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.

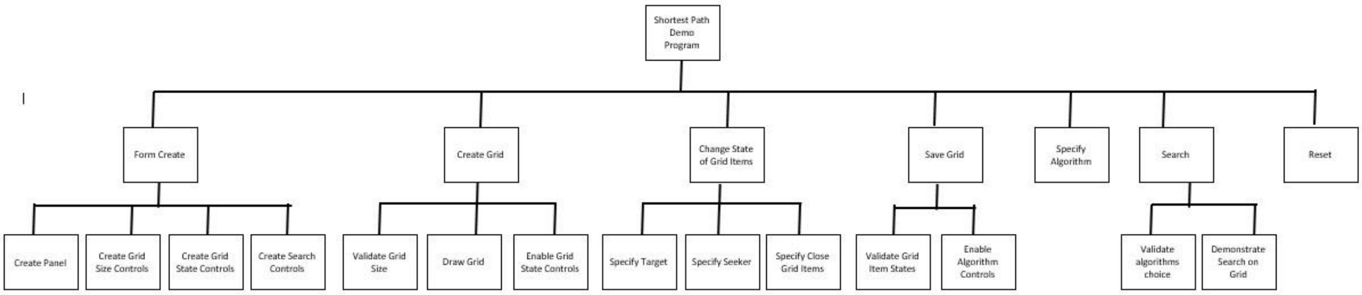
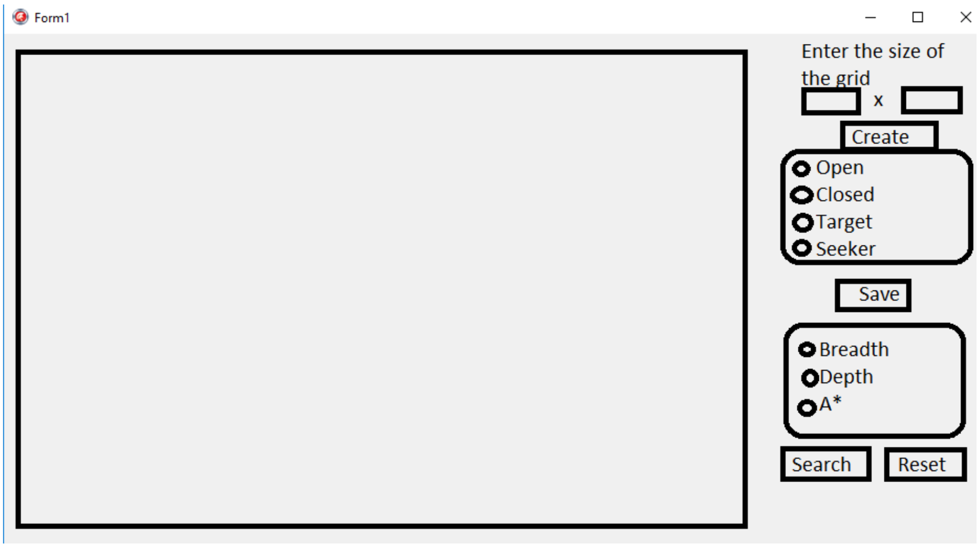

Shortest Path Demo Program
Form Create
Create Grid
Change State of Grid Items
Save Grid
Specify Algorithm
Search
Reset
Create Panel
Create Grid Size Controls
Create Grid State Controls
Create Search Controls
Validate Grid Size
Draw Grid
Enable Grid State Controls
Specify Target
Specify Seeker
Specify Close Grid Items
Validate Grid Item States
Enable Algorithm Controls
Validate algorithms choice
Demonstrate Search on Grid

Figure 5: Hierarchy Diagram

2.2 User Interface Design

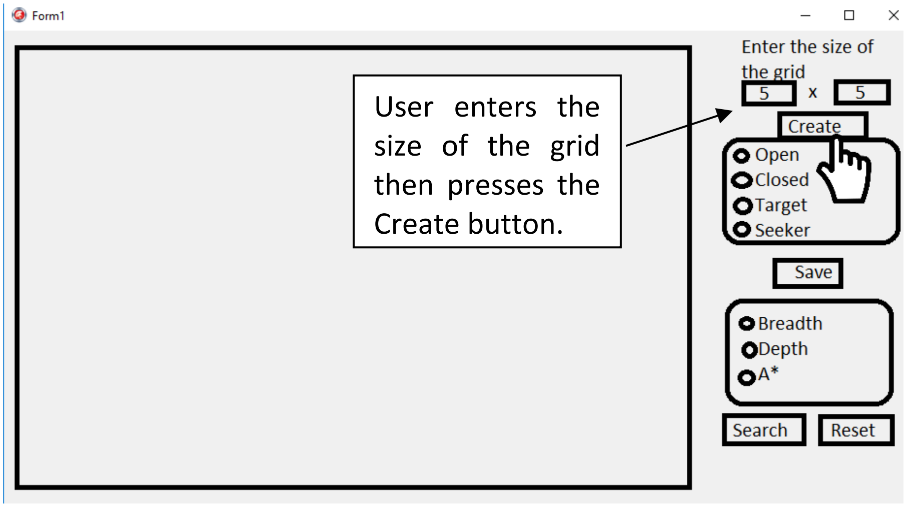
This is the user interface design in the stages in the order they would occur.

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

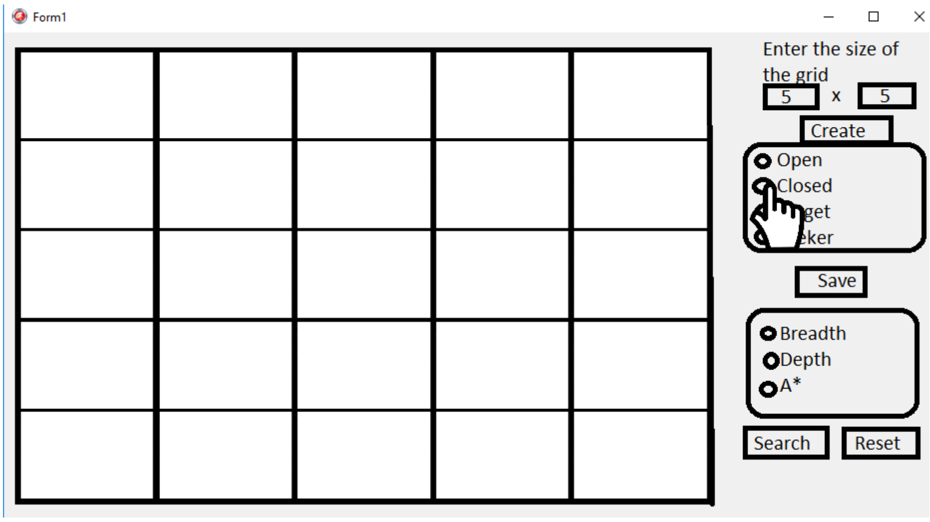


User enters the size of the grid then presses the Create button.

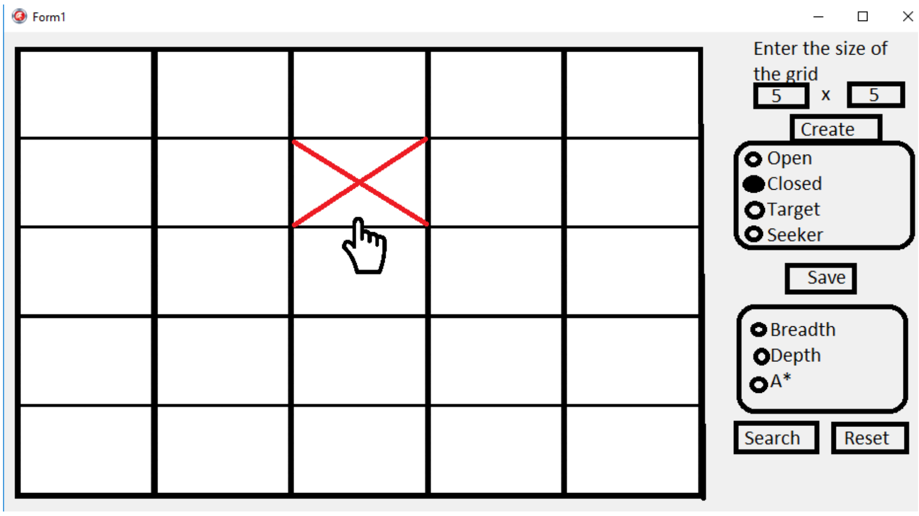
This is the stage where the user will specify the size of the grid. When create is clicked validation will be present to ensure both edit boxes have values between 1 and 10 as the maximum grid size is 10 x 10. If the size is valid, the gird will be created and the grid item type radio group and save 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 is it closed and not traversable. This can be seen below:



Form1

Enter the size of the grid

5 x 5

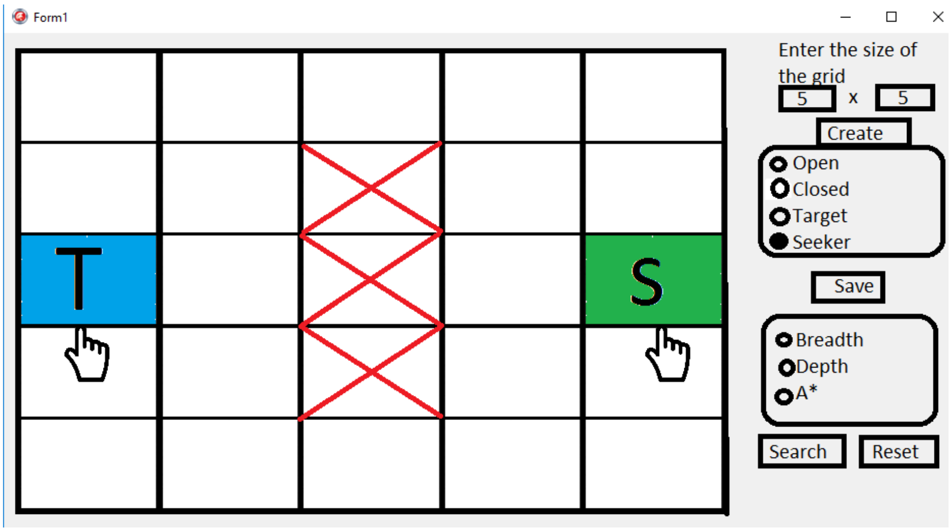
Create

○ Open
● Closed
○ Target
○ Seeker

Save

○ Breadth
○ Depth
○ A*

Search Reset



Form1

Enter the size of the grid

5 x 5

Create

○ Open
○ Closed
○ Target
● Seeker

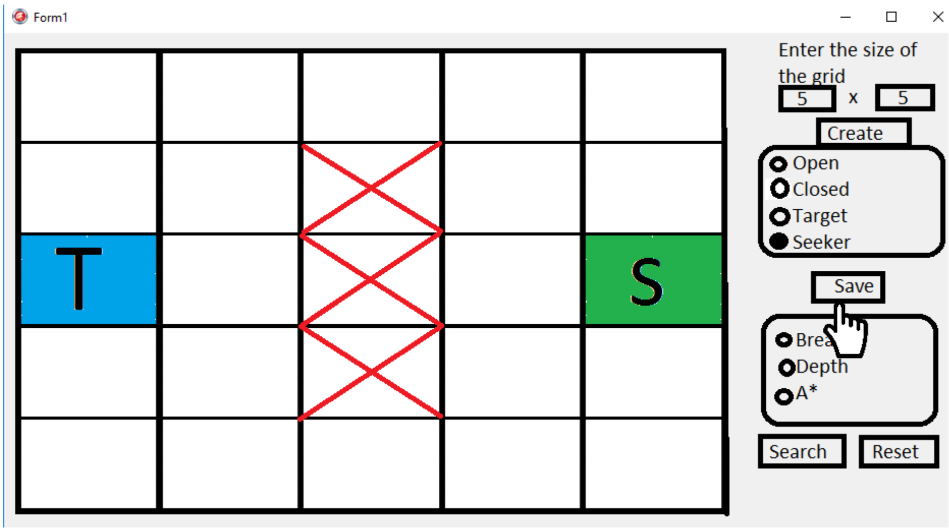
Save

○ Breadth
○ Depth
○ A*

Search Reset

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



Form1

Enter the size of the grid

5 x 5

Create

○ Open
○ Closed
○ Target
● Seeker

Save

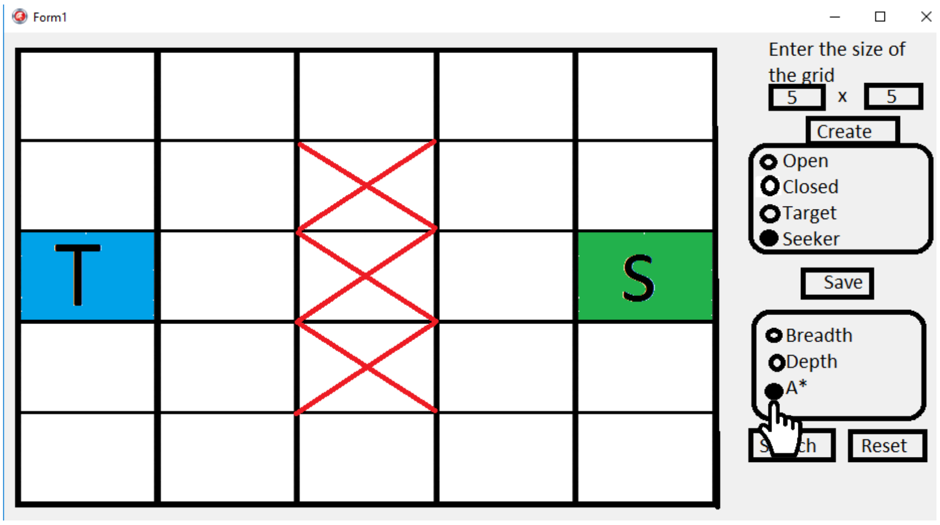
○ Breadth
○ Depth
○ A*

Search Reset

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



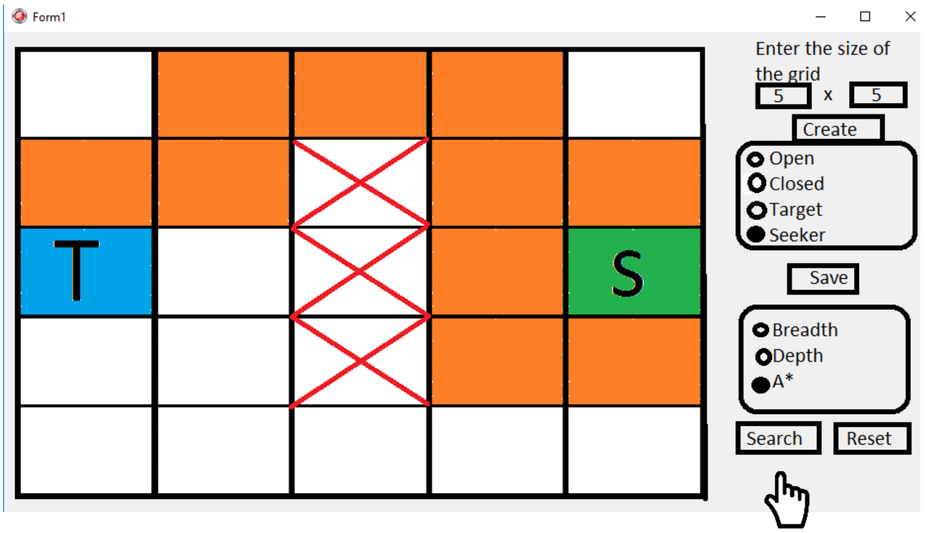
Enter the size of the grid 5 x 5

○ Open
○ Closed
○ Target
● Seeker

○ Breadth
○ Depth
● A*

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



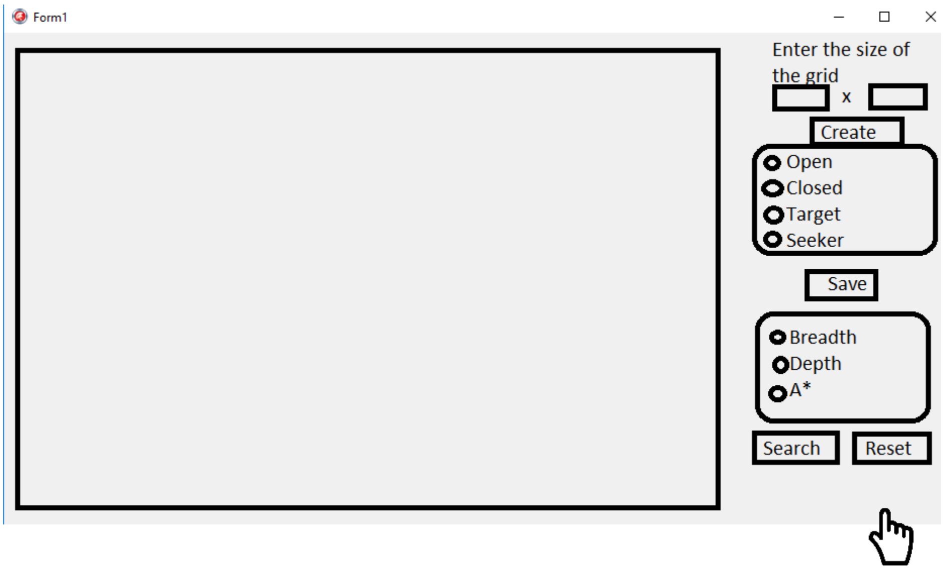
Enter the size of the grid 5 x 5

○ Open
○ Closed
○ Target
● Seeker

○ Breadth
○ Depth
● A*

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



Enter the size of the grid x

○ Open
○ Closed
○ Target
○ Seeker

○ Breadth
○ Depth
○ A*

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 gird 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 came 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 s3ets 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 came 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>`	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.

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.

Pseuodocode

```
 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 = pos x of the target node – pos x of the I position in the list.
  - B = pos y of the target node – pos y of the I position in the list.
  - A = A * A
  - B = B * B
  - C = 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.

Pseuodocode

```
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.

Pseuodocode

```
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.

Pseuodocode

```
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.

Pseuodocode

```
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 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.

Pseuodocode

```
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
```

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.

Pseuodocode

```
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.

Pseuodocode

```
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.

Pseuodocode

```
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.

Pseuodocode

```
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
  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 hast 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.

Pseuodocode

```
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
```