the breadth-first or depth-first search
    a. Get the seeker grid item and use this as the root node
    b. Define graph based on where the seeker can move based using the stored connections
    c. Run the breadth-first or depth-first search accordingly using the graph created
        i. For every grid item visited during the search change the current image to an orange image to show it has been visited
        ii. Stop the search when the target has been reached
13. If the user has selected the A* search
    a. Get the seeker grid item and use this as the root node
    b. Move through each grid item and calculate the ***distance to target*** from the current grid item using Pythagoras c^2=a2+b^2
    c. Define graph based on where the seeker can move based using the stored connections
    d. Run the A* search using the graph created
        i. Use the calculation ***Next move*** = ***distance to target + 1 (cost of making the move)*** for each connected grid item
        ii. Determine the lowest value calculated for next move and make the current grid item image change to an orange image to show it has been visited
        iii. Stop the search when the target has been reached

1.6.5 Reset

This will be the ability for the user to completely restart the program if they have either made a mistake when they pressed the *Save Locations* button and want to undo the changes or have run a search and want to change to another.

14. Destroy all grid items
15. Reset all form defaults

1.7 Research Methods used

I have interviewed my client in order to determine what the project was to be based on. I have informally communicated with my client using email and arranging short meetings with him where we could discuss the program and whether I understand his needs fully (these are included in the appendix section). I have also used lots of research methods in order to decide which three algorithms to base my project on such as looking at how each one works online and its efficiency. I used my teacher's workbooks to learn how to use forms and program using object orientated techniques. I have used her PowerPoint slides to refresh my memory about breadth-first, depth-first and Dijkstra's algorithms. I used websites and spoke with my Maths teacher to find out more information on how Dijkstra's works and weighed up the pros and cons of each. I researched the stress on memory of each search in order to help with my selection. I have referenced where I needed to in the analysis section to show where they were useful in my project. I have included references as footnotes etc. in my work.