

---

# Teacher Standardisation

Spring 2018

---

A-level Computer Science (7517)

---

# Booklet 3

## Contents

1	Analysis Section.....	4
1.1	Introduction.....	4
1.2	Research .....	4
1.2.1	Breadth First Search.....	4
1.2.2	Depth First Search.....	6
1.2.3	A* Search – More Complex Algorithm .....	8
1.3	Analysis Data Dictionary.....	9
1.4	Object Oriented Planning.....	10
1.4.1	Class Diagrams .....	10
1.4.2	TGridItem Class .....	11
1.4.3	TSearch Class.....	11
1.5	Users and User Needs .....	11
1.6	Objectives of New System.....	12
1.6.1	System Start up .....	12
1.6.2	Create the Grid.....	12
1.6.3	Saving the Grid Locations.....	13
1.6.4	Running a Search .....	13
1.6.5	Reset .....	14
1.7	Research Methods used .....	14
2	Design Section.....	15
2.1	Overall System Design.....	15
2.1.1	Hierarchy Diagram .....	15
2.2	User Interface Design .....	16
2.2.1	Stage 1 – Form Create.....	16
2.2.2	Stage 2 - Create Grid .....	16
2.2.3	Change state of Grid Items .....	16
2.2.4	Save Grid .....	17
2.2.5	Specify Algorithm.....	18
2.2.6	Search.....	18
2.2.7	Reset .....	18
2.2.8	Data Dictionary for Form Controls.....	19
2.3	Classes .....	20
2.3.1	TGridItem Class .....	20
2.3.2	TSearch Class.....	22
2.3.3	ShortestPath Unit.....	27
3	System Testing .....	37
3.1	Typical Data Testing .....	37

3.2	Erroneous Data Testing .....	39
3.3	Extreme Data Testing .....	40
3.4	Boundary Testing .....	40
4	System Maintenance .....	41
4.1	Overview Guide .....	41
4.1.1	GridItemClass .....	41
4.1.2	SearchClass .....	41
4.1.3	Objects and User Interface .....	42
4.2	Commented Code .....	44
5	Evaluation .....	45
5.1	Original Objective Evaluation Based On Client Feedback .....	45
5.1.1	System Start up .....	45
5.1.2	Create the Grid.....	45
5.1.3	Saving the Grid Locations.....	46
5.1.4	Running a Search .....	46
5.1.5	Reset .....	47
5.2	Further Development.....	47
5.2.1	Current Problems.....	47
5.2.2	Improvements.....	47
6	Appendices.....	48
6.1	Appendix 1 – Initial Client Meeting.....	48
6.2	Appendix 2 - Further Meeting .....	49
6.3	Appendix 3 - Client Feedback.....	50
6.4	Appendix 4 - Commented Program Listings.....	51

# 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.

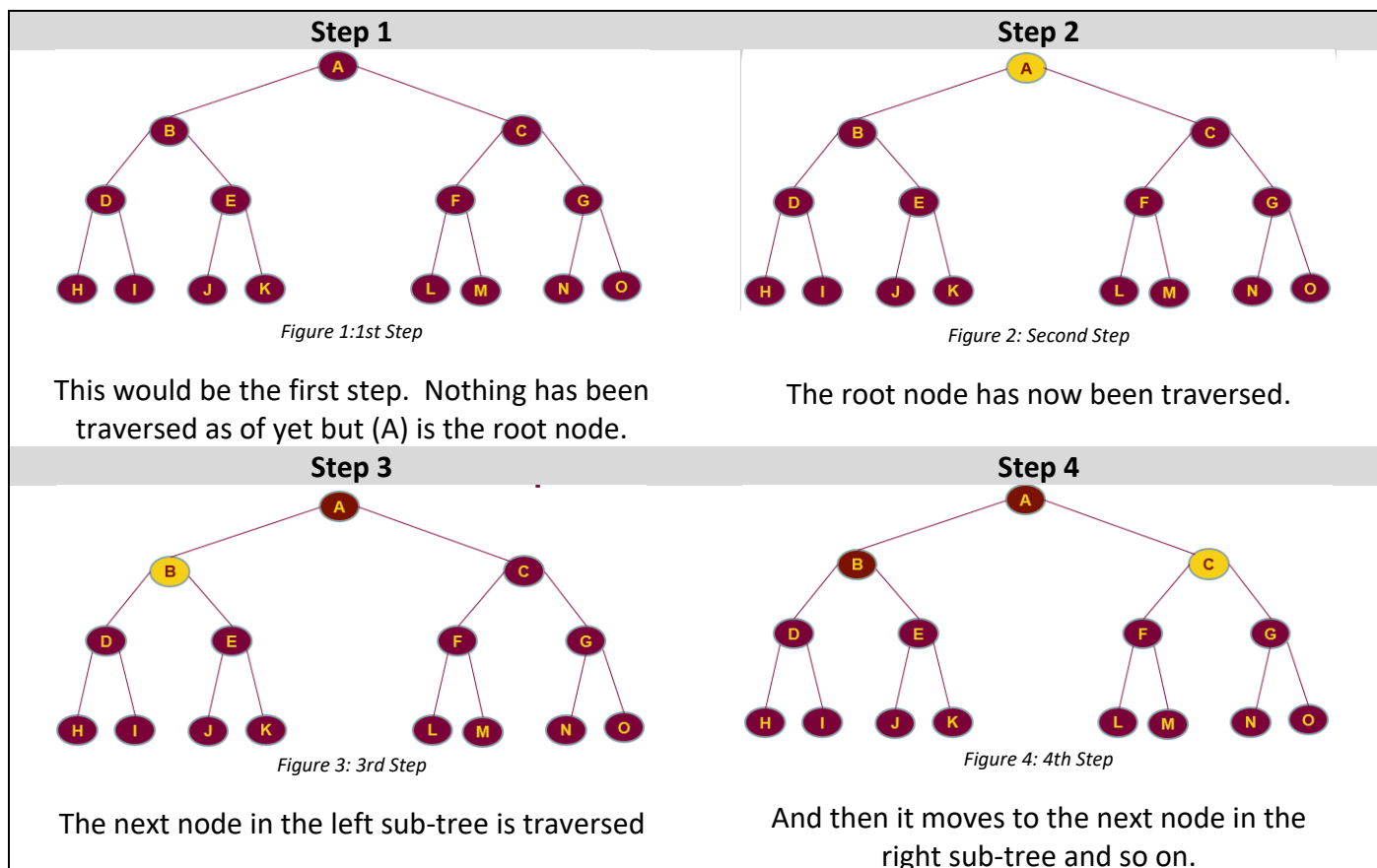
### 1.2.1 Breadth First Search

A Breadth first search<sup>1</sup> 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)	Explanation
<b>BEGIN</b> queue ← [] visited ← [] enqueue vertex <b>WHILE</b> queue NOT EMPTY DO dequeue item and put in currentNode set colour of currentNode to "dark blue" append currentNode to visited <b>FOR</b> each neighbour of currentNode DO <b>IF</b> colour of neighbour = "white" <b>THEN</b> enqueue neighbour set colour of neighbour to "pale blue"	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

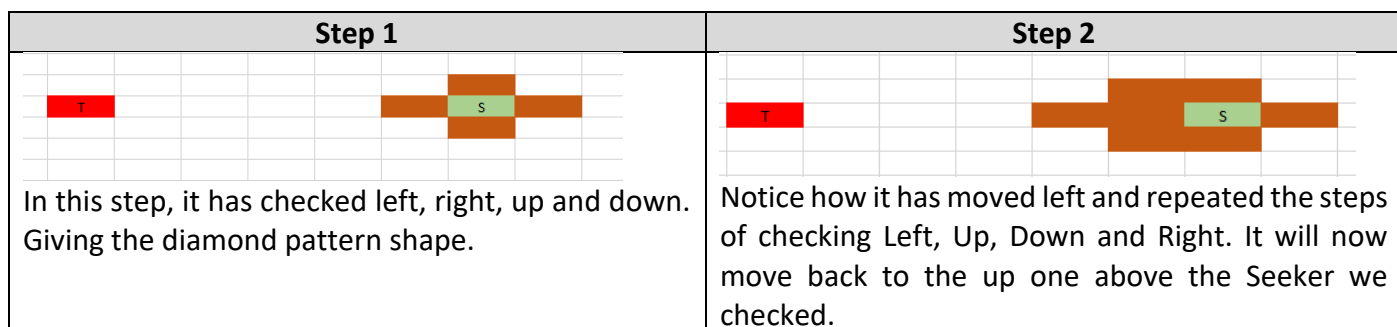
<b>END IF</b> <b>END FOR</b> <b>END WHILE</b> RETURN visited <b>END FUNCTION</b>	7. End all statements and after end of while statement return visited list.
--	---


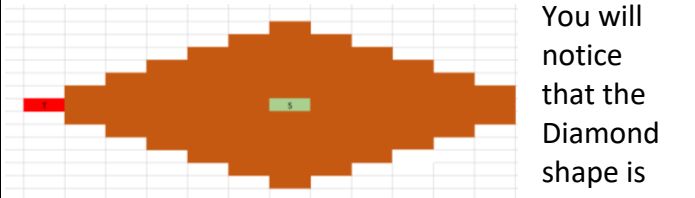
I have made these diagrams to demonstrate the first level search. Yellow is used as the node visited.



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 3	Step 4
 <p>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.</p>	 <p>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.</p>

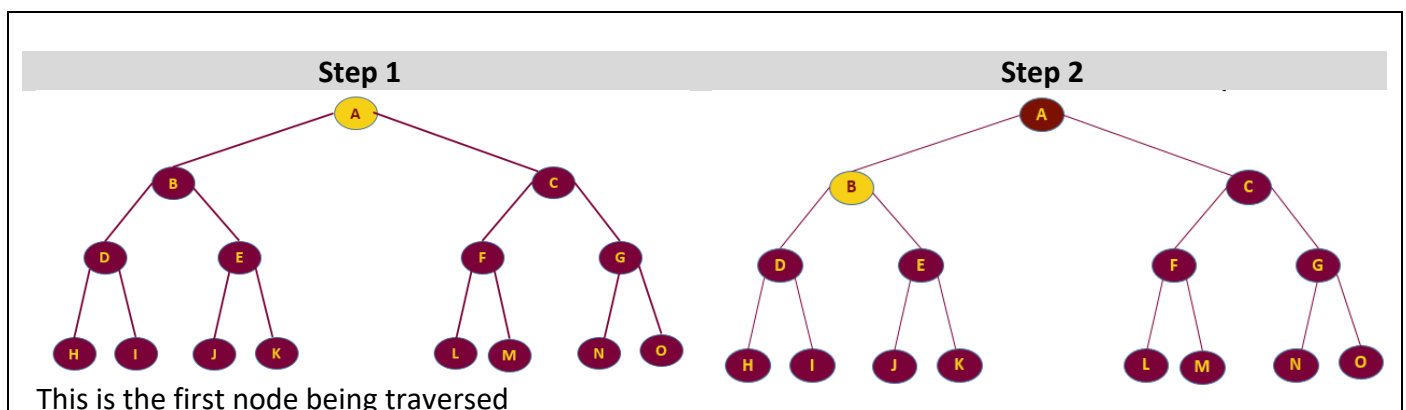
I think this search will be a good search to include in the program though the algorithm will need to be tailored in order to be suitable for the program. My program will need to resemble the spreadsheet screen prints I included as opposed to simply changing the colour of a vertex. Once the search is 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<sup>2</sup> works by starting at the Root node working down first then moving left to right. A standard algorithm for this is:

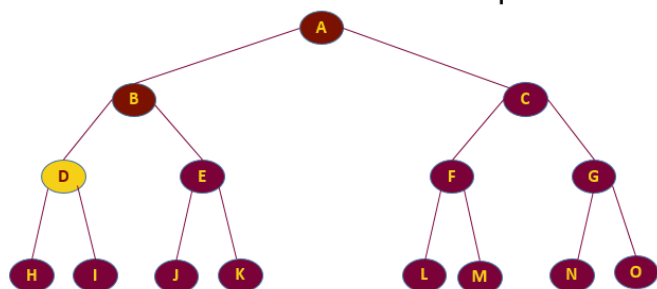
<pre> visitedList = [ ] <b>FUNCTION</b> dfs(graph, currentVertex, visited) Begin     <b>FOR</b> vertex IN graph[currentVertex] <b>DO</b>         <b>IF</b> vertex NOT IN visited <b>THEN</b>             dfs(graph, vertex, visited)         <b>END IF</b>     <b>END FOR</b>     Return visited <b>END FUNCTION</b> Traversal = dfs(GRAPH, "A", visitedList) OUTPUT "Nodes visited in this order: ", traversal </pre>	<p>Explanation</p> <ol style="list-style-type: none"> <li>1. an empty list of visited nodes (vertices) is created</li> <li>2. the Function dfs is called with parameters being passed of the graph, the current vertex and the visited list</li> <li>3. Each vertex in the graph is checked to see 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</li> <li>4. The nodes visited are then displayed in order.</li> </ol>
--	---

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



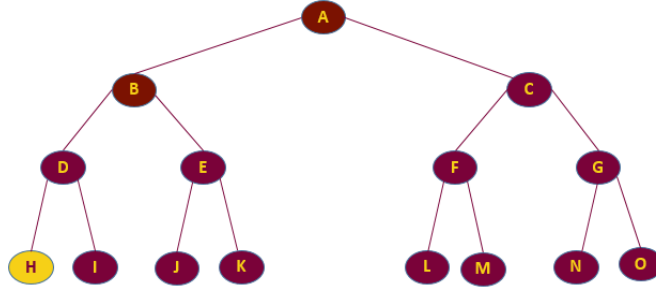
At step 2 it still looks like the same traversal as depth-first in that the next node in the left sub-tree is examined.

Step 3



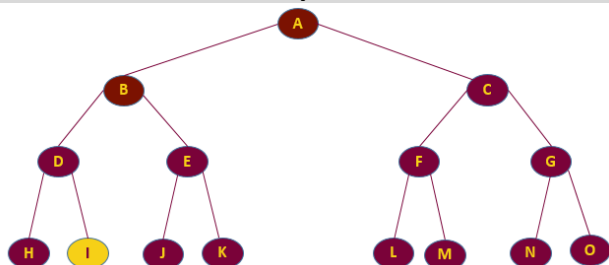
However, this is where it now differs; it continues to traverse the left sub-trees.

Step 4



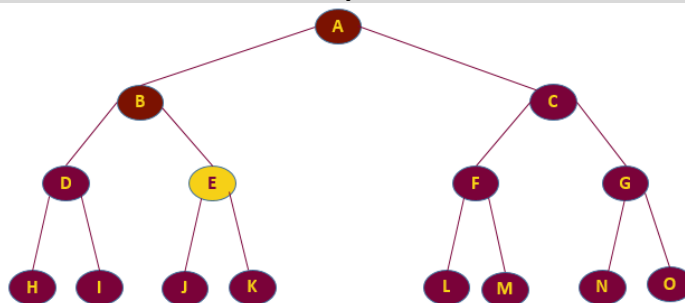
This is the final left sub-tree it can traverse from D

Step 5



So it now moves the right sub-tree from D

Step 6


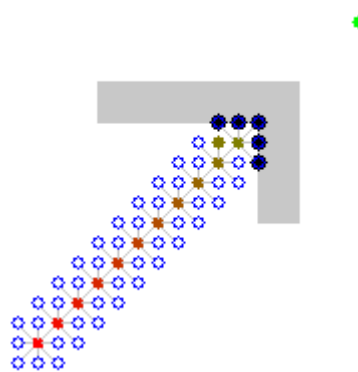
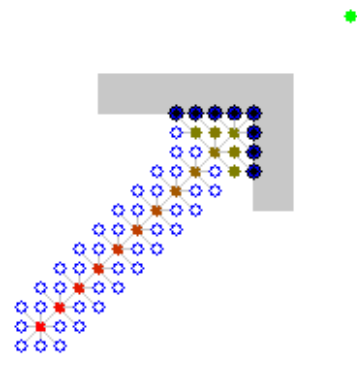
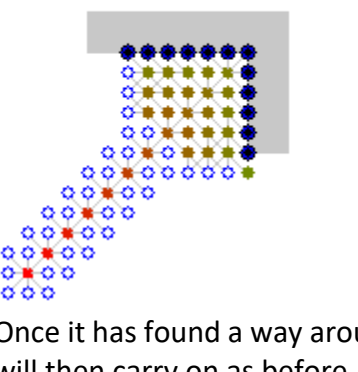
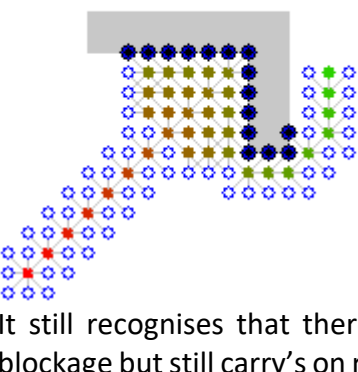
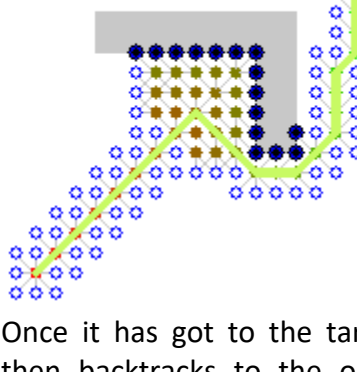


D has now been fully explored so it moves back to B. The left sub-tree of B has now been fully explored so it moves to the right sub-tree of B and so on.

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<sup>3</sup>. This is an example<sup>4</sup> of an A\* search being carried out.

Step 1	Step 2	Step 3
 <p>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.</p>	 <p>It then hits the wall so back tracks the nodes it has visited.</p>	 <p>It keeps doing this until it can get around the blockage.</p>
Step 4	Step 5	Step 6
 <p>Once it has found a way around it will then carry on as before.</p>	 <p>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.</p>	 <p>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.</p>

This is the pseudocodes

```

1: // A*
2: initialize the open list
3: initialize the closed list
- put the starting node on the open list (you can leave its f at zero)
4:
5: while the open list is not empty
6:     find the node with the least f on the open list, call it "q"
```

<sup>3</sup> <http://stackoverflow.com/questions/29470253/astar-explanation-of-name>

<sup>4</sup> [https://www.reddit.com/r/programming/comments/1cylmb/pathfinding\\_algorithm\\_visually\\_explained/](https://www.reddit.com/r/programming/comments/1cylmb/pathfinding_algorithm_visually_explained/)

<sup>5</sup> <http://web.mit.edu/eranki/www/tutorials/search/>



```

7:      pop q off the open list
8:      generate q's 8 successors and set their parents to q
9:      for each successor
10:         if successor is the goal, stop the search
11:         successor.g = q.g + distance between successor and q
12:         successor.h = distance from goal to successor
13:         - successor.f = successor.g + successor.h
14:         - if a node with the same position as successor is in the OPEN list \
15:           which has a lower f than successor, skip this successor
16:         - if a node with the same position as successor is in the CLOSED list \
17:           which has a lower f than successor, skip this successor
18:         otherwise, add the node to the open list
19:      end
20:      push q on the closed list
21:  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

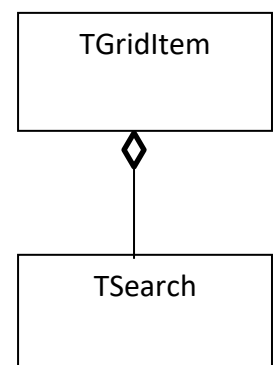
## 1.4 Object Oriented Planning

### 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

<b>TGridItem</b>
<b>Private</b> CurrentState : string Posx:Integer Posy:Integer leftConnection : TGridItem rightConnection : TGridItem topConnection : TGridItem bottomConnection : TGridItem parentNode : TGridItem visited : Boolean
<b>Public</b> 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 come 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.

<b>TSearch</b>
<b>Private</b>
<b>Public</b> 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 accidentally 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 certain 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  $c^2=a^2+b^2$
  - 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.

## 2 Design Section

### 2.1 Overall System Design

The system will first run with the main form created along with the variables and dictionary. The user will create the size of the grid. They will then have the opportunity to change the states of each grid nodes to either: Open, Closed, Target or Seeker.

- Open means the node is open and traversable.
- Close means the node is an obstacle and is not traversable
- Target is the node where the user wants to get
- Seeker is the node that will find the target

Once the user has completed these options, they will click a button to save the state of the nodes, which will trigger a process that build the connections for each node in the grid and saves it into the connections list. The user will not be able to change the grid after this point. From here, they will select which algorithm they want to run from the options given:

- A\*
- Depth first
- Breadth first

Once selected they will click a search button which will trigger the demonstration of the algorithm. This will involve changing the nodes visited to orange.

If the user would like to change the grid after they have selected the save button there will be able to through the click of a reset button. This will reset the program so the user can start again.

#### 2.1.1 Hierarchy Diagram

This diagram shows the main stages that will be required in the program.

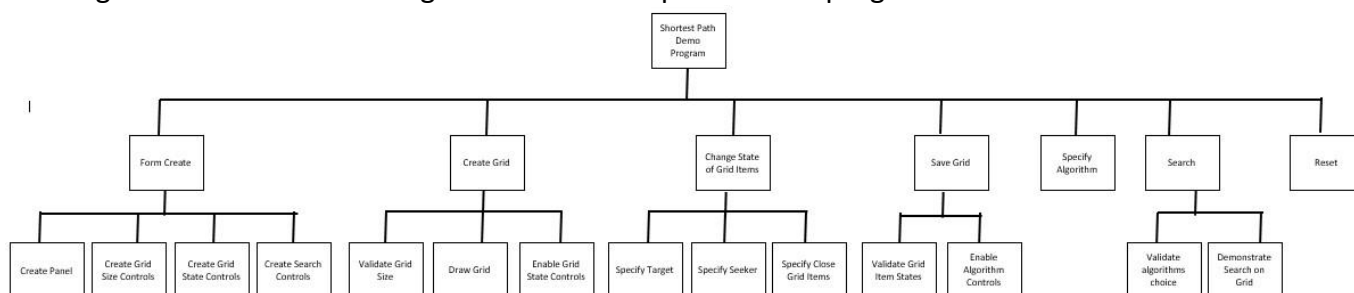


Figure 5: Hierarchy Diagram

## 2.2 User Interface Design

This is the user interface design in the stages in the order they would occur.

### 2.2.1 Stage 1 – Form Create

This is the first stage and it will occur as the form is created. It will hold a panel where the grid will be drawn and all of the controls the user needs in order to specify the grid they would like, the types of node, which algorithm they would like to run and a reset option. The only controls enabled at this point will be the panel where the grid will be drawn, the grid size edit boxes and the create button.

### 2.2.2 Stage 2 - Create Grid

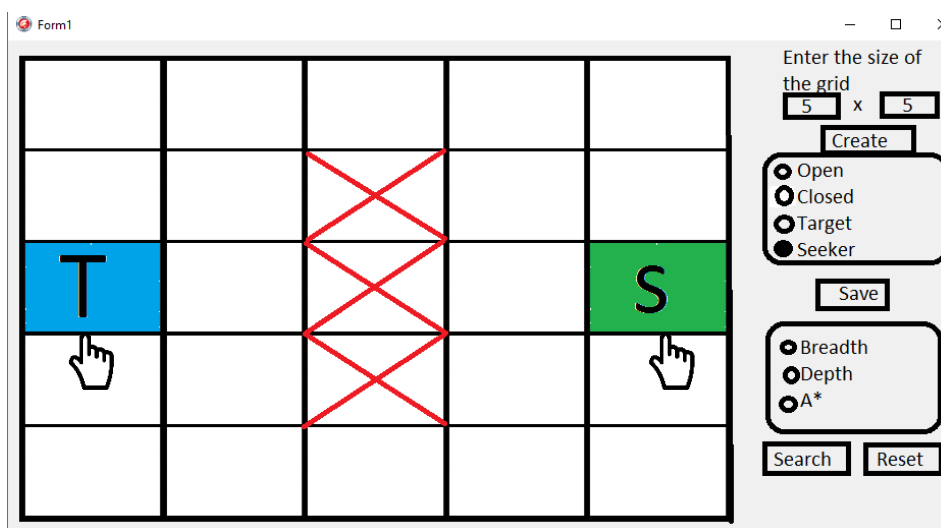
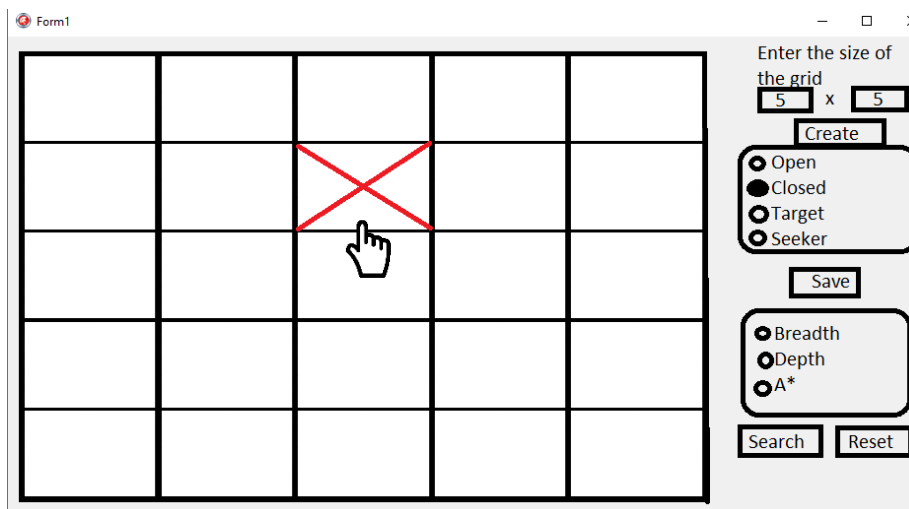
This is the stage where the user will specify the size of the grid. When create is clicked validation will be present to ensure both edit boxes have values between 1 and 10 as the maximum grid size is 10 x 10. If the size is valid, the grid will be created and the grid item type radio group and save button will become enabled.

### 2.2.3 Change state of Grid Items

Once the grid has been drawn the user will then specify special nodes.

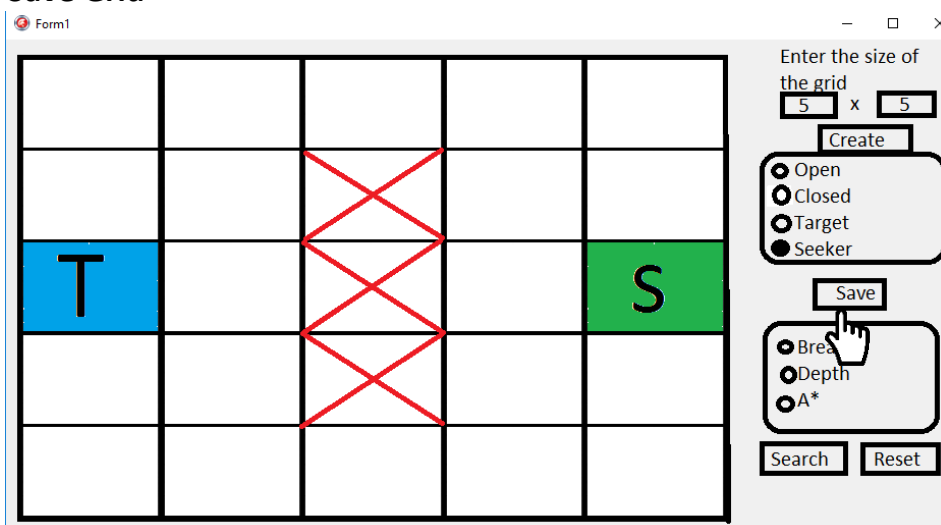
In this example, a closed grid item will be specified. The user will click closed and the grid item selected will change to a red cross to show that it is closed and not traversable. This can be seen below:





The user will specify a target and seeker too. These will be shown as a T and S, blue and green respectively as shown.

#### 2.2.4 Save Grid

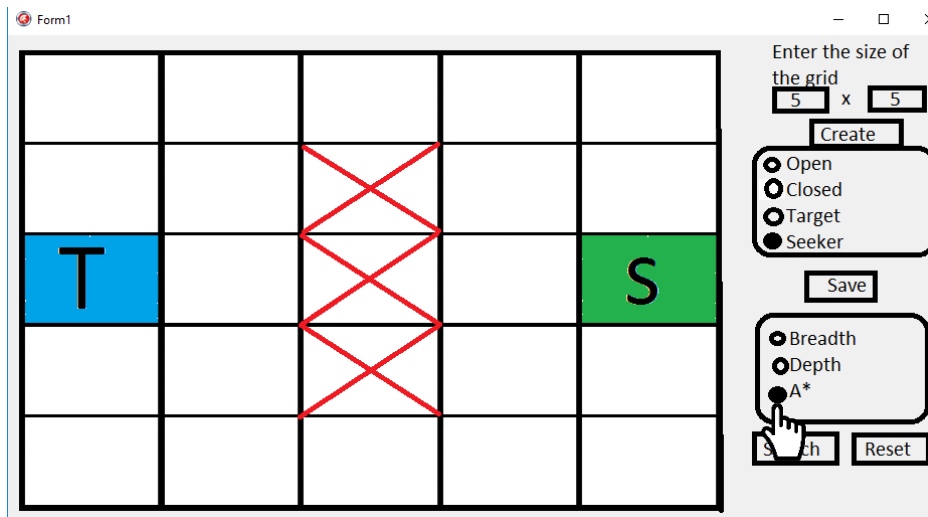


Once the Target and Seeker have been placed the user will click on the save button. Validation will be applied to ensure a target and seeker have been specified.

The user will not see anything happen at this point as all processing will be background processing. All of the node connections will be

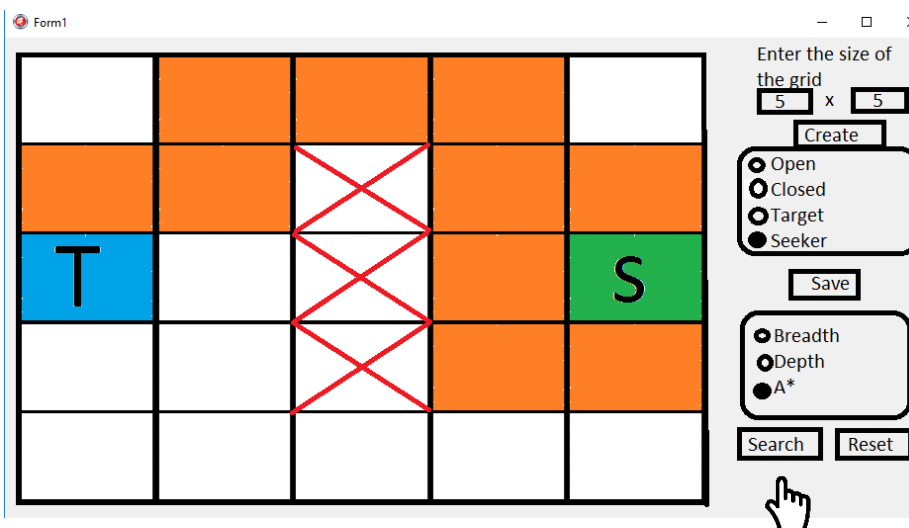
determined and stored and from this point, changing the grid will be prevented. The algorithm type radio group, search and reset button will become enabled at this point.

## 2.2.5 Specify Algorithm



The user can then select which search algorithm they wish to run. In this example the A\* algorithm has been selected.

## 2.2.6 Search



The user can then select the save button. Validation will ensure an algorithm has been specified. The traversal will then be carried out and shown by changing the grid items visited to orange as in this example.

## 2.2.7 Reset



If the user would like to change the grid e.g. the size of the grid or the setup of each grid items, then they will have the ability to click on the reset button which will completely reset the program back to how it was in the beginning..

### 2.2.8 Data Dictionary for Form Controls

These are the controls that I think will be required for the form

Control Name	Control Type	Description
frmShortestPath	Form	This will be the form itself.
pnlGrid	Panel	This will hold the grid. This will be enabled.
edtWidth	Edit box	This will be used to get the width of the grid, it will be used to determine how many columns are required. This will be enabled.
edtHeight	Edit box	This will be used to get the height of the grid. It will be used to determine how many rows are required. This will be enabled.
lblGridSize	Label	This will be used to explain the width and height edit boxes so the user knows what they need to input.
btnGrid	Button	This will be enabled. When the user has specified the grid size they will click this and this will enable other controls as specified below.
rgrpObject	Radio group	This will be enabled after the user has clicked btnGrid. Options will be 'Open', 'Closed', 'Target', 'Seeker'. Only one option at a time will be able to be selected and it will be used to specify what each grid item will be. There can only be one target and one seeker.
lblValidation	Label	This will be used to let the user know that only 1 Seeker and 1 Target can be placed.
btnSave	Button	This will be for the user to press once they have setup the grid i.e placed the target and seeker and whatever closed items they want to place. The button saves the Grid by building the connections between the different Grid items. It will also disable the ability to change the grid.
rgrpSearch	Radio Group	This will be for the user to decide which search algorithm they want to run. The options will be 'Breadth First Search', 'Depth First Search' and 'A* Search'. There will be one Item already selected so it does not cause any issues further on.
btnSearch	Button	This will be to carry out the search algorithm the user has selected in the previous radio group. It will then update the grid by changing the states of the different grid item and the visual of each node.
BtnReset	Button	This will completely reset the program to its original state when it was booted up. This could occur for example if the user set up the grid, clicked on the save button and would like to backtrack and change the grid.

## 2.3 Classes

There will be two classes:

### 2.3.1 TGridItem Class

This class is needed to manage the grid

Class Name		TGridItem class of TImage	
Private	Name	Data Type	Comments
	CurrentState	string	This will be used to store whether the Grid Item is either: 'Open', 'Closed', 'Visited', 'Target' and 'Seeker'.
	Posx	integer	This will be used to store which column the Grid Item is.
	Posy r	Integer	This will be used to store which row the Grid Item is.
	distance	Integer	This will be used in connection to the A* Search Algorithm as it will store the distance that the Grid Item will be once visited.
	leftConnection	TGridItem	This will be to store what is left to the current grid item.
	rightConnection	TGridItem	This will be to store what is right to the current grid item.
	TopConnection	TGridItem	This will be to store what is top to the current grid item.
	bottomConnection	TGridItem	This will be to store what is bottom to the current grid item.
	parentNode	TGridItem	This will be to show which node it had previously been at to get to the current Grid Item.
	visited	Boolean	This will be for the depth first search algorithm where the current Grid item will be classed as either True for visited or False for not visited.
Public	Method	Comments	
	Function getCurrentState : String	This will be for when the program needs to find out what the state of the current Grid Item.	
	Function getPosx :Integer	This will be used to work out the position of the current Grid Item.	
	Function getPosy :Integer	This will be used to work out the position of the current Grid Item.	
	Function getLeftConnection : TGridItem	This will be for when the algorithm is wanting to move to the node to the left of the current grid item or just to check if there is a node there.	
	Function getRightConnection : TGrid	This will be for when the algorithm is wanting to move to the node to the right of the current grid item or just to check if there is a node there.	

Function getTopConnection : TGri	This will be for when the algorithm is wanting to move to the node to the top of the current grid item or just to check if there is a node there.
Function getBottomConnection : TGridItem	This will be for when the algorithm is wanting to move to the node to the bottom of the current grid item or just to check if there is a node there.
Function getParentNode :	This will be for when the algorithm needs to work out where the node has come from in order.
Function getVisited : boolean	This will be for when the algorithms need to check if it has already been visited or not.
Function getDistance:integer	This will be specifically for the A* Search algorithm to check the distance so far from that current Grid Item.
Procedure setCurrentState(pCurrentState : st	This will be used to set the current state of the Grid Item.
Procedure setPosx (pPosx :Integer)	This will be used to set the column in which the Grid Item is positioned.
Procedure setPosy (pPosy :Integer)	This will be used to set the row in which the Grid Item is positioned.
Procedure setDistance(pDistance:Integer)	This will be for when the A* algorithm sets the distance so far to the Grid Item.
Procedure setLeftConnection(pLeftConnection : TGridItem)	This will be used to set what node is to the left of the current Grid Item.
Procedure setRightConnection(pRightConnection : TGridItem)	This will be used to set what node is to the right of the current Grid Item.
Procedure setTopConnection(pTopConnection : TGridItem)	This will be used to set what node is to the top of the current Grid Item.
Procedure setBottomConnection(pBottomConnection : TGridItem)	This will be used to set what node is to the bottom of the current Grid Item.
Procedure setParentNode(pParentNode : TGridItem)	This will be to set where the current Grid Item has come from.
Procedure setVisited(pVisited : boolean)	This will be specifically for the depth first search to set the current Grid Item as visited or not.

No algorithms are provided for the methods as they are purely setters and getters.

### 2.3.2 TSearch Class

This will use composition aggregation with TGridItem Class. TGridItem objects will be instantiated in this class.

Class Name		TSearch class of TGridItem
Public	Method	Comments
	Procedure runBreadthFirstSearch(rootNode: TGridItem)	This will be when 'Breadth First Search' algorithm is to be run and will also mark on the grid the visited nodes.
	procedure runDepthFirstSearch(rootNode : TGridItem)	This will be when 'Depth First Search' algorithm is to be run and will also mark on the grid the visited nodes.
	procedure runAStarSearch(rootNode, targetNode : TGridItem)	This will be when 'A* Search' algorithm is to be run and will also mark on the grid the visited nodes.

#### 2.3.2.1 Procedure runBreadthFirstSearch(rootNode:TGridItem)

This will be used to run a breadth first search if the user has selected this option.

##### Local variables required:

queue: TQueue<TGridItem>	This will be used as the data structure for the search
itemFound : Boolean	This will be used to signify when the target has been found
currentNode : TGridItem	This will be used to hold the node being examined

It will run like this:

- The queue will be created
- The seeker will be enqueued
- ItemFound will be set to false as this will be used to signify when the target has been found
- While there are still items on the queue and the target has not been found an item from the queue will be dequeued and stored in currentNode. A call to the class method getCurrentState will be used to check whether this is the target. If it is itemFound will be set to true and the while loop will be exited.
- If it is not the target then it will make sure that it is not a closed, seeker or empty grid item then mark it as visited whilst also changing the image on the grid to mark as visited.
- It will then repeat this by then looking at right, top and bottom connection next.
- The Procedure will end once the Target has been found.

##### Pseudocode

**BEGIN**

queue.create

Enqueue the root node

Set itemFound to false

**WHILE** queue size > 0 and itemFound=False **DO**

currentNode = dequeued node

**IF** currentNode.getCurrentState = Target **THEN**

ItemFound=True

**ELSE**

**IF** (currentNode.getLeftConnection <> nil) And  
(currentNode.getCurrentState <> Closed) **THEN**  
queue.Enqueue(currentNode.getLeftConnection)  
currentNode.setLeftConnection(Nil)

```

        IF currentNode.getCurrentState <> Seeker THEN
            currentNode.setCurrentState(Visited)
            currentNode.Picture = visited picture
        END IF
    END IF

    IF (currentNode.getRightConnection <> nil) And
    (currentNode.getCurrentState <> Closed) THEN
        queue.Enqueue(currentNode.getRightConnection)
        currentNode.setRightConnection(Nil)
        IF currentNode.getCurrentState <> Seeker THEN
            currentNode.setCurrentState(Visited)
            currentNode.Picture = visited picture
        END IF
    END IF

    IF (currentNode.getTopConnection <> nil) And
    (currentNode.getCurrentState <> Closed) THEN
        queue.Enqueue(currentNode.getTopConnection)
        currentNode.setTopConnection(Nil)
        IF currentNode.getCurrentState <> Seeker THEN
            currentNode.setCurrentState(Visited)
            currentNode.Picture = visited picture
        END IF
    END IF

    IF (currentNode.getBottomConnection <> nil) And
    (currentNode.getCurrentState <> Closed) THEN
        queue.Enqueue(currentNode.getBottomConnection)
        currentNode.setBottomConnection(Nil)
        IF currentNode.getCurrentState <> Seeker THEN
            currentNode.setCurrentState(Visited)
            currentNode.Picture = visited picture
        END IF
    END IF
END IF
END WHILE
END PROCEDURE

```

### 2.3.2.2 Procedure runDepthFirstSearch(rootNode : TGridItem)

This will be used to run a depth first search if the user has selected this option.

#### Local variables required:

stack : TStack <TGridItem>  
 itemFound : boolean

currentNode : TGridItem

rightConnection: TGridItem

leftConnection : TGridItem

topConnection : TGridItem

This will be used for the data structure of the search.

This will be used to state if the Target has been found or not.

This will be used as the current node that the algorithm is currently using.

This will be used to store whatever is to the right of the current node.

This will be used to store whatever is to the left of the current node.

This will be used to store whatever is to the top of the current node.

`bottomConnection : TGridItem` This will be used to store whatever is to the bottom of the current node.

It will run like this:

- The stack will be created.
- The Root node will be added onto the stack.
- Whilst there is still something still in the stack and the item found hasn't been found, do:
  - Current Node becomes the grid item popped off the stack.
  - If the current node hasn't been visited and it is not classed as a 'closed' node then.
  - The current node becomes visited.
  - If current node isn't the seeker and not the target then.
  - Set the current nodes image to a visited image.
  - If the current node is the target then set item found as true.
  - Set top bottom left and right connections to the connections in the current node.
  - If each connection is not visited and exists then add it to the stack.
- End Procedure.

### Pseudocode

#### BEGIN

Stack.Create

Stack + RootNode

**WHILE** stack.count > 0 and itemfound = false **DO**

#### BEGIN

currentNode = popped node in stack

**IF** currentNode.getVisited = false and CurrentNode.getCurrentState <> Closed **THEN**

#### BEGIN

currentNode.setVisited(true)

**IF** CurrentNode.getCurrentState <> Seeker and CurrentNode.getCurrentState <> Target **THEN**

CurrentNode.Picture.Visited.jpg

**IF** currentNode.getCurrentState = 'Target' **THEN**

ItemFound = true

#### ELSE

#### BEGIN

rightConnection = currentNode.getRightConnection

leftConnection = currentNode.getLeftConnection

topConnection = currentNode.getTopConnection

bottomConnection = currentNode.getBottomConnection

**IF** rightConnection <> nil and rightConnection.getVisited = false **THEN**

Stack + currentNode.getRightConnection

**IF** leftConnection <> nil and leftConnection.getVisited = false **THEN**

Stack + currentNode.getLeftConnection

**IF** topConnection <> nil and topConnection.getVisited = false **THEN**

Stack + currentNode.getTopConnection

**IF** bottomConnection <> nil and bottomConnection.getVisited = false **THEN**

Stack + currentNode.getBottomConnection

**END ELSE**

**END IF**

**END WHILE**

**END PROCEDURE**



### 2.3.2.3 Procedure TSearch.runAStarSearch(rootNode, targetNode:TGridItem)

This will be used to run an A\* search if the user has selected this option.

#### Local variables required:

Found : boolean	This will be used to store if the Target has been found or not.
movementOptions : TList<TGridItem>	This will be used as a list to store different data at the same time.
I : integer	This will be used as a counter.
a,b,c : integer	This will be used to calculate the distance whilst storing different parts of the algorithm.

It will run like this:

- The movement options list will be created.
- The root node will be added to the list.
- Repeat...
  - If the first item in the lists left connection is not nil, not closed and not visited then add the first item in the lifts left connection to the list.
  - Do this for top bottom and right connection too.
  - If the first item in the lists state is the target then set found as true.
  - Else if it is not the seeker then set current image and set state to visited.
  - Delete the first item in the list and trim excess so everything moves down by 1.
  - For I = 0 to the number of items in the list -1 do.
  - $A = \text{pos } x \text{ of the target node} - \text{pos } x \text{ of the } I \text{ position in the list}$
  - $B = \text{pos } y \text{ of the target node} - \text{pos } y \text{ of the } I \text{ position in the list}$
  - $A = A * A$
  - $B = B * B$
  - $C = \text{Rounding the result of the square root of } A + B$ .
  - Set distance of I position in the list as C.
  - Sort the Items in the list in order of their distance.

#### Pseudocode

##### BEGIN

MovementOptions = TList.create

movementOptions + rootNode

##### REPEAT

```

IF movementOptions[0].getLeftConnection <> nil and
movementOptions[0].getLeftConnection.getCurrentState <> Closed and
movementOptions[0].getLeftConnection.getCurrentState <> Visited THEN
    MovementOptions + movementOptions[0].getLeftConnection
IF movementOptions[0].getRightConnection <> nil and
movementOptions[0].getRightConnection.getCurrentState <> Closed and
movementOptions[0].getRightConnection.getCurrentState <> Visited THEN
    movementOptions + movementOptions[0].getRightConnection
IF movementOptions[0].getTopConnection <> nil and
movementOptions[0].getTopConnection.getCurrentState <> Closed and
movementOptions[0].getTopConnection.getCurrentState <> Visited THEN
    movementOptions + movementOptions[0].getTopConnection
  
```

```

IF movementOptions[0].getBottomConnection <> nil and
movementOptions[0].getBottomConnection.getCurrentState <> Closed and
movementOptions[0].getBottomConnection.getCurrentState <> Visited THEN
    movementOptions + movementOptions[0].getBottomConnection

IF movementOptions[0].getCurrentState = Target THEN
    Found = True
ELSE
BEGIN

    IF movementOptions[0].getCurrentState <> Seeker THEN
        BEGIN
            movementOptions[0].Picture.Visited.jpg
            movementOptions[0].setCurrentState(Visited)
        END IF
        movementOptions.Delete(0)
        movementOptions Trim Excess
        FOR i = 0 to movementOptions.Count -1 DO
            BEGIN
                a= targetNode.getPosx-movementOptions[i].getPosx
                b= targetNode.getPosy-movementOptions[i].getPosy
                a=a*a
                b=b*b
                c=round(SquareRoot(a+b))
                movementOptions[i].setDistance(c)
            END IF
            sortList(movementOptions)
        END ELSE
    UNTIL found = true
END PROCEDURE

```

### 2.3.3 ShortestPath Unit

Form Name		TfrmShortestPath class of TForm	
Private	Name	Data Type	Comments
	Item	TGridItem	This will be used to store the data of a grid.
	onClickEvent	TNotifyEvent	This will be for when an Grid Item has been clicked it will be used to store what happens to it.
	GridItem	TGridItem	This will be used to store the data of a second GridItem.
	gridItemDictionary	TDictionary<string, TGridItem>	This will be a dictionary to store the connections and link them together.
	connectionsFrom	TList<string>	This will be for building the connections as to where the connection has come from.
	connectionsTo	TList<string>	This will be for building the connections as to where the connection is going.
	XSize	Integer	The Position of the grid item on the X axis.
	YSize	Integer	The Position of the grid item on the Y axis.
	Seeker	Boolean	This will be to set if the Seeker has been placed or not.
	Target	Boolean	This will be to set if the Target has been placed or not.
	Stop Change	Boolean	This is will be to disable the user from changing the grid afterwards by setting it to true.
Public	Method		Comments
	Procedure FindAllConnections(var connections:array of string;Key:String)		This will be to find the connections and put them into an appropriate array.
	function findSeeker() : TGridItem;		This will be to check if the Seeker has already been placed or not.
	function findTarget() : TGridItem;		This will be to check if the Target has already been placed or not.

#### 2.3.3.1 Procedure TfrmShortestPath.btnResetClick (Sender : TObject)

This will be use to reset the form to its original state before the grid was initially created.

It will run like this:

- Destroy the current Grid.
- Create the lists and dictionaries.
- Disable all objects and validations on the form that should not be accessible to the user until the appropriate time has come.
- Enable the ability for the user to create a new grid.

#### Pseudocode

##### BEGIN

```

pnlGrid.Destroy
gridItemDictionary = TDictionary<string, TGridItem>.Create
connectionsFrom = TList<string>.Create
connectionsTo = TList<string>.Create

```

```

edtwidth.Enabled=False
edtHeight.Enabled=False
btnGrid.Enabled=False
rgrpObject.Enabled=False
btnSaveLocations.Enabled=False
rgrpSearchType.Enabled=False
btnSearch.Enabled=False
StopChange= False
lblValidation.Enabled=False
Seeker=False
Target=False

```

```

edtwidth.Enabled=True
edtheight.Enabled=True
btnGrid.Enabled=True

```

**END PROCEDURE**

### 2.3.3.2 Procedure TfrmShortestPath.FindAllConnections(var connections:array of string;Key:String)

This will be to store all the connections from and to each GridItem.

**Local variables required:**

I : Integer                This will be used as a counter

Count : Integer        This will be used to find a position in an array.

It will run like this:

- Count will become equal to "0".
- For each connections from it will then do the opposite and place it into the connections to.

**Pseudocode**

**BEGIN**

count = 0

**FOR** I = 0 to connectionsFrom.Count -1 **do**

**BEGIN**

**IF** connectionsFrom.Items[I] = Key **THEN**

**BEGIN**

connections[count] = connectionsTo.Items[I]

count = count +1

**END**

**END**

**END PROCEDURE**

### 2.3.3.3 Procedure TfrmShortestPath.btnSaveLocationsClick(Sender : TObject)

This will be to build up the connections and allow the user to then run the search.

It will run like this:

- If the seeker or target hasn't been placed then remind the user that at least 1 target and 1 seeker must be placed.
- Else: Disable the previous objects on the form so the user can't edit it and enable the ability to run the searches.
- Run the Procedure to build all connections between the grid items.

#### Pseudocode

##### BEGIN

IF (Seeker <> True) or (Target <> True) THEN  
    ShowMessage('You need to place at least 1 Target and 1 Seeker')

##### ELSE

##### BEGIN

    rgrpObject.Enabled = False  
    btnSaveLocations.Enabled = False  
    rgrpSearchType.Enabled = True  
    btnSearch.Enabled = True  
    StopChange = True  
    EstablishConnections

##### END

### 2.3.3.4 Procedure TfrmShortestPath.EstablishConnections

This will be to work out the connections between which grid items are connected.

#### Local variables required:

I : Integer	This will be used as a count.
J : Integer	This will be used as a count.
currentItem : TGridItem	This will be used to determine which node it is currently using.
neighbourItem : TGridItem	This will be used to store what the Item next to the Current Item is.
connectionsTemp : array [0..3] of string	This will be a temporary array for the connections.

It will run like this:

- For every Item in the Grid dictionary see if the CurrentItem is equal to dictionary position.
- See if it has a neighbour
- See if the neighbour is closed.
- If its not closed then add the connectionsFrom the connectionsTo the parentNode and the RightConnection
- Repeat for LeftConnections, TopConnections and BottomConnections.
- If the Current item is not in the dictionary then there is a serious error and the program should be restarted.

**Pseudocode****BEGIN****FOR** J = 0 to YSize -1 **DO****FOR** I = 0 to XSize -1 **DO****BEGIN****IF** gridItemDictionary.TryGetValue(inttostr(i)+inttostr(j), currentItem) **THEN**  
**BEGIN****IF** gridItemDictionary.TryGetValue(inttostr(i + 1)+inttostr(j), neighbourItem) **THEN****BEGIN****IF** neighbourItem.getCurrentState <> Closed **THEN****BEGIN**connectionsFrom.Add(inttostr(i)+inttostr(j))  
connectionsTo.Add(inttostr(i + 1)+inttostr(j))  
neighbourItem.setParentNode(currentItem)  
currentItem.setRightConnection(neighbourItem)**END****END****IF** gridItemDictionary.TryGetValue(inttostr(i - 1) + inttostr(j), neighbourItem) **THEN****BEGIN****IF** neighbourItem.getCurrentState <> Closed **THEN****BEGIN**connectionsFrom.Add(inttostr(i)+inttostr(j))  
connectionsTo.Add(inttostr(i-1)+inttostr(j))  
neighbourItem.setParentNode(currentItem)  
currentItem.setLeftConnection(neighbourItem)**END****END****IF** gridItemDictionary.TryGetValue(inttostr(i)+inttostr(j + 1), neighbourItem) **THEN****BEGIN****IF** neighbourItem.getCurrentState <> Closed **THEN****BEGIN**connectionsFrom.Add(inttostr(i)+inttostr(j))  
connectionsTo.Add(inttostr(i)+inttostr(j+1))  
neighbourItem.setParentNode(currentItem)  
currentItem.setBottomConnection(neighbourItem)**END****END****IF** gridItemDictionary.TryGetValue(inttostr(i)+inttostr(j - 1), neighbourItem) **THEN****BEGIN****IF** neighbourItem.getCurrentState <> Closed **THEN****BEGIN**connectionsFrom.Add(inttostr(i)+inttostr(j));  
connectionsTo.Add(inttostr(i)+inttostr(j-1))  
neighbourItem.setParentNode(currentItem)  
currentItem.setTopConnection(neighbourItem)**END**

```

                END
            END
        ELSE
            ShowMessage(Grid Item Does not exist)
        END
    END
END PROCEDURE

```

### 2.3.3.5 Function TfrmShortestPath.findSeeker() : TGridItem

This will be to find where on the grid the seeker has been placed.

#### Local variables required:

I : Integer                      This will be used as a count  
 J : Integer                      This will be used as a count  
 currentNode : TGridItem      This will be used to store the current node it is at.

It will run like this:

- it will go through all the columns then the rows to look at each item.
- If the state is a seeker then
- Set findSeeker equal to the current node it is at so then it knows where the seeker.

#### Pseudocode

```

BEGIN
    FOR J = 0 to YSize -1 DO
        FOR I = 0 to XSize -1 DO
            BEGIN
                gridItemDictionary.TryGetValue(inttostr(i)+inttostr(j), currentNode)
                IF currentNode.getCurrentState = Seeker THEN
                    findSeeker = currentNode
                END
            END
        END
    END
END

```

### 2.3.3.6 Function TfrmShortestPath.findTarget() : TGridItem

This will be to find where on the grid the Target has been placed.

#### Local variables required:

I : Integer                      This will be used as a count  
 J : Integer                      This will be used as a count  
 currentNode : TGridItem      This will be used to store the current node it is at.

It will run like this:

- it will go through all the columns then the rows to look at each item.
- If the state is a target then
- Set findTarget equal to the current node it is at so then it knows where the Target.

#### Pseudocode

```

BEGIN
    FOR J = 0 to YSize -1 DO
        FOR I = 0 to XSize -1 DO
            BEGIN
                gridItemDictionary.TryGetValue(inttostr(i)+inttostr(j), currentNode)
                IF currentNode.getCurrentState = Target THEN

```

findTarget = currentNode

**END**

**END**

### 2.3.3.7 Procedure TfrmShortestPath.btnSearchClick(Sender : TObject)

This is for when the user wants to run the search algorithm and will be linked to the Search radio group to work out which search they want to run.

#### Local variables required:

searcher : TSearch	This will be the search class.
rootNode : TGridItem	This will be where the seeker is.
targetNode : TGridItem	This will be where the target node is.
I : Integer	This will be used as a count.
J : Integer	This will be used as a count.
currentNode : TGridItem	This will be used to look at the current node.

It will run like this:

- It will go through every node and if it has been classed as visited then
- Set it to 'Open'.
- RootNode becomes equal to findSeeker
- TargetNode becomes equal to findTarget
- Create the Searcher class.
- Depending on which item has been selected on the radio group it will then run the search in a separate class.

#### Pseudocode

begin

EstablishConnections

**FOR** J := 0 to YSize -1 **DO**

**FOR** I := 0 to XSize -1 **DO**

**BEGIN**

gridItemDictionary.TryGetValue(inttostr(i)+inttostr(j), currentNode)

currentNode.setVisited(false)

**IF** currentNode.getCurrentState = Visited **THEN**

**BEGIN**

CurrentNode.Picture.LoadFromFile(Open.jpg)

CurrentNode.setCurrentState(Open)

**END**

**END**

rootNode = findSeeker

targetNode = findTarget

searcher = TSearch.Create

**CASE** rgrpSearchType.ItemIndex **OF**

0: searcher.runBreadthFirstSearch(rootNode)

1: searcher.runDepthFirstSearch(rootNode)

2: searcher.runAStarSearch(rootNode, targetNode)

**END**

**END**



**2.3.3.8 procedure TfrmShortestPath.FormCreate(Sender : TObject)**

This will be for when the form is initially created and will disable most objects so the user cannot use them until needed.

It will run like this:

- Create GridItemDictionary.
- Create both ConnectionsFrom and ConnectionsTo Lists.
- Set Seeker, Target, StopChange and lblValidation to False.

**Pseudocode****BEGIN**

```
gridItemDictionary = TDictionary<string, TGridItem>.Create
connectionsFrom = TList<string>.Create
connectionsTo = TList<string>.Create
Seeker=False
Target=False
StopChange=False
lblValidation.Enabled=False
```

**END****2.3.3.9 Procedure TfrmShortestPath.ObjectChanger(sender : TObject)**

This will be for when the user clicks on the grid to change a grid item.

It will run like this:

- If stopChange is true then the user will get an error message stating they will have to press reset to change the grid.
- Depending on the item picked on the radio group for the target and seeker it will check if one has already been placed and if so an error message will pop up stating that only 1 Target and 1 Seeker can be placed.
- For open it will see if it is a target or seeker and if so it will then set the appropriate Booleans to false to state that there is no longer a target or seeker on the grid.
- For closed it will do the same as the open except it will change its state to closed.

**Pseudocode****BEGIN**

```
IF StopChange = True THEN
    ShowMessage(Please press Reset if you are wanting to change the grid)
```

**ELSE****BEGIN**

```
CASE rgrpObject.ItemIndex OF
```

**0:BEGIN**

```
    IF (sender as TGridItem).GetCurrentState = Target THEN
```

```
        Target= False
```

```
    IF (sender as TGridItem).GetCurrentState = Seeker THEN
```

```
        Seeker= False
```

```
    (sender as TGridItem).picture.LoadFromFile(Open.jpg)
```

```
    (sender as TGridItem).setCurrentState(Open)
```

**END****1:BEGIN**

```
    IF Target <> True THEN
```

```

        BEGIN
            (sender as TGridItem).picture.LoadFromFile(Target.jpg)
            (sender as TGridItem).setCurrentState(Target)
            IF (Seeker = True) THEN
                BEGIN
                    btnSaveLocations.Enabled=True
                    lblValidation.Enabled=False
                END
            END
            ELSE
                ShowMessage(Only 1 Seeker and 1 Target can be placed)
            END
        2:BEGIN
            IF Seeker <> True THEN
                BEGIN
                    (sender as TGridItem).picture.LoadFromFile(Seeker.jpg)
                    (sender as TGridItem).setCurrentState(Seeker)
                    Seeker=True
                    IF (Target = True) THEN
                        BEGIN
                            btnSaveLocations.Enabled=True
                            lblValidation.Enabled=False
                        END
                    END
                    ELSE
                        ShowMessage(Only 1 Seeker and 1 Target can be placed)
                    END
                END
            3:BEGIN
                IF (sender as TGridItem).getCurrentState = Target THEN
                    Target= False
                IF (sender as TGridItem).getCurrentState = Seeker THEN
                    Seeker= False
                (sender as TGridItem).picture.LoadFromFile(Closed.jpg)
                (sender as TGridItem).setCurrentState(Closed)
            END
        END
        END
    END PROCEDURE

```

#### 2.3.3.10 Procedure TfrmShortestPath.btnGridClick(Sender : TObject)

This will be used to set up the grid when the user click on create.

##### Local variables required:

Count1,Count2:Integer                      This will be used as a count.

PanelWidth,PanelHeight:Integer          This will be used to store the size of the panel.

It will run like this:

- It will validate that the text entered is less than 10 if not then it will show a message saying that it has to be less than 11.
- It will then check if it is less than 0 and if it is then a message will pop up letting the user know that it has to be more than 0.

- X and Y size will become equal to the two values entered in the width and height text boxes.
- The Panel width and height will both be set to 670
- It will then check to see if X and Y size are the same and if not then make Ysize equal to Xsize
- For each Grid Item it will be created on the grid as an open grid item and placed accordingly and size to the amount of grid items the user wants.
- It will then be added to a dictionary.

### Pseudocode

#### BEGIN

**IF** (StrToInt(edtwidth.Text) > 10) Or (StrToInt(edtHeight.Text) > 10) **THEN**

    ShowMessage(Grid size must be less than 11)

**ELSE IF** (edtwidth.Text) < 1 Or (edtHeight.Text) < 1 **THEN**

    ShowMessage(Grid size must be more than 0)

**ELSE**

**BEGIN**

    XSize= edtwidth.Text

    YSize= edtHeight.Text

    PanelWidth=670

    PanelHeight=670

**IF** XSize > YSize **THEN**

        Xsize=YSize

**ELSE**

        YSize=Xsize

**FOR** Count1 = 0 to XSize -1 **DO**

**BEGIN**

        for Count2 = 0 to YSize -1 **DO**

**BEGIN**

        item = TGridItem.Create(pnlGrid)

        item.setPosx(count2)

        item.setPosy(count1)

        item.setCurrentState(Open)

**WITH** item **DO**

**BEGIN**

        height=(PanelHeight Div YSize)

        width=(PanelWidth Div XSize)

        top=Count1\*height

        left=Count2\*Width

        parent = pnlGrid

        picture.LoadFromFile(Open.jpg)

        Stretch=True

        onClickEvent = ObjectChanger

        onClick = onClickEvent

**END**

    gridItemDictionary.Add(inttostr(count2) + inttostr(count1), item)

**END**

**END**

edtwidth.Enabled=False

edtHeight.Enabled=False

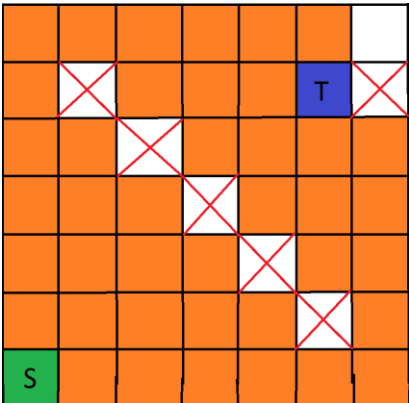
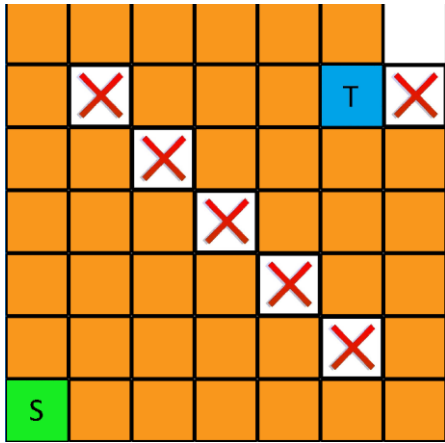
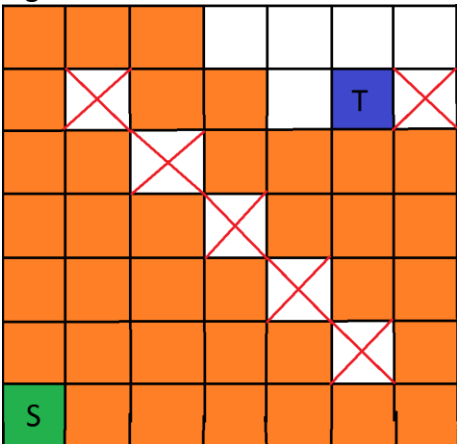
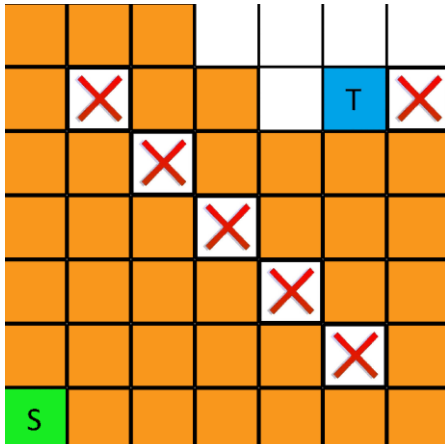
btnGrid.Enabled=False

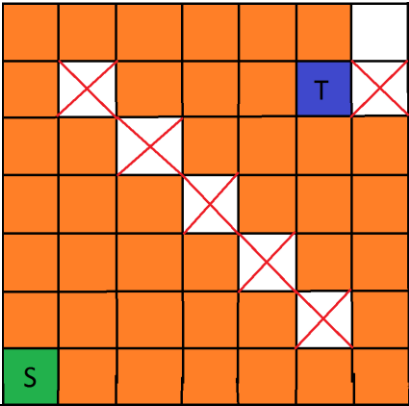
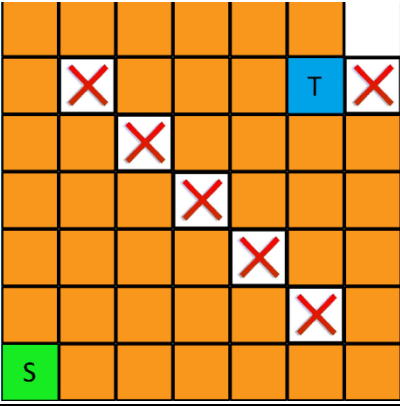
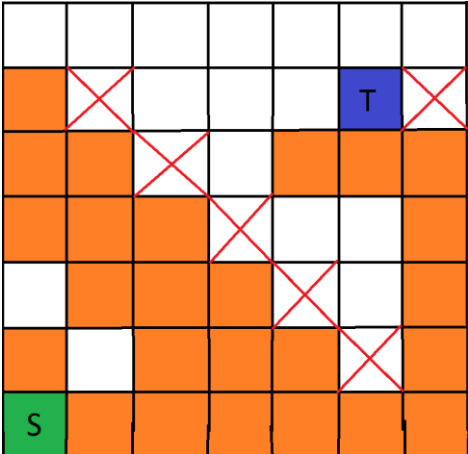
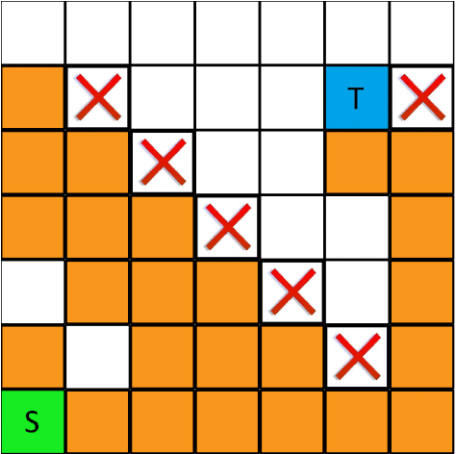
lblValidation.Enabled=True

```
        rgrpObject.Enabled=True  
    END  
END PROCEDURE
```

### 3 System Testing

#### 3.1 Typical Data Testing

Test No.	Reason	Input	Expected Result	Actual Result
1	Creating the Grid	(7x7) Pressing Create Button.	The grid should be created with 7 Grid items on the x axis and 7 on the y axis.	Time: 00:10
2	Placing the Target, Seeker and Target Gird Item	Target radio group option selected	The graph to change with the Target Seeker and closed items to be in place where I clicked on the graph.	Time: 00:22
3	Pressing Save locations Button	Clicking on the Button.	Nothing to appear to the user. Save locations to be saved	Time: 01:00
4	Running Breadth First search Algorithm.	Selecting Breadth First Search on the radio group then clicking on the search button	For the grid to show where the visited nodes the algorithm has visited. 	Time: 01:15 
5	Running Depth First Search algorithm.	Selecting Depth First Search on the radio group then pressing the search button.	The Grid should change from the previous algorithm to show the paths the depth first search algorithm has ran. 	Time: 01:18 
6	Going back to run an algorithm after one	Selecting Breadth First search	The grid should then change to as if the breadth first search had been ran for the first time so no changes should be made apart from the	Time: 01:21

	has previously been ran.	again on the radio group then clicking on the search button.	previous algorithm that had been ran being removed. 	
7	Running A* Algorithm.	Selecting A* Search on the radio group then clicking on the search button.	The Grid should change to see what where the A* Algorithm has visited. 	Time: 01:25 
8	Pressing Reset Button.	Click reset button	The Grid Should be deleted and the search algorithms should be disabled and enable the ability to create the grid again.	Time: 01:28

### 3.2 Erroneous Data Testing

Test No.	Reason	Valid Input	Actual Input	Expected Result	Actual Result
9	Validating that only 1 Target and 1 Seeker can be placed.	With a target and seeker already on the graph selecting the Target or Seeker from the radio group then clicking somewhere on the graph to try and place another Target or seeker.	Selecting target radio group option and clicking on grid square	An error message should come up to warn the user that only 1 Target and 1 Seeker can be placed.	Time: 00:36
10	Validating that once the save locations button has been clicked then the grid can no longer be edited.	Trying to click on the grid to change the Grid Item and trying to change the grid size or creating a duplicate copy of the grid.	Trying to click on the grid to change the Grid Item and trying to change the grid size or creating a duplicate copy of the grid.	An error Message should come up to say to the user that that the grid can no longer be changed and if they would like to do so then they must press the reset button.	Time: 01:00
11	Testing to make sure that when Objects have been disabled that they do not work as intended.	When the grid has been created but the locations haven't been saved yet then clicking on trying to run a search.	When the grid has been created but the locations haven't been saved yet then clicking on trying to run a search.	Nothing should happen as the objects should be disabled.	Time: 02:20
12	Testing to make sure that the grid size can't be changed and a duplicate of the original graph cannot be created.	The grid must have been created first and then try to change the values in the x and y axis boxes and trying to click create.	The grid must have been created first and then try to change the values in the x and y axis boxes and trying to click create.	Nothing Should happen as they will have to press reset as it could make the program faulty if there was a possible two graphs simultaneously.	Time: 02:31

### 3.3 Extreme Data Testing

Test No.	Reason	Input	Expected Result	Actual Result
13	Validating that the grid size must be between 0 and 11.	Entering 0 as the x and y axis and pressing create.	An error message to pop up and warn the user than it must be more than 0.	Time: 01:36
14	Validating that the grid size must be between 0 and 11.	Entering 11 as the x and y axis and pressing create.	An error message to pop up and warn the user than it must be less than 11.	Time: 01:47

### 3.4 Boundary Testing

Test No.	Reason	Input	Expected Result	Actual Result
15	Testing that the Lowest grid size works (1x1)	Entering 1 as the x and y axis and pressing the create button	The grid should be created appropriately with 1 on the x axis and 1 on the y axis.	Time: 01:56  (Once I ran this test I realised that this was not an appropriate size for a user to create so I have edited the code to make sure it is more than 4.)
16	Testing that the highest grid size works. (10x10)	Entering 10 as the x and y axis and pressing the create button	The grid should then be created appropriately with 10 Grid items on the x axis and 10 on the y axis.	Time: 02:04

When I was Programming and debugging the program to run the breadth first search it would take a rather long time for anything on the program to happen and the user would possibly think it had crashed. I fixed this issue by making my code more efficient because the problem was once it had visited a node it would still class it as one to look at so it would create an infinite loop where it would just keep visiting the same node backwards and forwards. I changed that so it no longer visited once it had been there once.

I also realised during my testing that my validation of the grid was not sensible. It would allow a grid of only 1 to be created which did not make sense as there had to be at least one Target and one Seeker. This would be impossible with just 1 node. I then also had to make its minimum four because anything lower would not be appropriate to run a search as the grid would be simply too small.



## 4 System Maintenance

### 4.1 Overview Guide

Variable	Why they are used
Item : TGridItem	This is used to store a copy of a grid item's attributes without overriding the "GridItem" variable.
onClickEvent : TNotifyEvent	This is used to notify the Object Changer procedure that the user is wanting to change a specific grid item.
GridItem:TGridItem	This is used to store the attributes of the appropriate Grid Item.
gridItemDictionary : TDictionary<string, TGridItem>	This is to store all the grid items and their positions.
connectionsFrom, connectionsTo : TList<string>	This is too store the connections to and from each neighbouring grid item.
XSize,YSize : integer	This is too store the height and width sizes the user has entered to create the grid to determine the size of each grid item and their position.
Seeker:Boolean	This is used to validate that a seeker has been placed and if true then no more seekers can be placed.
Target : Boolean	This is used to validate that a target has been placed and if true then no more targets can be placed.
StopChange : Boolean	This is used to validate that the user has already pressed saved locations button so it will prevent the user from changing the Grid afterwards.

#### 4.1.1 GridItemClass

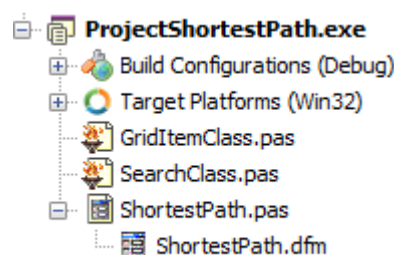
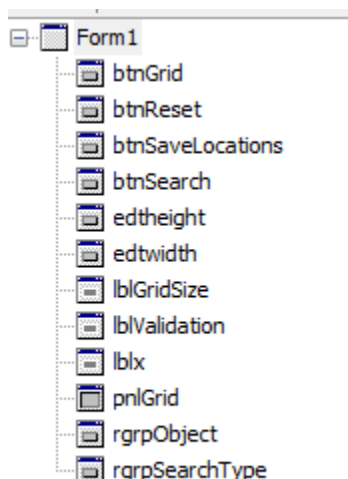
Appropriate Setters and Getters provided in commented code file.

#### 4.1.2 SearchClass

Procedure: TSearch.runBreadthFirstSearch	Why they are used
queue: TQueue<TGridItem>	This is used as the data structure for the search.
itemFound : Boolean	This is used to signify when the target has been found.
currentNode : TGridItem	This is used to hold the node being examined.
Procedure: TSearch.runDepthFirstSearch	Why they are used
stack : TStack <TGridItem>	This will be used for the data structure of the search.
itemFound : boolean	This will be used to state if the Target has been found or not.
currentNode : TGridItem	This will be used as the current node that the algorithm is currently using.
rightConnection: TGridItem	This will be used to store whatever is to the right of the current node.
leftConnection : TGridItem	This will be used to store whatever is to the left of the current node.

topConnection : TGridItem	This will be used to store whatever is to the top of the current node.
bottomConnection : TGridItem	This will be used to store whatever is to the bottom of the current node.
<b>Procedure: TSearch.sortList</b>	<b>Why they are used</b>
I : integer	This will be used as a count for the items in the list.
Swapped : boolean	This will be used to make sure the loop continues until there are no more items left in the list.
Temp : TGridItem	This is used as a temporary storing space to for an appropriate Grid Item in the list.
<b>Procedure: TSearch.runAStarSearch</b>	<b>Why they are used</b>
Found : boolean	This will be used to store if the Target has been found or not.
movementOptions : TList<TGridItem>	This will be used as a list to store different data at the same time.
I : integer	This will be used as a counter.
a,b,c : integer	This will be used to calculate the distance whilst storing different parts of the algorithm.

### 4.1.3 Objects and User Interface



- These are the objects that can be found on the initial form.
- The code can be found in the appendix section
- The design can be found on pages 16-19.
- GridItemClass.pas is the class for GridItem
- SearchClass.pas is the class for SearchClass
- The code for the classes can be found in the appendix section
- The design can be found on pages 20-26.
- Properties for each class and the form are discussed above.

This shows the form in design view. The objects on the form are shown in the table on the previous page. This is the initial form the user is presented with when the program initially runs

The interface has been built in this way as it is simple for the user to use as the panel on the left hand side is where the grid will be created and all the options on the right is what the user can interact with. It is in a simple layout as the user can start from the top and work their way down.

It has been designed so that the buttons and objects that are not yet enabled for the user to click on are slightly faded out and so the user can understand that the option is there but they have to do something first to then unlock this ability.

The full design can be found in the design section and the full code can be found in the appendix section.

## **4.2 Commented Code**

Please find the code in the appendix section.

## 5 Evaluation

### 5.1 Original Objective Evaluation Based On Client Feedback

#### 5.1.1 System Start up

These were the original 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

In my opinion the objectives for this section are all fully implemented and work accordingly. I could perhaps only have one text box for the user to enter the size of the grid they want which would mean that it is always going to be a square grid. I could also possibly implement more grid types such as one way roads etc. My client also agreed that these objectives had been met. He liked the fact that you could *“specify different sized grids and that you could customise the location of the seeker and target”*. He thought the interface was *“very intuitive”*. He did think that there was potential scope to be able to randomise the size of the grid and the positions of the target etc. I do think that this would have been a good addition. It was not in the objectives but I would certainly include it if I were to extend the program.

#### 5.1.2 Create the Grid

These were the objectives:

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

In my opinion, the objectives for this section are fully functional in the program however when the image is being loaded in it can take a significant amount of time depending on the size of the grid inputted by the user. This is a processing problem and can only be fixed by the user's hardware they use. I do think though that I should have been referring back to my objectives more often as I did get a problem during testing whereby a grid size of 1 could be specified. I did rectify the problem but should have noticed earlier that this was actually one of the objectives to include validation on the size of the grid. My client agreed the objectives were met. He was *“happy with the display of the grid, even sized squares, positioning”* etc. He liked the fact that the *“images used for the target and seeker were very easy to distinguish”*. He thought the *“red cross*

*was very relevant to illustrate a closed node*". He did say that he thought it *"would have been better if the grid appeared rather than watching each square being placed one by one"*. I am happy with the comments and would try to change the drawing of the grid. However, I think that it would be very difficult to do without the user wondering whether the program had crashed if it took a long time.

### 5.1.3 Saving the Grid Locations

These were the original objectives:

1. Ensure target grid has been specified and that the user cannot specify more than one target
2. Ensure seeker grid has been specified and that the user cannot specify more than one seeker
3. 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

In my opinion, the objectives in this section work completely fine and there are no issues at all. It successfully validates the target and seeker and also then goes on to determine the connections between the Grid Items as it should do with no errors as I have tested this with many different variables put into account. My client was happy that the system *"automatically did this without user input to physically check that everything is there that needs to be e.g. one target, one seeker"*. He thought the objectives were fully met.

### 5.1.4 Running a Search

These were the objectives:

1. 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
2. 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  $c^2=a^2+b^2$
  - 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

In my opinion the objectives in this section have been met. It is very simple to choose which search you would like to run and there is no chance making a mistake and trying to run more than one at the same time. The system then correctly identifies the target and seeker and runs the correct search between the two points. My testing has proven that the code I wrote does carry out the search with 100% accuracy and this

is very nicely visually displayed I think. My client agrees that the objectives have been met saying that *“the process has been done well and that he is happy that all of the searches can be correctly executed and displayed”*. However, he did point out that there could be a potential future addition of being *“able to place multiple targets and seekers and being able to run the search between each”*. I think this would be a very good addition to the program and would certainly incorporate it if I were extending the program. I think that this could be incorporated by allowing multiple grid items to be added using the current methods. However, rather than simply storing one of them, you could use a list to store multiple. The program could then loop through the list and run the search for each target and each seeker, this would then complete the future requirement.

### 5.1.5 Reset

These were the original objectives

3. Destroy all grid items
4. Reset all form defaults

In my opinion both objectives are met. The grid instantly disappears and the form is reset. My client agreed that *“both objectives were met and that no changes were required”*.

## 5.2 Further Development

### 5.2.1 Current Problems

The current problems with the program is that the time it can take to load the grid on the program once the Create button has been clicked. I did notice this myself and it was pointed out by my client. As I said I would try to incorporate this but do think it would be very hard to do for the reasons already given.

### 5.2.2 Improvements

This program could be improved by possibly introducing an algorithm that would then look at the visited nodes and calculate the shortest path it would have created rather than just showing all the visited nodes it had been to. I think this would be a good feature to include. This was an acceptable limitation.

Also an option for a larger grid could be implemented for the user to have more ability to experiment with the search algorithms so that a full diamond etc. could be seen. At the moment the pattern can be slightly difficult to detect.

Another feature could be for the user to be able to change the grid even after the connections and states of the nodes had been saved so that the user didn't have to click on reset every time they wanted to change the grid after a search algorithm had been ran.

Finally, it would be very useful if a snapshot of all three searches could be saved and displayed so that the user could see all three algorithms at the same time once they have run. A screenshot could be grabbed, saved and displayed side by side or each search could use the same grid but use different colours so that all three could be shown at once. If all three searches used the same node(s) then all three colours would show on the node.

## 6 Appendices

### 6.1 Appendix 1 – Initial Client Meeting

#### Meeting

Date: 25<sup>th</sup> September 2016

Time: 12:15

#### In Attendance

#### Client Background

HC: Can you please provide me with information regarding your background which may be applicable to this project?

MB: I am currently a PhD Student at the University of Sunderland in the department of Computer Science. My career goal is to become a lecturer/professor of Artificial Intelligence. To aid me with teaching various topics in the subject I am looking for visual educational tools to be developed showing the workings of different topics.

HC: What are the topics you would be teaching so I can have an idea of what I could program for you?

MB: I would be looking at possibly encryption algorithms and Search algorithms.

HC: How about I could program for an application to work out the steps an algorithm takes to find a shortest distance between two points?

MB: Yes that would be great, however I would need for there to be more than one search algorithm during the run so that I could compare and show how different algorithms run compared to others that may take more steps to find point B from point A.

#### The Project

HC: So if I created a grid with the ability for you to change where the seeker and target is and possibly some closed blocks where the search algorithm can't visit would this meet your requirements.

MB: Yes however I would like it if the user could enter in the size of the grid they want themselves so it could be more unique every time it is run.

HC: Yes no bother.

#### The Search Algorithms

HC: Is there a specific search algorithm you would like me to include in the program?

MB: Not any particular ones but the ones I have in mind are Dijkstras, Breadth First search and Depth First Search.

HC: Thank you that's great. I will go away and do some research on different types of search algorithms and which ones are more efficient.



## 6.2 Appendix 2 - Further Meeting

### Meeting

Date: 28<sup>th</sup> September 2016

Time: 13:00

### In Attendance

### Search Algorithms

HC: Thank you for coming today, I wanted to discuss the algorithms I have researched for the shortest path project.




MB: Yes.

HC: I have looked at five in total and deemed three the most sensible to program into this application.

MB: What are these five algorithms?

HC: I have looked at Dijkstras, Hill Climb, Breadth First Search, Depth First Search and A\* Search. I believe however that Breadth First and Depth First would be appropriate to include as they are similar and can compare the two in the steps they take on the application. I have also found that A\* Search is a little more reliable and efficient at finding the target where as Dijkstras and hill climbing are not as accurate.

MB: Yes I completely agree and it would be great to have that in my program to be able to compare the amount of steps each algorithm has taken.

 Reply  Reply All  Forward



College Project

05/10/2016



Can you please confirm that last week on Wednesday 12:15 we had a discussion about what your job role is and that you would like me to create a software that can find the shortest path on a grid to get from a designated spot to another.

We also discussed the different types of algorithms (Breadth First Search, Depth First Search, Dijkstras, Hill Climbing and A\* Search).

## 6.3 Appendix 3 - Client Feedback



14:43



I can confirm that the evaluation does include a true account of my feedback.

has been a pleasure to work with this year. We have had three formal meetings. The first two were at the beginning of the project and the third at the end where I tested his system and provided feedback.

He has also informally contacted me throughout the year as and when I have been in college. I did test his system at various times throughout the year providing feedback for him to act upon. We had conversations about aspects of the system where Harry needed clarification.

I am very pleased with the system as a whole and would certainly work with Harry again in the future.

Sincerely

---

Firstly, let me take this opportunity to thank you for agreeing to be my client for his computing project.

---

Firstly, let me take this opportunity to thank you for agreeing to be my client for his computing project.

has submitted his work and which includes an evaluation based upon your feedback. Could you please confirm that the evaluation is a true account and could you also provide feedback in terms of communication with you throughout the project.

Regards

---

## 6.4 Appendix 4 - Commented Program Listings