Sebastian Gaume

A\* Path Finder

Sebastian Gaume – Candidate Number: 0103 – Centre Number: 51518

Contents

[Analysis 2](#_Toc478501257)

[Background to Project 2](#_Toc478501258)

[Evidence of Analysis 2](#_Toc478501259)

[Description of Current System 2](#_Toc478501260)

[Proposed Solution 2](#_Toc478501261)

[Identification of End-User 3](#_Toc478501262)

[Acceptable Limitations 3](#_Toc478501263)

[Data Flow Diagram 3](#_Toc478501264)

[Data Dictionary 4](#_Toc478501265)

[Data Volumes 6](#_Toc478501266)

[Objectives 6](#_Toc478501267)

[Analysis appendices 8](#_Toc478501268)

[Design 9](#_Toc478501269)

[Overall System Design 9](#_Toc478501270)

[Hierarchy Chart 9](#_Toc478501271)

[System Flowchart 11](#_Toc478501272)

[Description of Algorithms 11](#_Toc478501273)

[Description of Data Structures 14](#_Toc478501274)

[Design of User Interface 14](#_Toc478501275)

[Technical solution 16](#_Toc478501276)

[Graphics 16](#_Toc478501277)

[Main 24](#_Toc478501278)

[Testing 33](#_Toc478501279)

[Black Box Testing 33](#_Toc478501280)

[White Box Testing 38](#_Toc478501281)

[Appraisal 48](#_Toc478501282)

[Comparison of Project Performance Against My General and Specific Objectives 48](#_Toc478501283)

[User Feedback authenticated by Assesor 50](#_Toc478501284)

[Analysis of User Feedback 50](#_Toc478501285)

[Possible Extensions 50](#_Toc478501286)

[Appraisal Appendices 52](#_Toc478501287)

# Analysis

## Background to Project

My project is called A\* Path Finder. I am going to be creating a path-finding system to be used in my client’s logical skills lessons, which the students and tutors can interact with to help the students better understand the logic behind a pathfinding algorithm.

My client, Nick Brearley of LogicOut, provides logical skills lessons for 8-14 year olds, and one of the topics they cover is pathfinding. The aim of the new system is to make the lessons more interactive for the children by giving them a system that they can use to explore pathfinding and try and get a better sense of how it works and to make the lessons more engaging.

## Evidence of Analysis

See analysis appendixes.

## Description of Current System

Currently LogicOut uses PowerPoints to deliver lessons about various subjects including the Manhattan pathfinding algorithm. These PowerPoints contain only diagrams and text, and LogicOut thinks that they aren’t interactive enough to engage the pupils fully. The tutors can also use a whiteboard to further illustrate points from the PowerPoint, however this still lacks interactivity for the pupils.

The Manhattan algorithm is a pathfinding algorithm based off of 3 key values, known as the G cost, the H cost and the F cost. The G cost is the total distance moved so far along this path, the H cost is the estimated cost to move from this point to the destination and the F cost is the previous two costs added together. You then compare the F costs of available positions to move to, and move to whichever has the lowest, giving you (rather quickly) a working path that may not be the quickest possible, but is a fairly short path relatively.

#### Issues with the current system

* The current system is not interactive enough for LogicOut as the children only get to see the text and diagrams on the PowerPoint backed up by drawings from the tutor
* Currently the students don’t get the chance to see the algorithm in action
* The current system doesn’t promote engagement and understanding as well as LogicOut would like

## Proposed Solution

My proposed solution is a system based on the Manhattan pathfinding algorithm – a heuristic algorithm that works by assigning nodes and paths different weights in order to find a short path (but not the shortest) quickly. The system will be created using Java, which is freely available, easily portable to different operating systems and the programming language I have most experience with, and is therefore easily available to my client and a suitable choice of language for my skillset. The system will also include a Graphical User Interface to make it easy for the tutors and pupils to use. This Graphical User Interface will contain an editing mode to allow the pupils to explore various scenarios and see how the pathfinder reacts and a simulation mode, which will display the path the algorithm finds. The algorithm itself is not very resource-intensive and the Graphical User Interface will be made with the hardware in mind, so that the system will run on my client’s hardware, however the hardware LogicOut already has is likely sufficient for any system I build. My system will not, however, contain a login system, as explained in acceptable limitations.

## Identification of End-User

The end-users are the tutors and students of LogicOut’s logical skills lessons. The students are from 8-14 years old, meaning that the lessons have to be engaging to keep their attention, as such my system has to be interactive. The system also has to be intuitive and easy to use so that the students don’t get confused or lost, but complex enough that it can convey the lesson effectively. The system should also be able to be used for exploration of the algorithm by the students.

The system should also be usable as a demonstration of what the tutor is teaching, meaning it should be clear and the interface should be simple to understand when being viewed on a projector or similar, without being too basic to take away from the interactivity. My client has also asked that I provide a login system so students can save their progress and or settings, so the system could also include easy-to-use networking elements.

## Acceptable Limitations

#### Budget

There is no budget for this project, as such the system must be made using freely available software or software already owned by LogicOut.

#### Time

While there is no specific deadline for this project it is required ‘as soon as possible’, so unnecessary features, not core to the program, will likely not be added.

#### Hardware

The hardware LogicOut has available is likely to be sufficient for any system that I build, but if there were a bottleneck in their system it would likely be the G.P.U. so my system should not be too graphically intensive.

#### Skill

My client, Nick, has stated that he would like a login system, however due to my not possessing the necessary skill and the fact that the program is wanted reasonably soon this feature will not be implemented.

## Data Flow Diagram

##### Level 0

Path

Map Data

User

User

##### Level 1

Compiled Map Data

Map Data

Path

Map Data

User

Desired Map Design

User

## Data Dictionary

#### Graphical User Interface

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Size/Range | Example |
| window\_Width | The width of the G.U.I. | Numeric | 640 - 1920 | 1280 |
| window\_Height | The height of the G.U.I. | Numeric | 360 - 1080 | 720 |
| node\_Width | The width of a node | Numeric | 10 - 100 | 50 |
| node\_Height | The height of a node | Numeric | 10 - 100 | 50 |
| display\_Width | The width of the graphical representation of the algorithm and its path | Numeric | 700 - 1840 | 1200 |
| display\_Height | The height of the graphical representation of the algorithm and its path | Numeric | 420 -1020 | 650 |
| is\_In\_Edit\_Mode | Is the G.U.I. currently in edit mode | True/False | N/A | False |

#### Nodes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Size/Range | Example |
| is\_Start\_Node | Is this the node that is the start location of the algorithm | True/False | N/A | True |
| is\_End\_Node | Is this the node that is the end location of the algorithm | True/False | N/A | False |
| is\_Wall\_Node | Is this node a wall node | True/False | N/A | False |
| node\_F\_Cost | The calculated f cost of the node | Numerical | 0-40000 | 50 |
| node\_G\_Cost | The calculated g cost of the node | Numerical | 0-40000 | 20 |
| node\_M\_Cost | The calculated m cost of the node | Numerical | 0-40000 | 30 |
| node\_Parent\_Node | The parent node of the node | Object | N/A | N/A |
| node\_X\_CoOrdinate | The x co-ordinate of the node | Numerical | 0-200 | 56 |
| node\_Y\_CoOridnate | The y co-ordinate of the node | Numerical | 0-200 | 34 |
| node\_Is\_In\_Path | Whether or not the node is in the path | True/False | N/A | True |

#### Algorithm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Size/Range | Example |
| end\_Node | The destination node for the path | Object | N/A | N/A |
| start\_Node | The starting position of the path | Object | N/A | N/A |
| lowest\_Cost\_Node | The node in the open list with the lowest F cost | Object | N/A | N/A |
| path\_Found | Whether or not a path has been found | True/False | N/A | False |
| is\_Impossible | Whether or not a path can be found, given the value of true if no path can be found | True/False | N/A | True |

## Data Volumes

#### Graphical User Interface

The graphical user interface will use 193 bits or 24 bytes and 1 bit.

#### Nodes

The algorithm will use 165 bits per node, or 20 bytes and 5 bits per node.

|  |  |  |
| --- | --- | --- |
| Number of Nodes | Bytes | Bits |
| 100 | 2062 | 4 |
| 1000 | 20625 | 0 |
| 10000 | 206250 | 0 |

#### Algorithm

The algorithm will use 2 bits for Booleans and 165\*3 bits to store information about nodes currently being processed, totalling 497 bits, or 62 bytes and 1 bit.

## Objectives

#### General Objectives

To create a computerized, interactive system that demonstrates the use of the A\* pathfinding algorithm and helps promote understanding of how the algorithm works for 8-14 year olds. The system will have a Graphical User Interface, a graphical representation of the algorithm in progress and the ability for students to edit the map that the algorithm will be pathfinding through.

#### Specific Objectives

1. A Graphical User Interface will be added
   1. The Graphical User Interface will have a simulation mode
      1. In simulation mode the G.U.I. will display the graphical representation of the map
      2. The program will launch in simulation mode
      3. In simulation mode there will be a button to run the algorithm
      4. In simulation mode there will be a button to switch to the editing mode
   2. The Graphical User Interface will have an editing mode
      1. In the editing mode the G.U.I. will display the graphical representation of the map
      2. In the editing mode the user will be able to select a node or path location from the graphical representation
         1. While a node is selected the user will be able to change the type of the node
         2. While a path location is selected the user will be able to add/remove the path
      3. In editing mode the user will be able to change the dimensions of the map (in nodes)
         1. The default dimensions will be 20 nodes by 20 nodes
      4. In the editing mode there will be a button to switch to simulation mode
2. A graphical representation of the map will be added
   1. Each node of the graph will be represented by a coloured circle
      1. Nodes that are part of the open list will be coloured white
      2. Nodes that are part of the closed list will be coloured black
      3. Wall nodes will be coloured orange
      4. The start node will be coloured green
      5. The end node will be coloured red
      6. Normal nodes (Non-Wall, Non-Start/End nodes) will be coloured grey
3. The A\* pathfinding algorithm will be implemented
   1. The algorithm will be based upon the Manhattan algorithm currently taught by LogicOut
   2. The algorithm will use lists of nodes
      1. The algorithm will use a ‘closed list’ to track all the nodes that it has already checked
      2. The algorithm will use an ‘open list’ to track all the nodes that it can check next
   3. The algorithm will find a path from the designated start node to the designated end node
4. The user will be able to run the algorithm
   1. The user will be able to press a button to start the algorithm
      1. When the algorithm starts it will be run based on how the graph has been edited
5. The algorithm will run on LogicOut’s hardware
   1. The algorithm will run on Windows 8 64-bit
   2. The algorithm will run on a computer with 6GB of RAM, an 8-core intel CPU clocked at 2.67GHz and a GeForce GTX 285

## Analysis appendices

# Design

## Overall System Design

My system is, at its core, a pathfinder, using the Manhattan algorithm for pathfinding. This means it operates in a quick and effective manner. First of all a map of nodes is to be generated, adding randomised wall nodes, and randomly placing the start and end nodes. At this point the graphical user interface will be initialised, presenting the user with three choices:

1. The first choice is to run the pathfinder. When the program is told to start the pathfinder it will begin by finding the start node and adding it to the ‘open’ list. From here it will create a list of all its neighbours, and compare them based off three scores: G, H and F. The G score is the distance from that node to the start node, the H score is the estimated distance from that node to the end node and the F cost is the G and the H cost summed together. It then places the start node in the ‘closed list’ and repeats the process with the node with the lowest cost, having set that node’s parent to the previous node in the path. This process continues until the end node is in the closed list, at which point the path is displayed.
2. The second option is to refresh the map, clearing any drawn path or any text output from the title.
3. The third option is to enter edit mode, where the user can generate a new random map with larger dimensions, smaller dimensions or equal dimensions, or the user may choose to return to the previous mode (simulation mode).

## Hierarchy Chart

change\_Node\_Type

increase\_Map\_Dimensions

commit\_Changes

Edit Mode

draw\_Path

draw\_Map

Simulation Mode

Graphical Representation

User Interface

User Interface

A\* Pathfinder

Algorithm

Edit Mode

decrease\_Map\_Dimensions

start\_Pathfinder

Simulation Mode

start\_Edit\_Mode

Algorithm

Manhattan Algorithm

Node Functions

List Functions

get\_CoOrdinates\_Array

does\_Contain

Manhattan Algorithm

get\_Path

find\_Path

Node Functions

is\_End\_Node

get\_Adjacent\_Nodes

is\_Traversable

set\_Parent\_Node

is\_Start\_Node

get\_Parent\_Node

Node Functions

set\_H\_Cost

set\_F\_Cost

calculate\_G\_Cost

calculate\_H\_Cost

set\_G\_Cost

calculate\_F\_Cost

Node Functions

get\_H\_Cost

get\_F\_Cost

get\_G\_Cost

## System Flowchart

Manhattan Algorithm

A\* Pathfinder

Random Map Data

Random Map Generator

New Map Data

Monitor

Path

Map Editor

Mouse

## Description of Algorithms

#### Calculate G Cost

Calculates the G cost of a passed node by recursively going back through its parentage – for each parent until the start node 1 is added to the G cost.

##### calculate\_G\_Cost (node)

g\_Cost <- 0

IF node CALL is\_Start\_Node = false

g\_Cost <- g\_Cost + 1

node <- node.get\_Parent\_Node()

g\_Cost <- calculate\_G\_Cost(node, g\_Cost)

ENDIF

RETURN g\_Cost

##### calculate\_G\_Cost(node, g\_Cost)

IF node CALL is\_Start\_Node = false

g\_Cost <- g\_Cost + 1

node <- node.get\_Parent\_Node

g\_Cost <- calculate\_G\_Cost(node, g\_Cost)

ENDIF

RETURN g\_Cost

#### Calculate H Cost

Calculates the H cost of a node by finding the absolute difference between the x and y coordinates of the passed node and the end node.

##### calculate\_H\_Cost (node, node\_List)

FOR EACH node IN node\_List

IF node.is\_End\_Node() = true THEN

end\_Node <- node

ENDIF

ENDFOR

node\_CoOrdinate\_Array <- node.get\_CoOrdinate\_Array()

end\_Node\_CoOrdiante\_Array <- end\_Node.get\_CoOrdinate\_Array()

h\_Cost <- node\_CoOrdinate\_Array[0] - end\_Node\_CoOrdiante\_Array[0]

h\_Cost <- h\_Cost + (node\_CoOrdinate\_Array[1] - end\_Node\_CoOrdiante\_Array[1])

IF h\_Cost < 0 THEN

h\_Cost <- h\_Cost \* -1

ENDIF

RETURN m\_Cost

#### Calculate F Cost

Calculates the F cost of a node by adding together its G cost and its H cost, either by being passed them or by retrieving them from a passed node.

##### calculate\_F\_Cost (node)

g\_Cost <- node.get\_G\_Cost()

h\_Cost <- node.get\_H\_Cost()

RETURN g\_Cost + h\_Cost

##### calculate\_F\_Cost (g\_Cost, m\_Cost)

RETURN g\_Cost + h\_Cost

#### Get Path

Recursively adds nodes to the path list by going backwards through the parentage of the end node. A function to draw the path is then called.

##### get\_Path (node)

WHILE node.is\_Start\_Node()= false

path\_List <- get\_Parent\_Node(node)

node <- get\_Parent\_Node(node)

CALL get\_Path\_Without\_Draw (node)

ENDWHILE

FOR EACH node IN path\_List

Node.set\_In\_Path()

ENDFOR

##### get\_Path\_Without\_Draw (NODE)

WHILE node CALL is\_Start\_Node = false

path\_List <- get\_Parent\_Node(node)

node <- get\_Parent\_Node(node)

get\_Path\_Without\_Draw(node)

ENDWHILE

RETURN path\_List

#### Find Path

Finds the path by using two paths, open and closed. The open list begins containing the start node, henceforth known as the current node. The algorithm then finds all the neighbour nodes of the current node, and compares the F costs of all the non-wall nodes. When it finds the neighbour with the lowest F cost it moves the current node to the closed list, and makes that neighbour the current node. It then sets the parent of the current node to the previous current node, and repeats the cycle until the end node is in the current list. At that point the path is found and the get path function is called.

##### find\_Path (node\_List)

FOR each node IN node\_List

IF node.is\_Start\_Node = true THEN

current\_Node <- node

ENDIF

IF node.is\_End\_Node = true THEN

end\_Node <- node

ENDIF

ENDFOR

open\_List <- current\_Node

WHILE open\_List.does\_Contain(end\_Node) = false AND open\_List.is\_Empty() = false

FOR each node IN open\_List

IF node.get\_F\_Cost() < current\_Node.get\_F\_Cost() THEN

current\_Node <- node

ENDIF

ENDFOR

closed\_List <- current\_Node

FOR each node IN (current\_Node.get\_Adjacent\_Nodes())

IF node.is\_Traversable = true AND closed\_List.does\_Contain(node)= false THEN

IF open\_List.does\_Contain(node)= false THEN

open\_List <- node

node.set\_Parent\_Node(current\_Node)

node.set\_G\_Cost(calculate\_G\_Cost(node)

node.set\_H\_Cost(calculate\_H\_Cost(node)

node.set\_F\_Cost(calculate\_F\_Cost(node)

ELSE THEN

node\_G\_Cost <- calculate\_G\_Cost(node)

node\_H\_Cost <- calculate\_H\_Cost(node)

node\_F\_Cost <- calculate\_F\_Cost(node\_M\_Cost, node\_G\_Cost

IF node\_F\_Cost < node.get\_F\_Cost() THEN

Node.set\_Parent\_Node(current\_Node)

ENDIF

ENDIF

ENDIF

ENDFOR

ENDWHILE

IF open\_List.does\_Contain(end\_Node) = true THEN

CALL get\_Path(end\_Node)

ENDIF

## Description of Data Structures

#### Lists

|  |  |
| --- | --- |
| Name | Purpose |
| node\_List | A list of all the nodes in the map |
| open\_List | A list of all the nodes the algorithm could travel to next |
| closed\_List | A list of all the nodes the algorithm cannot travel to next |
| path\_List | A list of all the nodes in the path |

#### Arrays

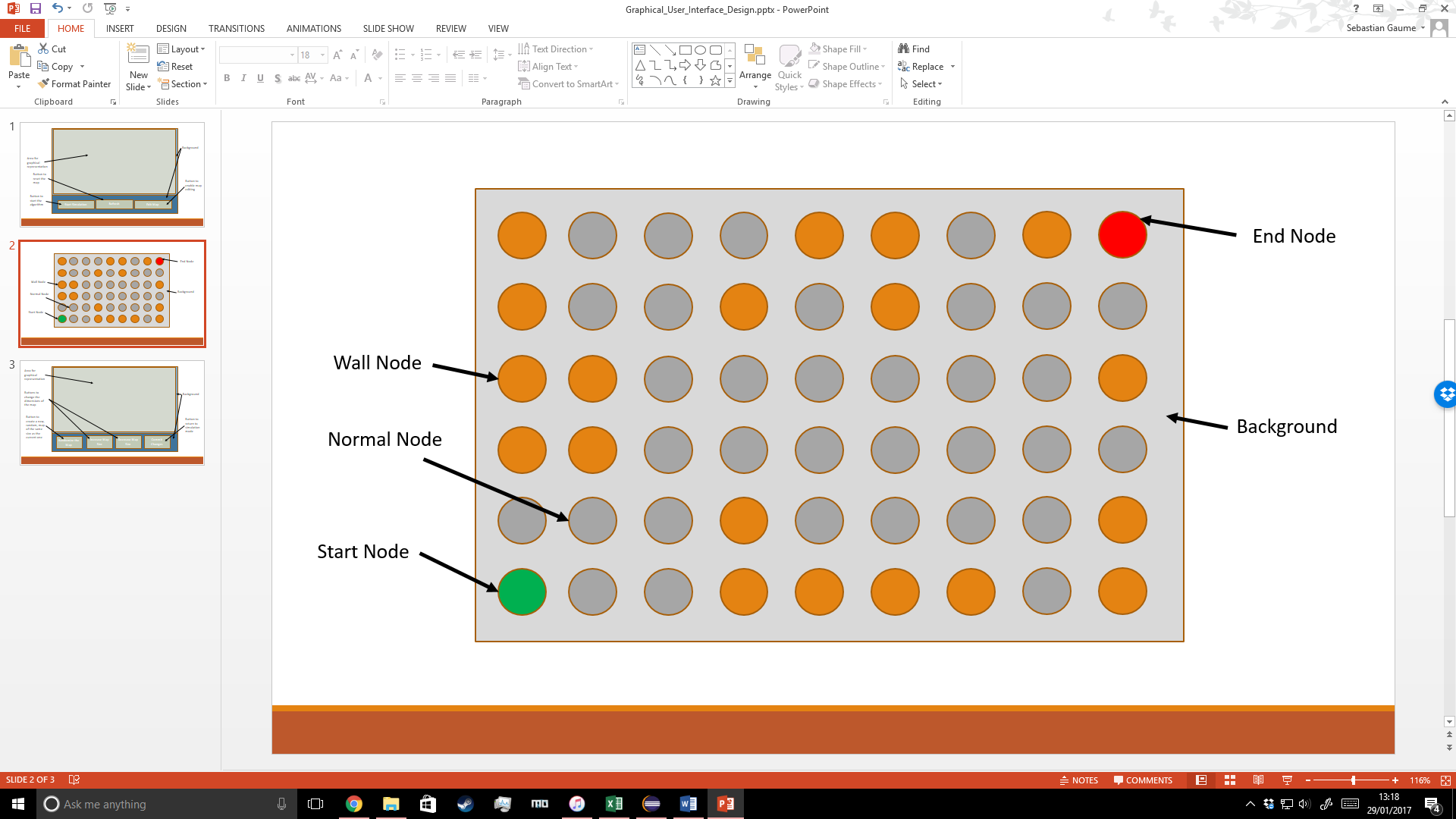
|  |  |
| --- | --- |
| Name | Purpose |
| node\_CoOridnate\_Array | An array containing the co-ordinates of the node being processed |
| end\_Node\_CoOrdinate\_Array | An array containing the co-ordinates of the end node |

## Design of User Interface

#### Simulation Mode

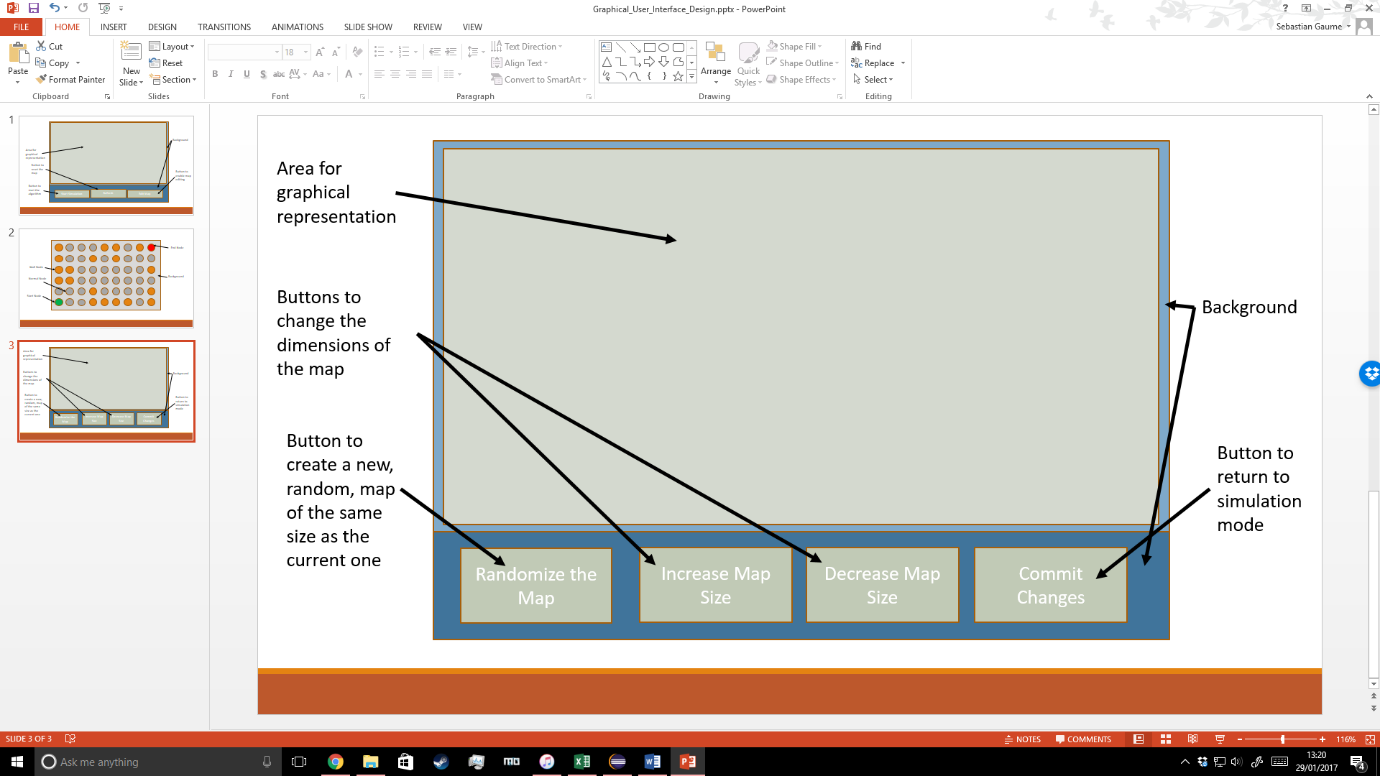
Simulation mode is the menu presented to the user when the system begins. The leftmost button will run the pathfinding algorithm for the displayed map, the middle button will clear the map of any output and the rightmost button will switch the menu to that of edit mode.

#### Graphical Representation Of Map



The graphical representation of the map is displayed at all times. The circles are the nodes, with different types of nodes being represented by different colours. Orange for wall nodes, green for the start node, red for the end node and grey for normal nodes. Once a path has been found any nodes in that path are displayed in cyan.

#### Edit Mode



Edit mode is accessible from simulation mode and consists of a menu with four buttons. The leftmost button creates a new random map with the same dimensions as the current one, the middle left button creates a new random map with dimensions 5 wider and taller, the middle right button creates a new random map with dimensions 5 less wide and 5 shorter and the rightmost button returns the user to simulation mode.

# Technical solution

## Graphics

#### Graphical\_User\_Interface

package graphics**;**

**import** java**.**awt**.**Color**;**

**import** java**.**awt**.**Dimension**;**

**import** java**.**awt**.**event**.**ActionEvent**;**

**import** java**.**awt**.**event**.**ActionListener**;**

**import** javax**.**swing**.**JButton**;**

**import** javax**.**swing**.**JFrame**;**

**import** javax**.**swing**.**JPanel**;**

**import** main**.**Node**;**

**import** main**.**Program**;**

public class Graphical\_User\_Interface **implements** Runnable **{**

// Variables containing various dimensions

public static int graphical\_User\_Interface\_Width **=** 1280**;**

public static int graphical\_User\_Interface\_Height **=** 720**;**

public int visualizer\_Width**,** visualizer\_Height**,** map\_Width**,** map\_Height**;**

// Swing components to make up the G.U.I.

public Simulation\_Visualizer simulation\_Visualizer**;**

private JFrame graphical\_User\_Interface**;**

private JButton start\_Button**,** edit\_Mode\_Button**,** refresh\_Button**,** increase\_Dimensions\_Button**,** decrease\_Dimensions\_Button**,** simulation\_Mode\_Button**,** randomize\_Button**;**

// Variables used to check what is running and what which mode the G.U.I. is

// in

public boolean pathfinder\_Is\_Started **=** **false;**

private boolean simulation\_Mode **=** **true;**

private boolean edit\_Mode\_Initialized **=** **false;**

private boolean simulation\_Mode\_Started **=** **true;**

private boolean edit\_Mode\_Started **=** **false;**

// The constructor for the G.U.I. class

public Graphical\_User\_Interface**(**int map\_Width**,** int map\_Height**,** int visualizer\_Width**,** int visualizer\_Height**)** **{**

// Setting local variables

**this.**visualizer\_Width **=** visualizer\_Width**;**

**this.**visualizer\_Height **=** visualizer\_Height**;**

**this.**map\_Width **=** map\_Width**;**

**this.**map\_Height **=** map\_Height**;**

// Creating the JFrame

graphical\_User\_Interface **=** **new** JFrame**();**

graphical\_User\_Interface**.**setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

graphical\_User\_Interface**.**setResizable**(false);**

graphical\_User\_Interface**.**setPreferredSize**(new** Dimension**(**graphical\_User\_Interface\_Width**,** graphical\_User\_Interface\_Height**));**

graphical\_User\_Interface**.**getContentPane**().**setLayout**(null);**

// Creating Swing components and adding them to the JFrame

JPanel lower\_Background **=** **new** JPanel**();**

lower\_Background**.**setBackground**(new** Color**(**62**,** 80**,** 180**));**

lower\_Background**.**setBounds**(**0**,** 550**,** 1280**,** 140**);**

graphical\_User\_Interface**.**getContentPane**().**add**(**lower\_Background**);**

start\_Button **=** **new** JButton**(**"Start Pathfinder"**);**

start\_Button**.**setBounds**(**50**,** 580**,** 350**,** 75**);**

graphical\_User\_Interface**.**getContentPane**().**add**(**start\_Button**);**

edit\_Mode\_Button **=** **new** JButton**(**"Edit Mode"**);**

edit\_Mode\_Button**.**setBounds**(**880**,** 580**,** 350**,** 75**);**

graphical\_User\_Interface**.**getContentPane**().**add**(**edit\_Mode\_Button**);**

refresh\_Button **=** **new** JButton**(**"Refresh"**);**

refresh\_Button**.**setBounds**(**465**,** 580**,** 350**,** 75**);**

graphical\_User\_Interface**.**getContentPane**().**add**(**refresh\_Button**);**

simulation\_Visualizer **=** **new** Simulation\_Visualizer**();**

simulation\_Visualizer**.**setBounds**(**5**,** 0**,** visualizer\_Width**,** visualizer\_Height**);**

graphical\_User\_Interface**.**getContentPane**().**add**(**simulation\_Visualizer**);**

// Finishing the creation of the JFrame

graphical\_User\_Interface**.**setVisible**(true);**

graphical\_User\_Interface**.**pack**();**

**}**

// A setter method for the mode of the G.U.I.

public void choose\_Mode**(**boolean simulation\_Mode**)** **{**

**this.**simulation\_Mode **=** simulation\_Mode**;**

**}**

// The run() method for the G.U.I.'s thread

@Override

public void run**()** **{**

// While loop that runs while the G.U.I. is in simulation mode

**while** **(**simulation\_Mode**)** **{**

// If statements to determine how the title should be formatted

**if** **(**Program**.**pathfinder**.**is\_Impossible**)** **{**

graphical\_User\_Interface**.**setTitle**(**"A\* Pathfinder - " **+** "Width: " **+** Program**.**pathfinder**.**map\_Width

**+** ", Height: " **+** Program**.**pathfinder**.**map\_Height **+** " IMPOSSIBLE"**);**

**}** **else** **if** **(**Program**.**pathfinder**.**path\_Found**)** **{**

graphical\_User\_Interface**.**setTitle**(**"A\* Pathfinder - " **+** "Width: " **+** Program**.**pathfinder**.**map\_Width

**+** ", Height: " **+** Program**.**pathfinder**.**map\_Height **+** " PATH FOUND"**);**

**}** **else** **{**

graphical\_User\_Interface**.**setTitle**(**"A\* Pathfinder - " **+** "Width: " **+** Program**.**pathfinder**.**map\_Width

**+** ", Height: " **+** Program**.**pathfinder**.**map\_Height**);**

**}**

// If statement that checks that the G.U.I. has updated to show // that it is in simulation mode, and updates it if it hasn't

**if** **(!**simulation\_Mode\_Started**)** **{**

graphical\_User\_Interface**.**getContentPane**()**

**.**add**(**edit\_Mode\_Button**);**

graphical\_User\_Interface**.**getContentPane**()**

**.**add**(**start\_Button**);**

graphical\_User\_Interface**.**getContentPane**()**

**.**add**(**refresh\_Button**);**

edit\_Mode\_Started **=** **false;**

simulation\_Mode\_Started **=** **true;**

graphical\_User\_Interface**.**repaint**();**

**}**

// Creating the ActionListeners for the buttons

ActionListener start\_Button\_Listener **=** **new** ActionListener**()** **{**

@Override

public void actionPerformed**(**ActionEvent e**)** **{**

// Runs if the 'start path-finder' button is // pressed and the path-finder hasn't

// already been started

**if** **(**e**.**getSource**()** **==** start\_Button **&&** pathfinder\_Is\_Started **==** **false)** **{**

// Updates variables so that the pathfinder // will run

pathfinder\_Is\_Started **=** **true;**

Program**.**pathfinder**.**path\_Found **=** **false;**

**}**

**}**

**};**

ActionListener refresh\_Button\_Listener **=** **new** ActionListener**()** **{**

@Override

public void actionPerformed**(**ActionEvent e**)** **{**

// Runs if the 'refresh' button is pressed and the // program has already found a path

**if** **(**Program**.**pathfinder**.**path\_List **!=** **null** **&&** e**.**getSource**()** **==** refresh\_Button**)** **{**

// Resets all the nodes, lists and // variables associated with storing

// the path

**for** **(**Node node\_Iterator **:** Program**.**pathfinder**.**path\_List**)** **{**

node\_Iterator**.**set\_Not\_In\_Path**();**

**}**

Program**.**pathfinder**.**path\_List**.**clear**();**

Program**.**pathfinder**.**is\_Impossible **=** **false;**

Program**.**pathfinder**.**path\_Found **=** **false;**

// Repaints the graphical representation

simulation\_Visualizer**.**repaint**();**

**}**

**}**

**};**

ActionListener edit\_Button\_Listener **=** **new** ActionListener**()** **{**

@Override

public void actionPerformed**(**ActionEvent e**)** **{**

// Runs if the 'edit mode' button is pressed and // the program is not in the process of

// finding a path

**if** **(**e**.**getSource**()** **==** edit\_Mode\_Button **&&** **!**pathfinder\_Is\_Started**)** **{**

// Sets the mode of the G.U.I. to edit mode

choose\_Mode**(false);**

// Removes all the simulation mode buttons

graphical\_User\_Interface**.**getContentPane**()**

**.**remove**(**edit\_Mode\_Button**);**

graphical\_User\_Interface**.**getContentPane**()**

**.**remove**(**start\_Button**);**

graphical\_User\_Interface**.**getContentPane**()**

**.**remove**(**refresh\_Button**);**

// Repaints the G.U.I.

graphical\_User\_Interface**.**repaint**();**

**}**

**}**

**};**

// Adds the new ActionListeners to their respective buttons

start\_Button**.**addActionListener**(**start\_Button\_Listener**);**

refresh\_Button**.**addActionListener**(**refresh\_Button\_Listener**);**

edit\_Mode\_Button**.**addActionListener**(**edit\_Button\_Listener**);**

// Runs if the variables telling the program to start the

// path-finder

// are true

**if** **(**pathfinder\_Is\_Started**)** **{**

// Starts the path-finder and updates the variable // telling the program to start the path-finder

// so it's not run twice

Program**.**pathfinder

**.**find\_Path**(**Program**.**pathfinder**.**map**.**get\_Node\_List**());**

pathfinder\_Is\_Started **=** **false;**

**}**

// Repaints all of the components

start\_Button**.**repaint**();**

edit\_Mode\_Button**.**repaint**();**

refresh\_Button**.**repaint**();**

simulation\_Visualizer**.**repaint**();**

// The thread sleeps so that the components have time to // repaint

**try** **{**

Thread**.**sleep**(**10**);**

**}** **catch** **(**Exception e**)** **{**

System**.**err**.**println**(**e**);**

**}**

// Excess ActionListeners are removed from the buttons

start\_Button**.**removeActionListener**(**start\_Button\_Listener**);**

refresh\_Button**.**removeActionListener**(**refresh\_Button\_Listener**);**

edit\_Mode\_Button**.**removeActionListener**(**edit\_Button\_Listener**);**

**}**

// While loop that runs while the G.U.I. is in edit mode

**while** **(!**simulation\_Mode**)** **{**

// If statements to determine how the title should be formatted

**if** **(**Program**.**pathfinder**.**is\_Impossible**)** **{**

graphical\_User\_Interface**.**setTitle**(**"A\* Pathfinder - " **+** "Width: " **+** Program**.**pathfinder**.**map\_Width

**+** ", Height: " **+** Program**.**pathfinder**.**map\_Height **+** " IMPOSSIBLE"**);**

**}** **else** **if** **(**Program**.**pathfinder**.**path\_Found**)** **{**

graphical\_User\_Interface**.**setTitle**(**"A\* Pathfinder - " **+** "Width: " **+** Program**.**pathfinder**.**map\_Width

**+** ", Height: " **+** Program**.**pathfinder**.**map\_Height **+** " PATH FOUND"**);**

**}** **else** **{**

graphical\_User\_Interface**.**setTitle**(**"A\* Pathfinder - " **+** "Width: " **+** Program**.**pathfinder**.**map\_Width

**+** ", Height: " **+** Program**.**pathfinder**.**map\_Height**);**

**}**

// If statement that checks that the G.U.I. has updated to show // that it is in edit mode for the first time and instantiates

// the edit mode buttons and updates the G.U.I. if it hasn't

**if** **(!**edit\_Mode\_Initialized **&&** **!**edit\_Mode\_Started**)** **{**

// Instantiating new JButtons for edit mode and adding // them to the JFrame

increase\_Dimensions\_Button **=** **new** JButton**(**"Increase Map Dimensions"**);**

increase\_Dimensions\_Button**.**setBounds**(**50**,** 580**,** 250**,** 75**);**

graphical\_User\_Interface**.**getContentPane**()**

**.**add**(**increase\_Dimensions\_Button**)**

decrease\_Dimensions\_Button **=** **new** JButton**(**"Decrease Map Dimensions"**);**

decrease\_Dimensions\_Button**.**setBounds**(**360**,** 580**,** 250**,** 75**);**

graphical\_User\_Interface**.**getContentPane**()**

**.**add**(**decrease\_Dimensions\_Button**);**

randomize\_Button **=** **new** JButton**(**"Randomize the Map"**);**

randomize\_Button**.**setBounds**(**670**,** 580**,** 250**,** 75**);**

graphical\_User\_Interface**.**getContentPane**()**

**.**add**(**randomize\_Button**);**

simulation\_Mode\_Button **=** **new** JButton**(**"Simulation Mode"**);**

simulation\_Mode\_Button**.**setBounds**(**980**,** 580**,** 250**,** 75**);**

graphical\_User\_Interface**.**getContentPane**()**

**.**add**(**simulation\_Mode\_Button**);**

// Updating variables to reflect that the G.U.I. has // entered edit mode for the first time

edit\_Mode\_Started **=** **true;**

edit\_Mode\_Initialized **=** **true;**

simulation\_Mode\_Started **=** **false;**

// Repaints the G.U.I.

graphical\_User\_Interface**.**repaint**();**

**}**

// If statement that checks that the G.U.I. has updated to show // that it is in edit mode and if it hasn't then it updates

// the GUI

**if** **(!**edit\_Mode\_Started **&&** edit\_Mode\_Initialized**)** **{**

graphical\_User\_Interface**.**getContentPane**()**

**.**add**(**increase\_Dimensions\_Button**);**

graphical\_User\_Interface**.**getContentPane**()**

**.**add**(**decrease\_Dimensions\_Button**);**

graphical\_User\_Interface**.**getContentPane**()**

**.**add**(**simulation\_Mode\_Button**);**

graphical\_User\_Interface**.**getContentPane**()**

**.**add**(**randomize\_Button**);**

edit\_Mode\_Started **=** **true;**

simulation\_Mode\_Started **=** **false;**

graphical\_User\_Interface**.**repaint**();**

**}**

// If statement to run button listeners when in edit mode and // the GUI has been updated to edit mode

**if** **(**edit\_Mode\_Started**)** **{**

ActionListener increase\_Button\_Action\_Listener **=** **new** ActionListener**()** **{**

@Override

// Runs if the increase dimensions button has been

// pressed

public void actionPerformed**(**ActionEvent e**)** **{**

**if** **(**e**.**getSource**()** **==** increase\_Dimensions\_Button**)** **{**

**if** **(**Program**.**pathfinder**.**map\_Width **<** 200 **&&** Program**.**pathfinder**.**map\_Height **<** 200**)** **{**

// Increases the size of the // map as long as it

// has not reached a limit

Program**.**pathfinder**.**map\_Width **=** Program**.**pathfinder

**.**map\_Width **+** 5**;**

Program**.**pathfinder**.**map\_Height **=** Program**.**pathfinder

**.**map\_Height **+** 5**;**

**}**

// Creates a new random map with // increased dimensions

Program**.**pathfinder

**.**create\_Map**(**Program**.**pathfinder

**.**map\_Width**,** Program**.**pathfinder**.**map\_Height**);**

Program**.**pathfinder**.**is\_Impossible **=** **false;**

Program**.**pathfinder**.**path\_Found **=** **false;**

**}**

**}**

**};**

increase\_Dimensions\_Button

**.**addActionListener**(**increase\_Button\_Action\_Listener**);**

ActionListener decrease\_Button\_Action\_Listener **=** **new** ActionListener**()** **{**

@Override

public void actionPerformed**(**ActionEvent e**)** **{**

// Runs if the decrease dimensions button // has been pressed

**if** **(**e**.**getSource**()** **==** decrease\_Dimensions\_Button**)** **{**

**if** **(**Program**.**pathfinder**.**map\_Width **>** 10 **&&** Program**.**pathfinder**.**map\_Height **>** 10**)** **{**

// Decreases the size of the // map as long as it

// has not reached a minimum // size limit

Program**.**pathfinder**.**map\_Width **=** Program**.**pathfinder**.**map\_Width **-** 5**;**

Program**.**pathfinder**.**map\_Height **=** Program**.**pathfinder**.**map\_Height **-** 5**;**

**}**

// Creates a new random map with the new dimensions

Program**.**pathfinder

**.**create\_Map**(**Program**.**pathfinder**.**map\_Width**,** Program**.**pathfinder**.**map\_Height**);**

Program**.**pathfinder**.**is\_Impossible **=** **false;**

Program**.**pathfinder**.**path\_Found **=** **false;**

**}**

**}**

**};**

decrease\_Dimensions\_Button

**.**addActionListener**(**decrease\_Button\_Action\_Listener**);**

ActionListener random\_Button\_Action\_Listener **=** **new** ActionListener**()** **{**

@Override

public void actionPerformed**(**ActionEvent e**)** **{**

// Runs if the randomise button has been // pressed

**if** **(**e**.**getSource**()** **==** randomize\_Button**)** **{**

// Creates a new random map Program**.**pathfinder

**.**create\_Map**(**Program

**.**pathfinder**.**map\_Width**,**

Program**.**pathfinder**.**map\_Height**);**

Program**.**pathfinder**.**is\_Impossible **=** **false;**

Program**.**pathfinder**.**path\_Found **=** **false;**

**}**

**}**

**};**

randomize\_Button**.**addActionListener

**(**random\_Button\_Action\_Listener**);**

ActionListener simulation\_Button\_Listener **=** **new** ActionListener**()** **{**

@Override

public void actionPerformed**(**ActionEvent e**)** **{**

// Runs if the simulation mode button has // been pressed

**if** **(**e**.**getSource**()** **==** simulation\_Mode\_Button**)** **{**

// Updates the mode of the GUI to // simulation mode,

// removes the edit mode buttons and // repaints the

// interface so it loads the // simulation mode buttons

choose\_Mode**(true);**

graphical\_User\_Interface

**.**getContentPane**()**

**.**remove**(**increase\_Dimensions\_Button**);**

graphical\_User\_Interface

**.**getContentPane**()**

**.**remove**(**decrease\_Dimensions\_Button**);**

graphical\_User\_Interface

**.**getContentPane**()**

**.**remove**(**simulation\_Mode\_Button**);**

graphical\_User\_Interface

**.**getContentPane**()**

**.**remove**(**randomize\_Button**);**

graphical\_User\_Interface**.**repaint**();**

simulation\_Mode\_Started **=** **false;**

**}**

**}**

**};**

simulation\_Mode\_Button

**.**addActionListener**(**simulation\_Button\_Listener**);**

// Repaints all the buttons to prevent a flickering // visual glitch

increase\_Dimensions\_Button**.**repaint**();**

decrease\_Dimensions\_Button**.**repaint**();**

simulation\_Mode\_Button**.**repaint**();**

randomize\_Button**.**repaint**();**

simulation\_Visualizer**.**repaint**();**

// Puts the thread to sleep to allow the program time to //update

**try** **{**

Thread**.**sleep**(**10**);**

**}** **catch** **(**Exception e**)** **{**

System**.**err**.**println**(**e**);**

**;**

**}**

// Removes the button listeners created this loop so that there

// aren't multiple running at once

increase\_Dimensions\_Button

**.**removeActionListener**(**increase\_Button\_Action\_Listener**);**

decrease\_Dimensions\_Button

**.**removeActionListener**(**decrease\_Button\_Action\_Listener**);**

randomize\_Button

**.**removeActionListener**(**random\_Button\_Action\_Listener**);**

simulation\_Mode\_Button

**.**removeActionListener**(**simulation\_Button\_Listener**);**

**}**

**}**

// Re-runs the thread after leaving edit mode

run**();**

**}**

**}**

#### Simulation\_Visualizer

package graphics**;**

**import** java**.**awt**.**Color**;**

**import** java**.**awt**.**Graphics**;**

**import** java**.**util**.**Iterator**;**

**import** java**.**util**.**List**;**

**import** javax**.**swing**.**JPanel**;**

**import** main**.**Node**;**

**import** main**.**Program**;**

@SuppressWarnings**(**"serial"**)**

public class Simulation\_Visualizer **extends** JPanel **{**

// A private instance of the list of nodes on the map

private List**<**Node**>** node\_List**;**

// Information about how wide and tall a node should be when displayed

public int node\_Width**,** node\_Height**;**

// The method that paints the visualisation onto a JPanel

@Override

public void paintComponent**(**Graphics g**)** **{**

**this.**node\_List **=** Program**.**pathfinder**.**map**.**get\_Node\_List**();**

**super.**paintComponent**(**g**);**

Iterator**<**Node**>** node\_Iterator **=** node\_List**.**iterator**();**

Node temp **=** **null;**

// Finds a suitable width and height for nodes to be displayed

node\_Width **=** Program**.**pathfinder**.**visualizer\_Width **/** Program**.**pathfinder**.**map\_Width**;**

node\_Height **=** Program**.**pathfinder**.**visualizer\_Height **/** Program**.**pathfinder**.**map\_Height**;**

// Goes through each node in the node list and paints it according to its // properties

**while** **(**node\_Iterator**.**hasNext**())** **{**

Node next\_Node **=** node\_Iterator**.**next**();**

temp **=** next\_Node**;**

**if** **(**next\_Node**.**is\_Start\_Node**())** **{**

g**.**setColor**(**Color**.**GREEN**);**

g**.**fillOval**(**temp**.**get\_CoOrdinate\_Array**()[**0**]** **\*** node\_Width**,** temp**.**get\_CoOrdinate\_Array**()[**1**]** **\*** node\_Height**,**

node\_Width**,** node\_Height**);**

**}** **else** **if** **(**next\_Node**.**is\_End\_Node**())** **{**

g**.**setColor**(**Color**.**RED**);**

g**.**fillOval**(**temp**.**get\_CoOrdinate\_Array**()[**0**]** **\*** node\_Width**,** temp**.**get\_CoOrdinate\_Array**()[**1**]** **\*** node\_Height**,**

node\_Width**,** node\_Height**);**

**}** **else** **if** **(**next\_Node**.**is\_Wall\_Node**())** **{**

g**.**setColor**(**Color**.**ORANGE**);**

g**.**fillOval**(**temp**.**get\_CoOrdinate\_Array**()[**0**]** **\*** node\_Width**,** temp**.**get\_CoOrdinate\_Array**()[**1**]** **\*** node\_Height**,**

node\_Width**,** node\_Height**);**

**}** **else** **if** **(**next\_Node**.**is\_In\_Path**())** **{**

g**.**setColor**(**Color**.**CYAN**);**

g**.**fillOval**(**temp**.**get\_CoOrdinate\_Array**()[**0**]** **\*** node\_Width**,** temp**.**get\_CoOrdinate\_Array**()[**1**]** **\*** node\_Height**,**

node\_Width**,** node\_Height**);**

**}** **else** **{**

g**.**setColor**(**Color**.**GRAY**);**

g**.**fillOval**(**temp**.**get\_CoOrdinate\_Array**()[**0**]** **\*** node\_Width**,** temp**.**get\_CoOrdinate\_Array**()[**1**]** **\*** node\_Height**,**

node\_Width**,** node\_Height**);**

**}**

**}**

**}**

**}**

## Main

#### Manhattan\_Pathfinder

package main**;**

**import** graphics**.**Graphical\_User\_Interface**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

public class Manhattan\_Pathfinder **{**

// The objects which contain the G.U.I., the current state of the map and

// other information to be communicated publicly such as whether the path

// has been found

public Map map**;**

public Graphical\_User\_Interface graphical\_User\_Interface**;**

public int visualizer\_Width**,** visualizer\_Height**,** map\_Width**,** map\_Height**;**

public boolean path\_Found **=** **false,** is\_Impossible **=** **false;**

public List**<**Node**>** closed\_List**,** open\_List**,** path\_List**;**

// The method that initialises the map and the G.U.I.

public Manhattan\_Pathfinder**(**int map\_Width**,** int map\_Height**,**

int visualizer\_Width**,** int visualizer\_Height**)** **{**

**this.**visualizer\_Width **=** visualizer\_Width**;**

**this.**visualizer\_Height **=** visualizer\_Height**;**

**this.**map\_Width **=** map\_Width**;**

**this.**map\_Height **=** map\_Height**;**

create\_Map**();**

**if** **(**map**.**is\_Initialized**)** **{**

graphical\_User\_Interface **=** **new** Graphical\_User\_Interface**(**map\_Width**,** map\_Height**,** visualizer\_Width**,**

visualizer\_Height**);**

**}**

**}**

// The method that creates the first map based on the dimensions set in the

// 'Program' class

public void create\_Map**()** **{**

map **=** **new** Map**(**map\_Width**,** map\_Height**,** visualizer\_Width**,** visualizer\_Height**);**

**}**

// A method to create a map with dimensions given as parameters

public void create\_Map**(**int map\_Width**,** int map\_Height**)** **{**

map **=** **new** Map**(**map\_Width**,** map\_Height**,** visualizer\_Width**,** visualizer\_Height**);**

**}**

// A recursive method to calculate the G cost of a Node 'node' and return it

public int calculate\_G\_Cost**(**Node node**)** **{**

int g\_Cost **=** 0**;**

**if** **(!**node**.**is\_Start\_Node**())** **{**

g\_Cost**++;**

node **=** node**.**get\_Parent\_Node**();**

g\_Cost **=** calculate\_G\_Cost**(**node**,** g\_Cost**);**

**}**

**return** g\_Cost**;**

**}**

// A recursive method called by the method with the same name that takes

// only a Node parameter which calculates the G cost of the given Node

public int calculate\_G\_Cost**(**Node node**,** int g\_Cost**)** **{**

**if** **(!**node**.**is\_Start\_Node**())** **{**

g\_Cost**++;**

node **=** node**.**get\_Parent\_Node**();**

g\_Cost **=** calculate\_G\_Cost**(**node**,** g\_Cost**);**

**}**

**return** g\_Cost**;**

**}**

// A method to calculate the H cost for a Node by calculating the absolute

// distance between it and the end Node

public int calculate\_H\_Cost**(**Node node**,** List**<**Node**>** node\_List**)** **{**

Node end\_Node **=** **null;**

**for** **(**Node node\_Iterator **:** node\_List**)** **{**

**if** **(**node\_Iterator**.**is\_End\_Node**())** **{**

end\_Node **=** node\_Iterator**;**

**}**

**}**

int**[]** node\_CoOrdinate\_Array **=** node**.**get\_CoOrdinate\_Array**();**

int**[]** end\_Node\_CoOrdinate\_Array **=** end\_Node**.**get\_CoOrdinate\_Array**();**

int h\_Cost **=** Math**.**abs**(**end\_Node\_CoOrdinate\_Array**[**0**]**

node\_CoOrdinate\_Array**[**0**]);**

h\_Cost **=** h\_Cost **+** Math**.**abs**(**end\_Node\_CoOrdinate\_Array**[**1**]** node\_CoOrdinate\_Array**[**1**]);**

**return** h\_Cost**;**

**}**

// A method to calculate the F cost of a Node

public int calculate\_F\_Cost**(**Node node**)** **{**

**return** node**.**get\_H\_Cost**()** **+** node**.**get\_G\_Cost**();**

**}**

// A method to calculate the F cost when given a G cost and M cost

public int calculate\_F\_Cost**(**int g\_Cost**,** int h\_Cost**)** **{**

**return** g\_Cost **+** h\_Cost**;**

**}**

// The recursive method called once a path has been found, which works

// backwards from

// the end node by going from child to parent Node and adding each Node

// along the way to the publicly visible 'path\_List' list and updates the

// Nodes to reflect that they're part of the path

public void get\_Path**(**Node node**)** **{**

List**<**Node**>** path\_List **=** **new** ArrayList**<**Node**>();**

**if** **(!**node**.**is\_Start\_Node**())** **{**

path\_List**.**add**(**node**.**get\_Parent\_Node**());**

node **=** node**.**get\_Parent\_Node**();**

path\_List**.**addAll**(**get\_Path\_Without\_Draw**(**node**));**

**}**

**for** **(**Node node\_Iterator **:** path\_List**)** **{**

node\_Iterator**.**set\_In\_Path**();**

**}**

**this.**path\_List **=** path\_List**;**

**}**

// The recursive method called by 'get\_Path' which performs the same

// function, but without updating the publicly available 'path\_List' list

// and without updating the individual Nodes to reflect their part in the

// path

private List**<**Node**>** get\_Path\_Without\_Draw**(**Node node**)** **{**

List**<**Node**>** path\_List **=** **new** ArrayList**<**Node**>();**

**if** **(!**node**.**is\_Start\_Node**())** **{**

path\_List**.**add**(**node**.**get\_Parent\_Node**());**

node **=** node**.**get\_Parent\_Node**();**

path\_List**.**addAll**(**get\_Path\_Without\_Draw**(**node**));**

**}**

**return** path\_List**;**

**}**

// The method which use the Manhattan algorithm to find a path from the

// start node to the end node

public void find\_Path**(**List**<**Node**>** node\_List**)** **{**

// Creating the lists that sort nodes to show whether they have been

// used as part of a path (the closed list) or if they can be moved to

// (the open list)

List**<**Node**>** open\_List **=** **new** ArrayList**<**Node**>();**

List**<**Node**>** closed\_List **=** **new** ArrayList**<**Node**>();**

// The node objects that store the end node, the node that is the

// current position of the algorithm and the node in the open list with

// the lowest F cost respectively

Node end\_Node **=** **null;**

Node current\_Node **=** **null;**

Node lowest\_Cost\_Node **=** **null;**

// A for-each loop to find the start and end node and assign them to the

// 'current\_Node' and 'end\_node' variables respectively

**for** **(**Node node\_Iterator **:** node\_List**)** **{**

**if** **(**node\_Iterator**.**is\_Start\_Node**())** **{**

current\_Node **=** node\_Iterator**;**

**}**

**if** **(**node\_Iterator**.**is\_End\_Node**())** **{**

end\_Node **=** node\_Iterator**;**

**}**

**}**

open\_List**.**add**(**current\_Node**);**

// The loop which runs while the algorithm has not run out of Nodes to

// move to and a path has not been found

**while** **(!**Program**.**does\_Contain**(**closed\_List**,** end\_Node**)** **&&** **!**open\_List**.**isEmpty**())** **{**

lowest\_Cost\_Node **=** **null;**

// A for-each loop which finds the Node with the lowest F cost in

// the open list and assigns it to 'lowest\_Cost\_Node'

**for** **(**Node node\_Iterator **:** open\_List**)** **{**

**if** **(**lowest\_Cost\_Node **==** **null)** **{**

lowest\_Cost\_Node **=** node\_Iterator**;**

**}** **else** **{**

node\_Iterator**.**set\_F\_Cost**(**calculate\_F\_Cost

**(**calculate\_G\_Cost**(**nodeIterator**),**

calculate\_H\_Cost**(**node\_Iterator**,** node\_List**)));**

**if** **(**node\_Iterator**.**get\_F\_Cost**()** **<** lowest\_Cost\_Node**.**get\_F\_Cost**()){**

lowest\_Cost\_Node **=** node\_Iterator**;**

**}**

**}**

**}**

// Makes the lowest cost node the new 'current\_Node' and switches it

// to the closed list

current\_Node **=** lowest\_Cost\_Node**;**

closed\_List**.**add**(**current\_Node**);**

open\_List**.**remove**(**current\_Node**);**

// A for-each loop which finds all the traversable adjacent nodes to

// the current node and if they have not been traversed before adds

// them to the open list and set the current node as their parent,

// if they have then the F cost via this path is compared to the F

// cost via the previous path and if the new F cost is lower the

// current node is made the node's new parent without switching them

// to the open list

**for** **(**Node node\_Iterator **:** current\_Node**.**get\_Adjacent\_Nodes**())** **{**

**if** **(!**node\_Iterator**.**is\_Wall\_Node**()** **&&** **!**Program**.**does\_Contain**(**closed\_List**,** node\_Iterator**))** **{**

**if** **(!**Program**.**does\_Contain**(**open\_List**,** node\_Iterator**))** **{**

open\_List**.**add**(**node\_Iterator**);**

node\_Iterator**.**set\_Parent\_Node**(**current\_Node**);**

**}** **else** **{**

int new\_F\_Cost **=** calculate\_F\_Cost**(**calculate\_G\_Cost**(**node\_Iterator**),**

calculate\_H\_Cost**(**node\_Iterator**,** node\_List**));**

**if** **(**new\_F\_Cost **<** node\_Iterator**.**get\_F\_Cost**())** **{**

node\_Iterator**.**set\_F\_Cost**(**new\_F\_Cost**);**

node\_Iterator**.**set\_Parent\_Node**(**current\_Node**);**

**}**

**}**

**}**

**}**

**}**

// An if statement which runs when the while loop has ended - if the

// loop found a path it calls 'get\_Path' and updates the 'path\_Found'

// variable to true, however if not it updates the 'is\_Impossible'

// variable to true

**if** **(**Program**.**does\_Contain**(**closed\_List**,** end\_Node**))** **{**

get\_Path**(**end\_Node**);**

path\_Found **=** **true;**

**}** **else** **{**

is\_Impossible **=** **true;**

**}**

**}**

**}**

#### Map

package main**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Random**;**

public class Map **{**

// Variables containing information about the map and the nodes on it

public int map\_Width**,** map\_Height**;**

public List**<**Node**>** node\_List **=** **new** ArrayList**<**Node**>();**

public Node start\_Node**,** end\_Node**;**

// A boolean to show if the the map has been initialised

public boolean is\_Initialized **=** **false;**

// The constructor for the map class, creating a map based off of the

// dimensions it is passed and randomly distributing the start node, the end

// node and wall nodes

public Map**(**int map\_Width**,** int map\_Height**,** int display\_Width**,** int display\_Height**)** **{**

**this.**map\_Width **=** map\_Width**;**

**this.**map\_Height **=** map\_Height**;**

node\_List**.**clear**();**

Random random\_Number **=** **new** Random**();**

// The for loop that creates all of the nodes on the map and randomly

// allocates some of them as wall nodes

**for** **(**int y\_Iterator **=** 0**;** y\_Iterator **<** map\_Height**;** y\_Iterator**++)** **{**

**for** **(**int x\_Iterator **=** 0**;** x\_Iterator **<** map\_Width**;** x\_Iterator**++)** **{**

node\_List**.**add**(new** Node**(**x\_Iterator**,** y\_Iterator**,** random\_Number**.**nextInt**(**4**)));**

**}**

**}**

// The while loop that randomly finds two non-wall nodes and allocates

// them as the start and end node

**while** **(true)** **{**

start\_Node **=** node\_List**.**get**(**random\_Number**.**nextInt**(**map\_Width **\*** map\_Height **-** 1**));**

start\_Node**.**set\_Start\_Node**();**

end\_Node **=** node\_List**.**get**(**random\_Number**.**nextInt**(**map\_Width **\*** map\_Height **-** 1**));**

end\_Node**.**set\_End\_Node**();**

**if** **(!**start\_Node**.**is\_Wall\_Node**()** **&&** **!**end\_Node**.**is\_Wall\_Node**()**

**&&** start\_Node**.**get\_CoOrdinate\_Array**()[**0**]** **!=** end\_Node**.**get\_CoOrdinate\_Array**()[**0**]**

**&&** start\_Node**.**get\_CoOrdinate\_Array**()[**1**]** **!=** end\_Node**.**get\_CoOrdinate\_Array**()[**1**])** **{**

**break;**

**}** **else** **{**

start\_Node**.**set\_Not\_Start\_Node**();**

end\_Node**.**set\_Not\_End\_Node**();**

**}**

**}**

is\_Initialized **=** **true;**

**}**

// A public method to get the list of nodes on the map

public List**<**Node**>** get\_Node\_List**()** **{**

**return** node\_List**;**

**}**

**}**

#### Node

package main**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

public class Node **{**

// Variables containing the properties of a node

private boolean is\_Start\_Node**,** is\_End\_Node**,** is\_Wall\_Node**,** is\_In\_Path**;**

private Node parent\_Node**;**

private int node\_H\_Cost**,** node\_G\_Cost**,** node\_F\_Cost**,** node\_X\_CoOrdinate**,** node\_Y\_CoOrdinate**;**

// The constructor for the node class, that creates a node with given x and

// y coordinates along with an integer that dictates whether it is a wall

// node

public Node**(**int x\_CoOrdinate**,** int y\_CoOrdinate**,** int is\_Wall\_Node**)** **{**

node\_X\_CoOrdinate **=** x\_CoOrdinate**;**

node\_Y\_CoOrdinate **=** y\_CoOrdinate**;**

**if** **(**is\_Wall\_Node **==** 1**)** **{**

**this.**is\_Wall\_Node **=** **true;**

**}** **else** **{**

**this.**is\_Wall\_Node **=** **false;**

**}**

**}**

// Setter for the parent node variable

public void set\_Parent\_Node**(**Node parent\_Node**)** **{**

**this.**parent\_Node **=** parent\_Node**;**

**}**

// Getter for the parent node variable

public Node get\_Parent\_Node**()** **{**

**return** parent\_Node**;**

**}**

// A method to check if the node is the start node

public boolean is\_Start\_Node**()** **{**

**if** **(**is\_Start\_Node**)** **{**

**return** **true;**

**}** **else** **{**

**return** **false;**

**}**

**}**

// A method to check if the node is the end node

public boolean is\_End\_Node**()** **{**

**if** **(**is\_End\_Node**)** **{**

**return** **true;**

**}** **else** **{**

**return** **false;**

**}**

**}**

// A method to check if the node is a wall node

public boolean is\_Wall\_Node**()** **{**

**if** **(**is\_Wall\_Node**)** **{**

**return** **true;**

**}** **else** **{**

**return** **false;**

**}**

**}**

// A method to check if the node is in the path

public boolean is\_In\_Path**()** **{**

**if** **(**is\_In\_Path**)** **{**

**return** **true;**

**}** **else** **{**

**return** **false;**

**}**

**}**

// Setter for the H Cost variable

public void set\_H\_Cost**(**int h\_Cost**)** **{**

node\_H\_Cost **=** h\_Cost**;**

**}**

// Getter for the H Cost variable

public int get\_H\_Cost**()** **{**

**return** node\_H\_Cost**;**

**}**

// Setter for the F Cost variable

public void set\_F\_Cost**(**int f\_Cost**)** **{**

node\_F\_Cost **=** f\_Cost**;**

**}**

// Getter for the F Cost variable

public int get\_F\_Cost**()** **{**

**return** node\_F\_Cost**;**

**}**

// Setter for the G Cost variable

public void set\_G\_Cost**(**int g\_Cost**)** **{**

node\_G\_Cost **=** g\_Cost**;**

**}**

// Getter for the G Cost variable

public int get\_G\_Cost**()** **{**

**return** node\_G\_Cost**;**

**}**

// Gets the Coordinates of the node

public int**[]** get\_CoOrdinate\_Array**()** **{**

int**[]** coOrdinate\_Array **=** **new** int**[**2**];**

coOrdinate\_Array**[**0**]** **=** node\_X\_CoOrdinate**;**

coOrdinate\_Array**[**1**]** **=** node\_Y\_CoOrdinate**;**

**return** coOrdinate\_Array**;**

**}**

// A method to set the node as the start node

public void set\_Start\_Node**()** **{**

is\_Start\_Node **=** **true;**

**}**

// A method to set the node as a normal node

public void set\_Not\_Start\_Node**()** **{**

is\_Start\_Node **=** **false;**

**}**

// A method to set the node as the end node

public void set\_End\_Node**()** **{**

is\_End\_Node **=** **true;**

**}**

// A method to set the node as a normal node

public void set\_Not\_End\_Node**()** **{**

is\_End\_Node **=** **false;**

**}**

// A method to set the node in the path

public void set\_In\_Path**()** **{**

is\_In\_Path **=** **true;**

**}**

// A method to remove the node from the path

public void set\_Not\_In\_Path**()** **{**

is\_In\_Path **=** **false;**

**}**

// A method to return a list of all the nodes adjacent to the one it is run

// on

public List**<**Node**>** get\_Adjacent\_Nodes**()** **{**

List**<**Node**>** adjacent\_Nodes **=** **new** ArrayList**<**Node**>();**

Map map **=** Program**.**pathfinder**.**map**;**

List**<**Node**>** node\_List **=** map**.**get\_Node\_List**();**

**for** **(**int x\_Iterator **=** node\_X\_CoOrdinate **-** 1**;** x\_Iterator **<=** node\_X\_CoOrdinate **+** 1**;** x\_Iterator**++)** **{**

**for** **(**int y\_Iterator **=** node\_Y\_CoOrdinate **-** 1**;** y\_Iterator **<=** node\_Y\_CoOrdinate **+** 1**;** y\_Iterator**++)** **{**

**if** **(**x\_Iterator **==** node\_X\_CoOrdinate **&&** y\_Iterator **==** node\_Y\_CoOrdinate**)** **{**

**continue;**

**}**

**if** **(**x\_Iterator **!=** node\_X\_CoOrdinate **&&** y\_Iterator **!=** node\_Y\_CoOrdinate**)** **{**

**continue;**

**}**

**if** **(**x\_Iterator **>=** map**.**map\_Width **||** y\_Iterator **>=** map**.**map\_Height**)** **{**

**continue;**

**}** **else** **if** **(**x\_Iterator **<** 0 **||** y\_Iterator **<** 0**)** **{**

**continue;**

**}**

adjacent\_Nodes**.**add**(**node\_List**.**get**(**y\_Iterator **\*** map**.**map\_Width **+** x\_Iterator**));**

**}**

**}**

**return** adjacent\_Nodes**;**

**}**

// A method to return information about a node as a string - used in testing

@Override

public String toString**()** **{**

**return** "[" **+** node\_X\_CoOrdinate **+** ", " **+** node\_Y\_CoOrdinate **+** ", " **+** is\_Wall\_Node **+** "]"**;**

**}**

**}**

#### Program

package main**;**

**import** java**.**util**.**List**;**

public class Program **{**

// The path-finder object to be used throughout the program

public static Manhattan\_Pathfinder pathfinder**;**

public static void main**(**String**[]** args**)** **{**

// Initialises the path-finder

pathfinder **=** **new** Manhattan\_Pathfinder**(**20**,** 20**,** 1270**,** 550**);**

pathfinder**.**graphical\_User\_Interface**.**run**();**

**}**

// A method to search a list 'list' to see if it contains a node 'node'

public static boolean does\_Contain**(**List**<**Node**>** list**,** Node node**)** **{**

int does\_Contain\_Node **=** 0**;**

// Runs through each node in the list and checks if they are equal to

// the node being looked for, and if they are it increments a variable

**for** **(**Node node\_Iterator **:** list**)** **{**

**if** **(**node\_Iterator**.**equals**(**node**))** **{**

does\_Contain\_Node**++;**

**}**

**}**

// If the variable is greater than or equal to one it means that the

// list contains the node and so the method returns true

**if** **(**does\_Contain\_Node **>=** 1**)** **{**

**return** **true;**

**}** **else** **{**

**return** **false;**

**}**

**}**

**}**

# Testing

## Black Box Testing

#### Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Series | Description of Test | Test Data | Expected Result | Actual Result | Evidence |
| 1.1 | Check that when the ‘Start Pathfinder’ button is pressed for the default 20 by 20 map size and there is a valid path available, the program finds a valid path from the start node to the end node and displays it. | Map Width = 20  Map Height  = 20 | A path from the start node to the end node will be displayed in blue on the visualizer and the title will be changed to include ‘PATH FOUND’. | A path from the start to the end node is displayed in blue on the visualizer and the title is changed to include ‘PATH FOUND’. | Before:  Figure 1.1  After:  Figure 1.2 |
| 1.2 | Check that when the ‘Start Pathfinder’ button is pressed for the default 20 by 20 map size and there is no valid path available, the program displays that the map is impossible. | Map Width = 20  Map Height  = 20 | The title will be changed to include ‘IMPOSSIBLE’, because the start node is surrounded by walls. | The title is changed to include ‘IMPOSSIBLE’. | Before:  Figure 1.3  After:  Figure 1.4 |
| 1.3 | Check that the ‘Refresh’ button clears the displayed path. | Map Width = 20  Map Height = 20 | The title will be changed to no longer include ‘PATH FOUND’ and the path will no longer be displayed on the visualizer. | The title is changed to no longer include ‘PATH FOUND’ and the blue path is no longer displayed. | Before:  Figure 1.2  After:  Figure 1.5 |
| 1.4 | Check that the ‘Edit Mode’ button changes the G.U.I. to edit mode. | Map Width = 20  Map Height = 20 | The ‘Start Pathfinder’, ‘Refresh’ and ‘Edit Mode’ buttons will become unavailable to the user and the ‘Increase Map Dimensions’, ‘Decrease Map Dimensions’, ‘Randomize the Map’ and ‘Simulation Mode’ buttons will become available instead. | The ‘Start Pathfinder’, ‘Refresh’ and ‘Edit Mode’ buttons become unavailable to the user and the ‘Increase Map Dimensions’, ‘Decrease Map Dimensions’, ‘Randomize the Map’ and ‘Simulation Mode’ buttons become available instead. | Before:  Figure 1.5  After:  Figure 1.6 |
| 1.5 | Check that the ‘Increase Map Dimensions’ button randomly generates a new, larger, map. | Initial Map Width = 20  Initial Map Height = 20 | The visualizer will display a new, random, 25 by 25 map, and update the title to ‘Width: 25, Height: 25’ | The visualizer displays a new, random, 25 by 25 map, and updates the title to ‘Width: 25, Height: 25’. | Before:  Figure 1.6  After:  Figure 1.7 |
| 1.6 | Check that the ‘Decrease Map Dimensions’ button randomly generates a new, smaller, map. | Initial Map Width = 25  Initial Map Height = 25 | The visualizer will display a new, random, 20 by 20 map, and update the title to ‘Width: 20, Height: 20’ | The visualizer displays a new, random, 20 by 20 map, and updates the title to ‘Width: 20, Height: 20’. | Before:  Figure 1.7  After:  Figure 1.8 |
| 1.7 | Check that the ‘Randomize the Map’ button randomly generates a new map of the same dimensions as the current map. | Map Width = 20  Map Height = 20 | The visualizer will display a new, random, 20 by 20 map | The visualizer displays a new, random 20 by 20 map | Before: Figure 1.8 After: Figure 1.9 |
| 1.8 | Check that the ‘Simulation Mode’ button changes the G.U.I. to simulation mode. | Map Width = 20  Map Height = 20 | The ‘Increase Map Dimensions’, ‘Decrease Map Dimensions’, ‘Randomize the Map’ and ‘Simulation Mode’ buttons will become unavailable to the user and the ‘Start Pathfinder’, ‘Refresh’ and ‘Edit Mode’ buttons will become available instead. | The ‘Increase Map Dimensions’, ‘Decrease Map Dimensions’, ‘Randomize the Map’ and ‘Simulation Mode’ buttons become unavailable to the user and the ‘Start Pathfinder’, ‘Refresh’ and ‘Edit Mode’ buttons become available instead. | Before:  Figure 1.9  After:  Figure 1.10 |

#### Evidence

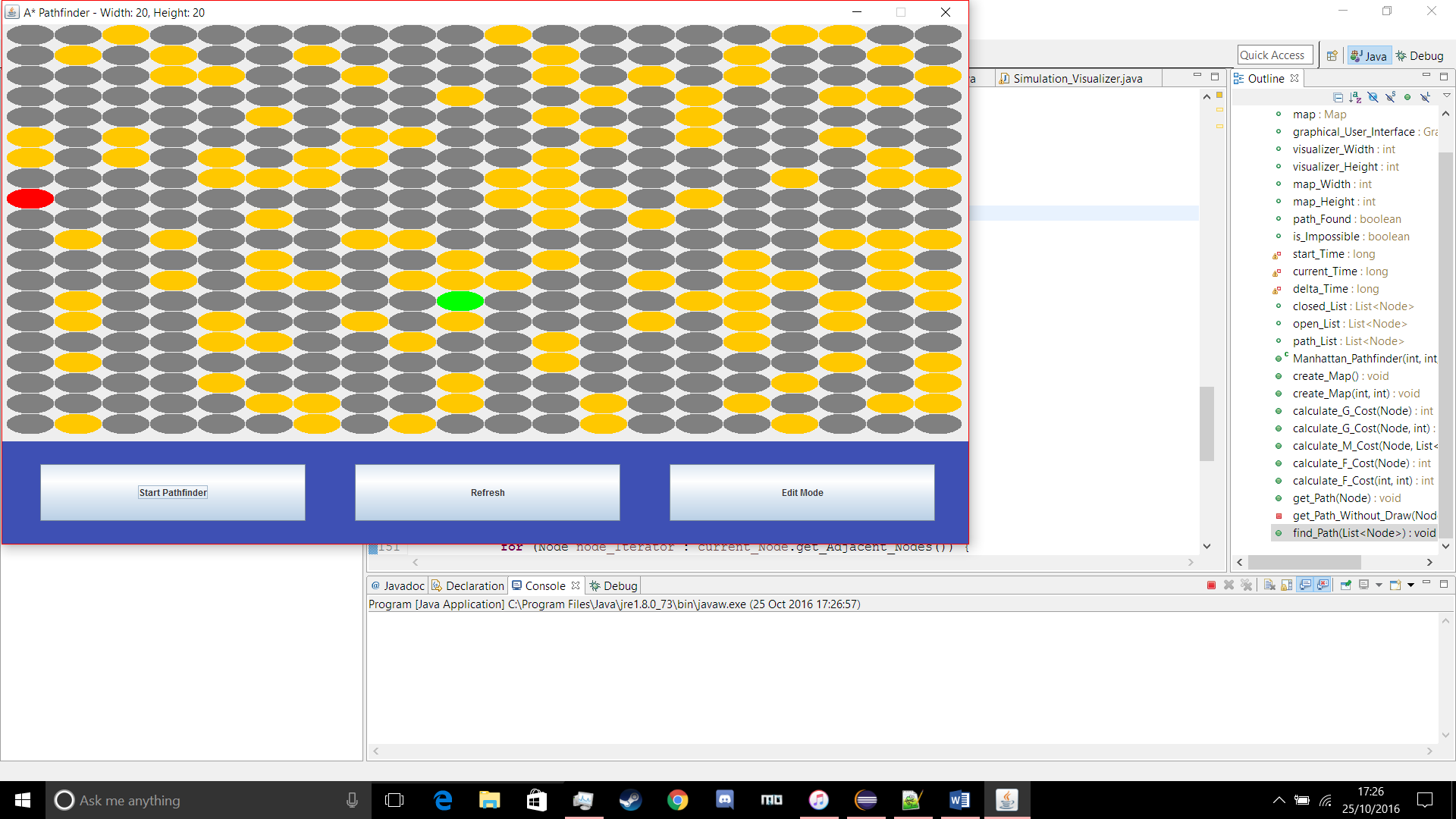


Figure 1.1

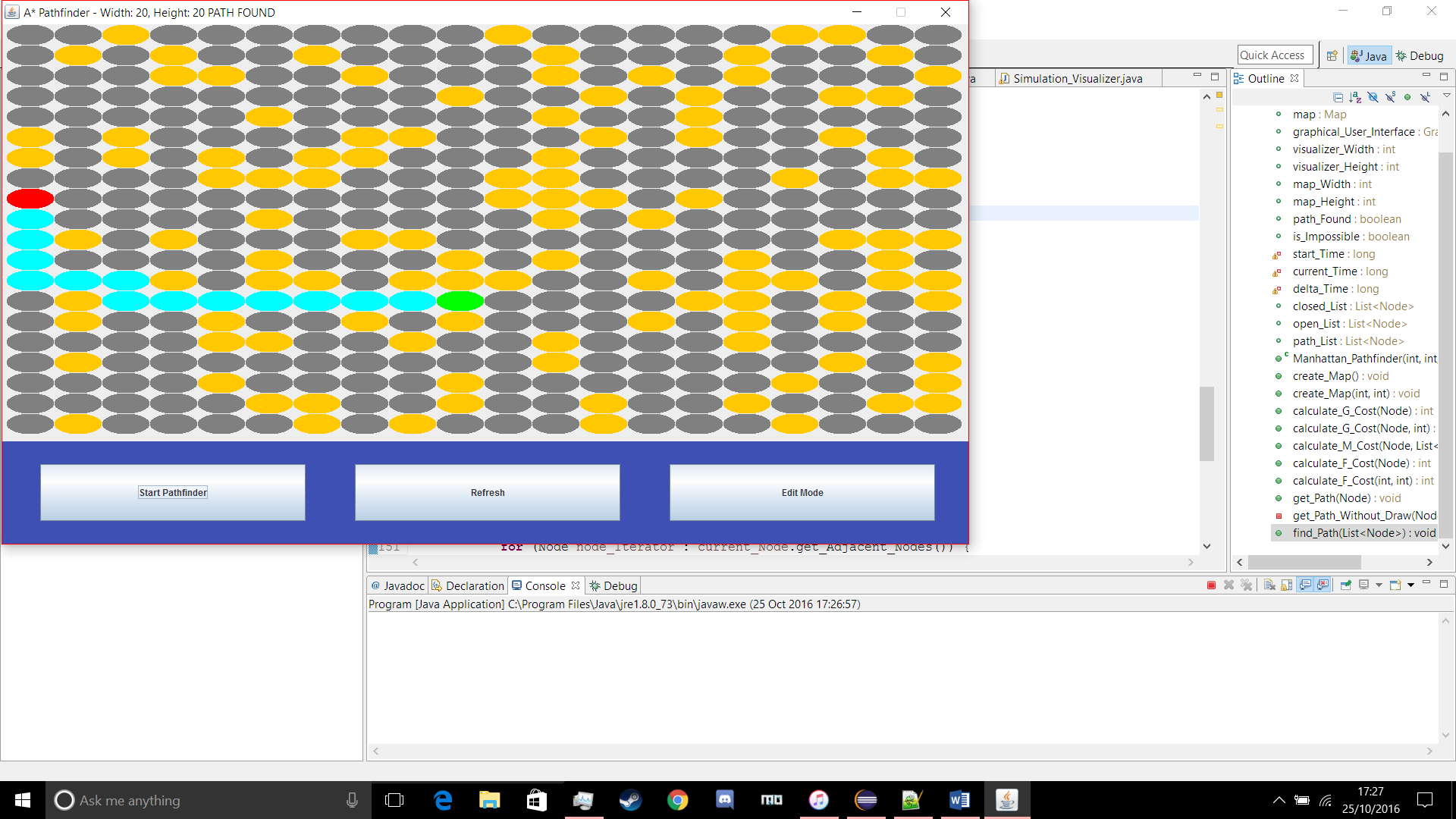


Figure 1.2

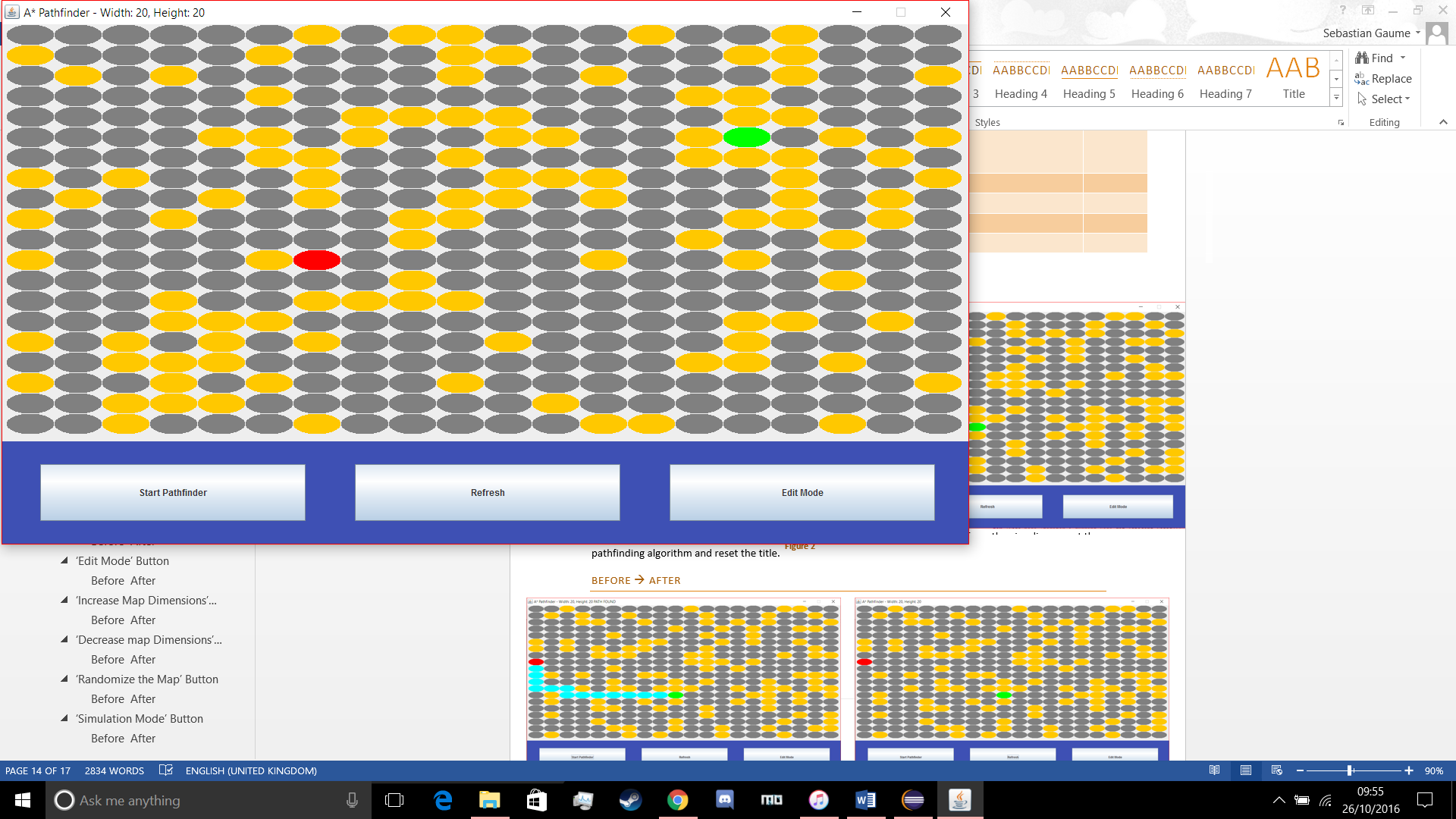


Figure 1.3

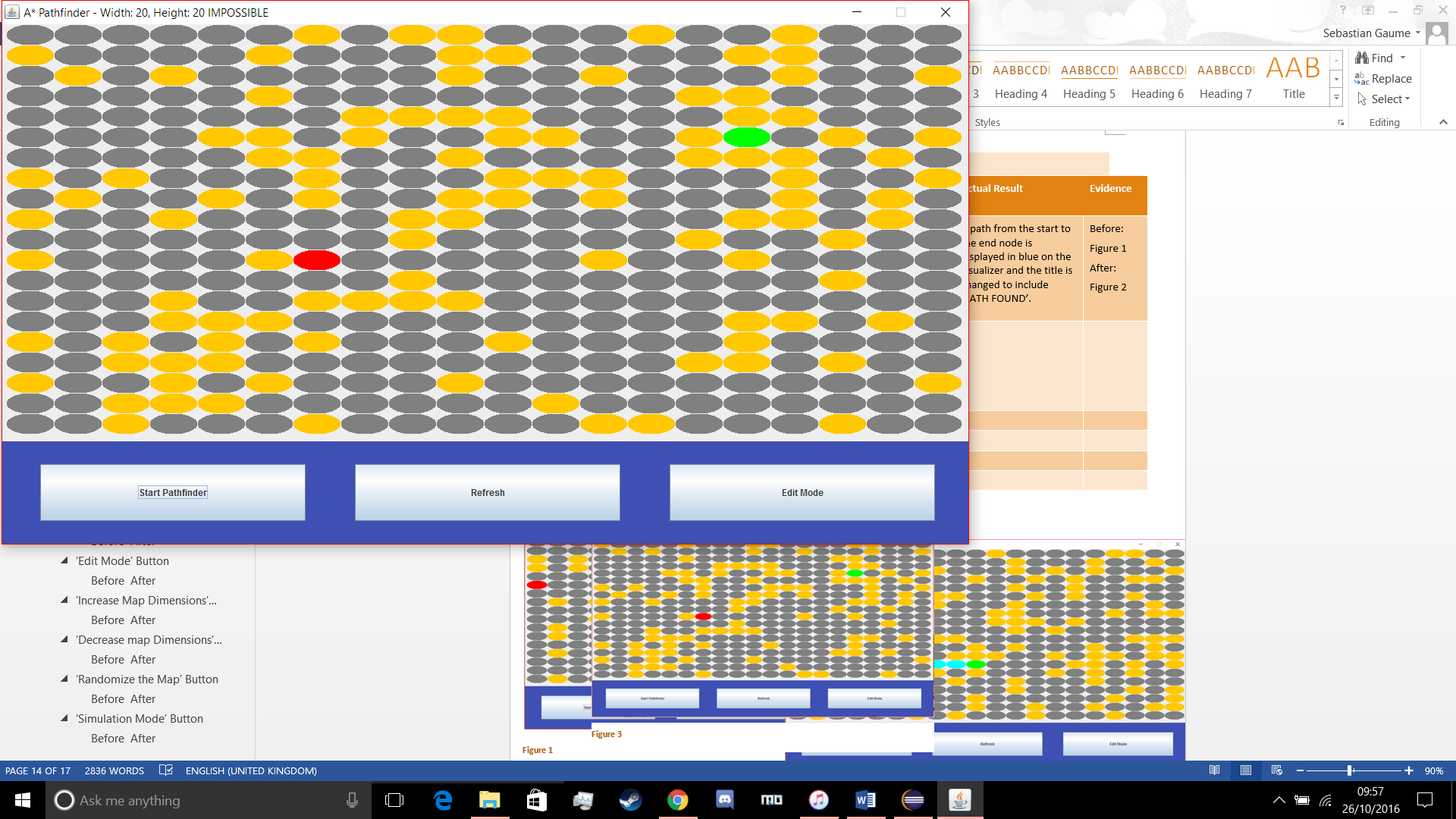


Figure 1.4

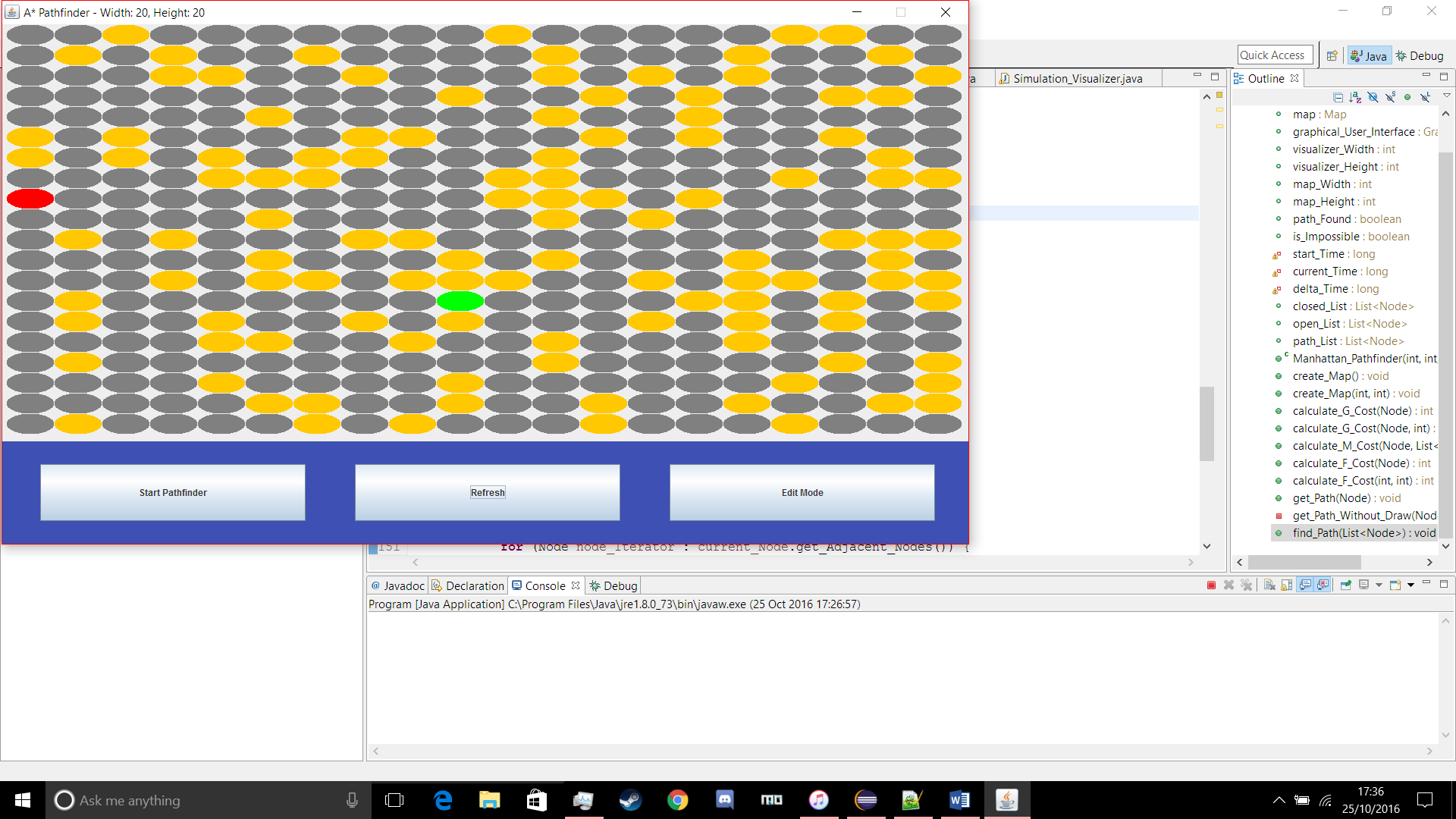


Figure 1.5

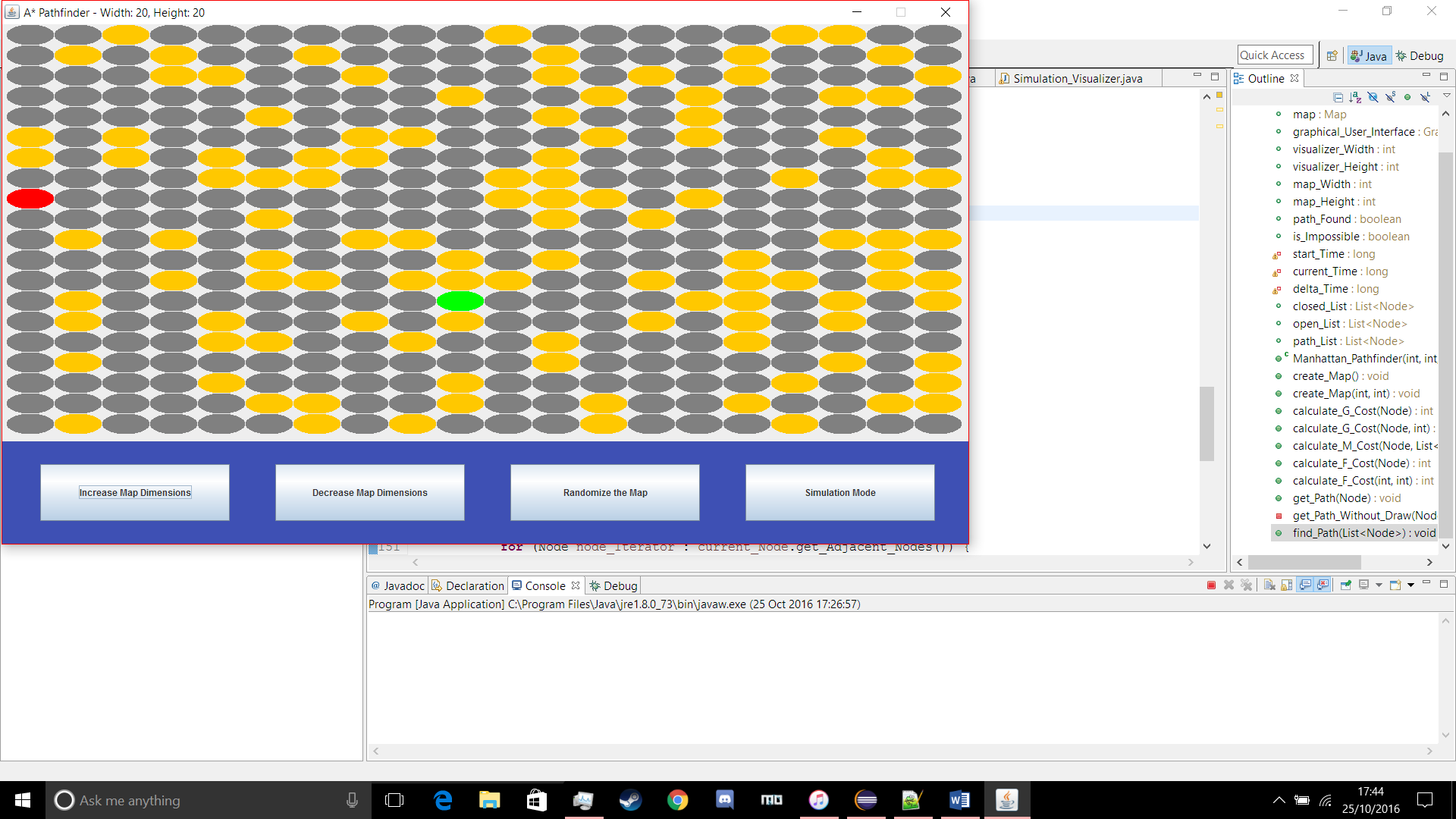


Figure 1.6

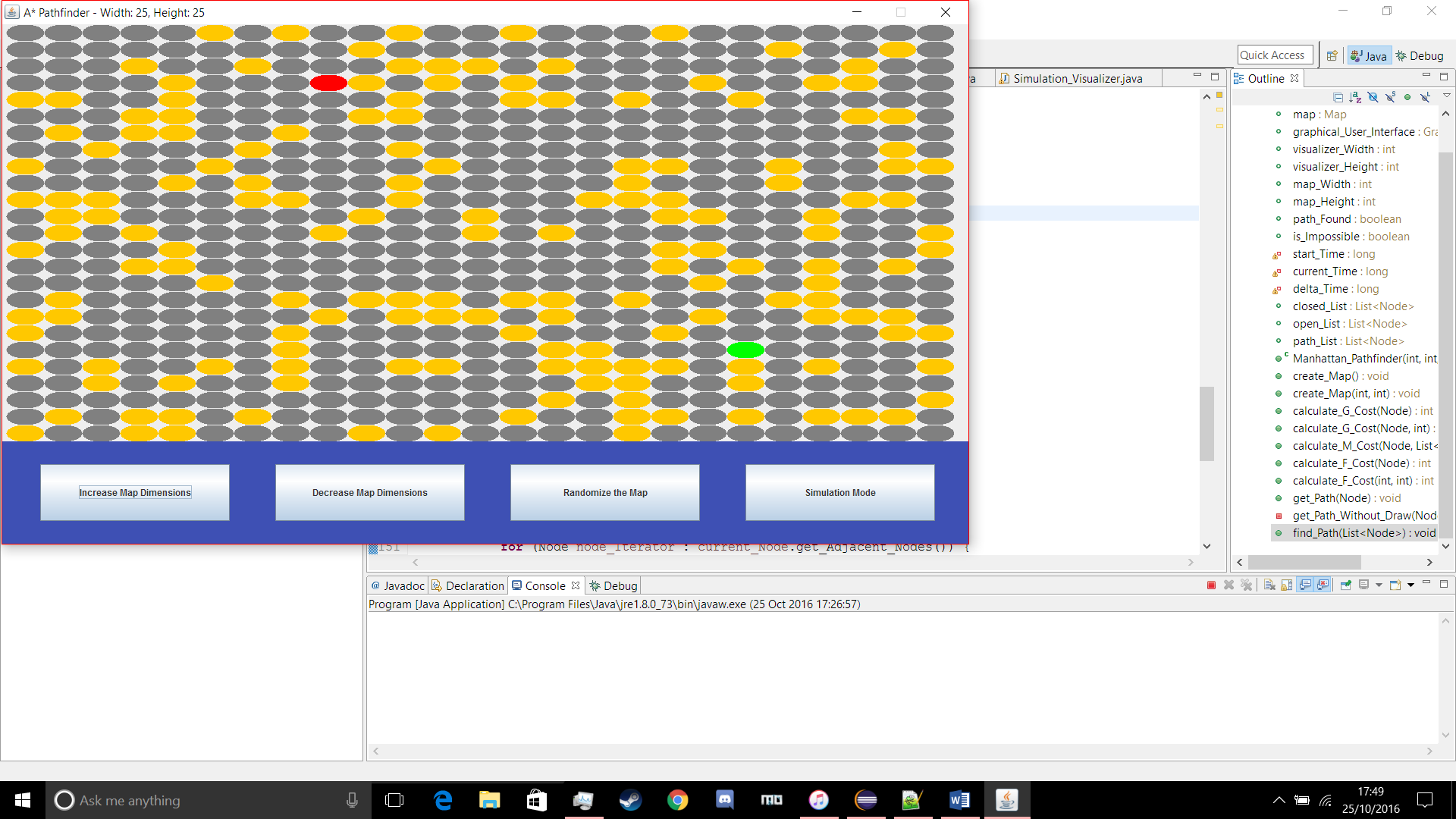


Figure 1.7

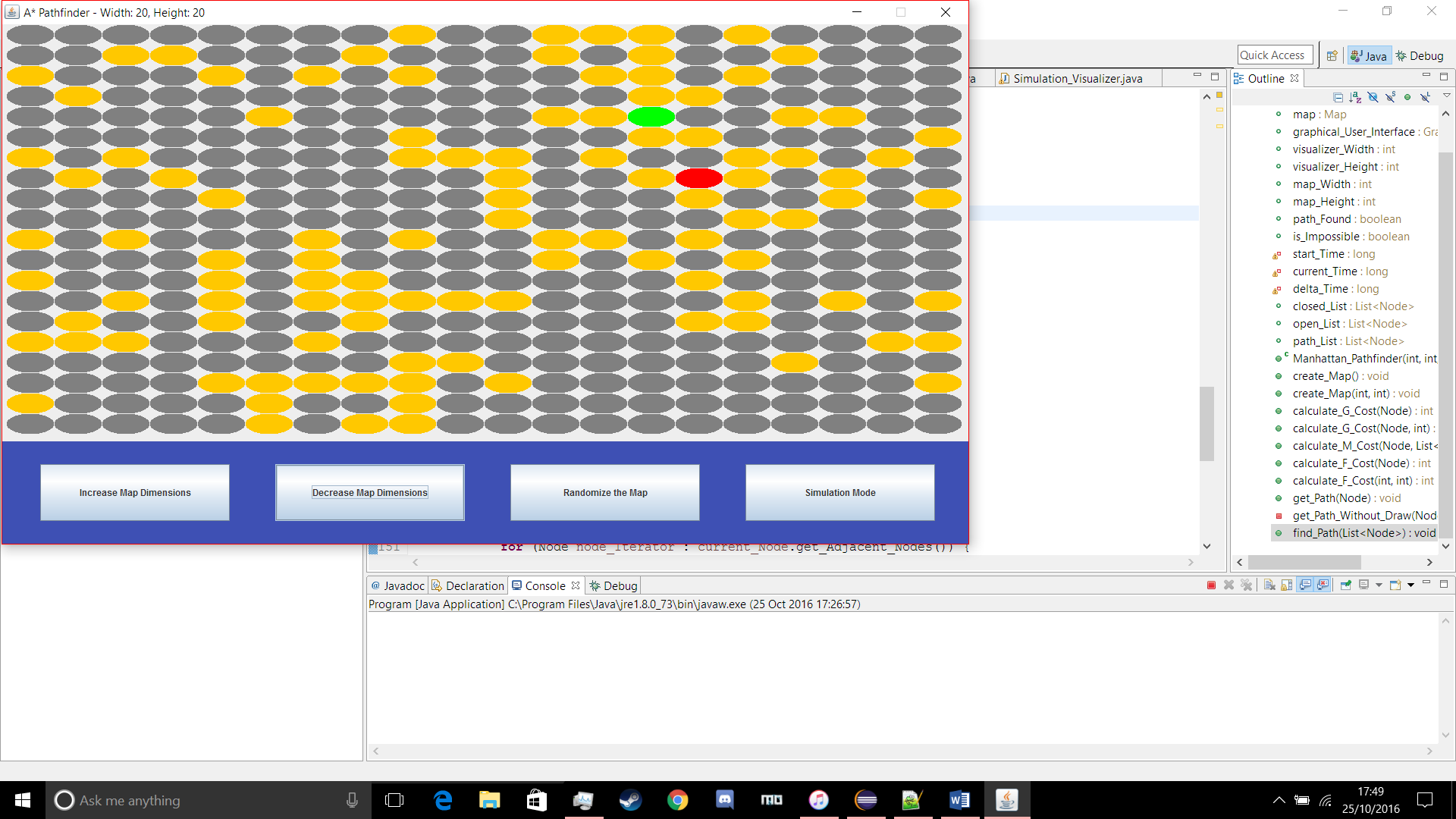


Figure1. 8

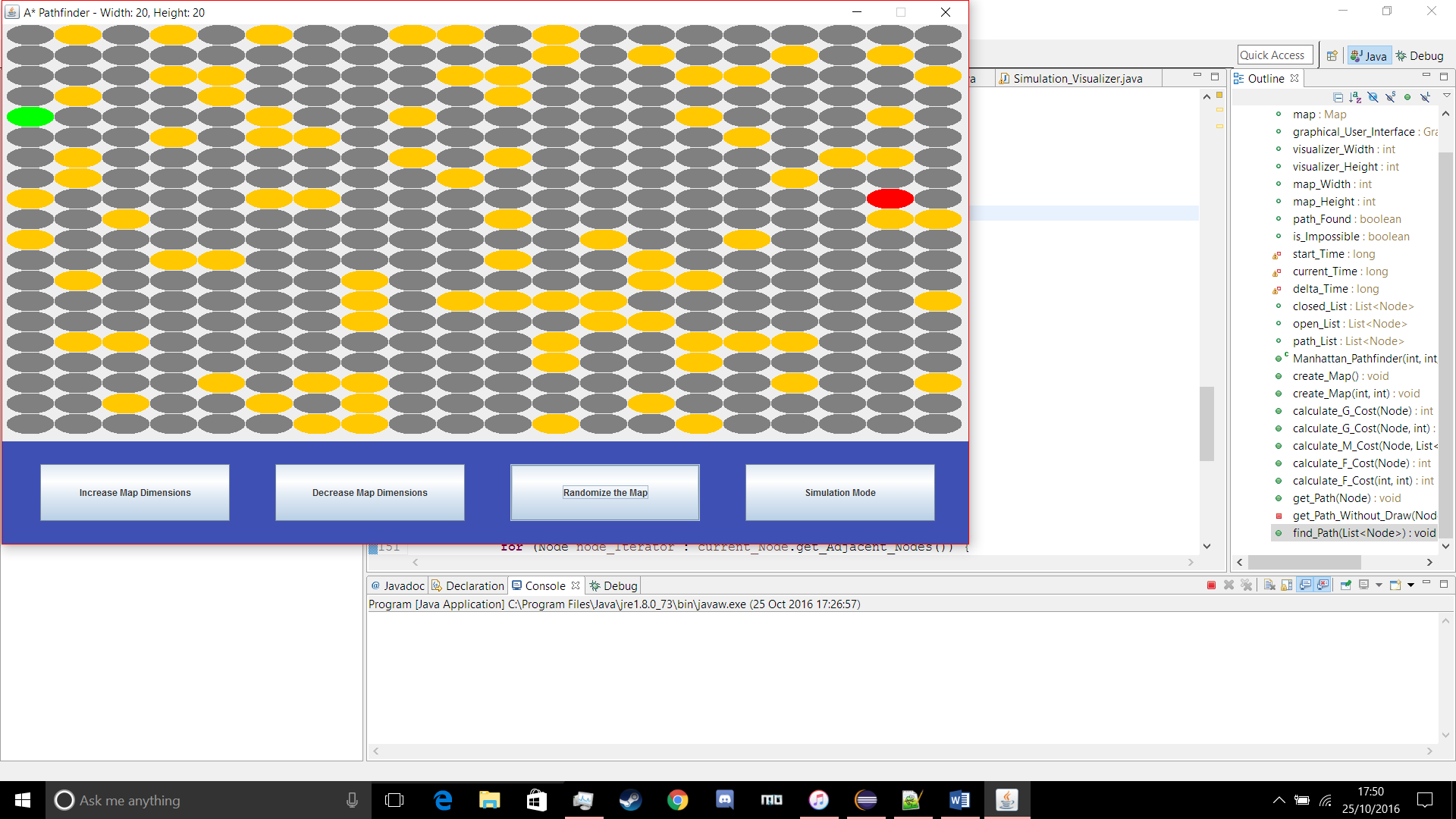


Figure 1.9

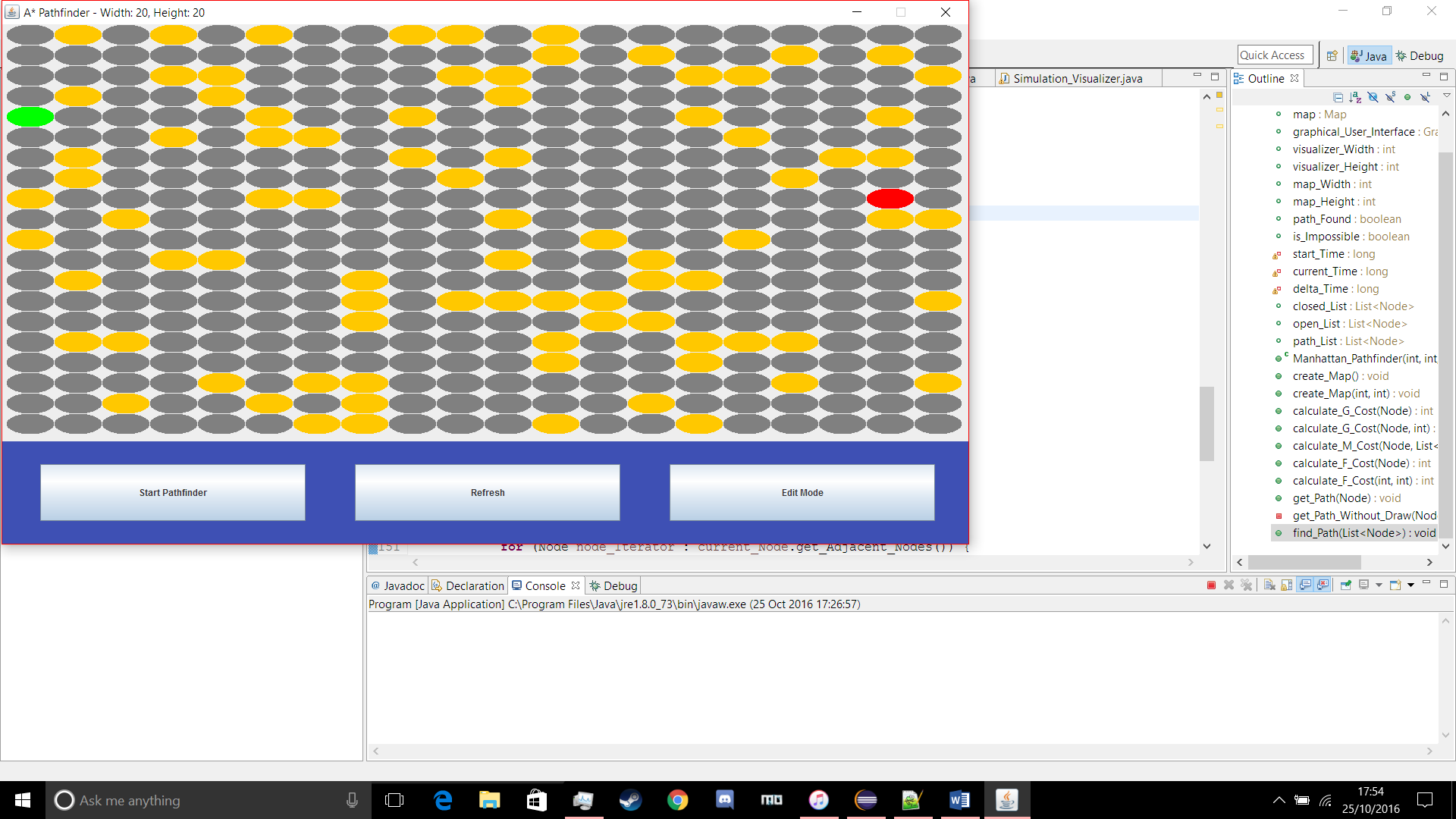
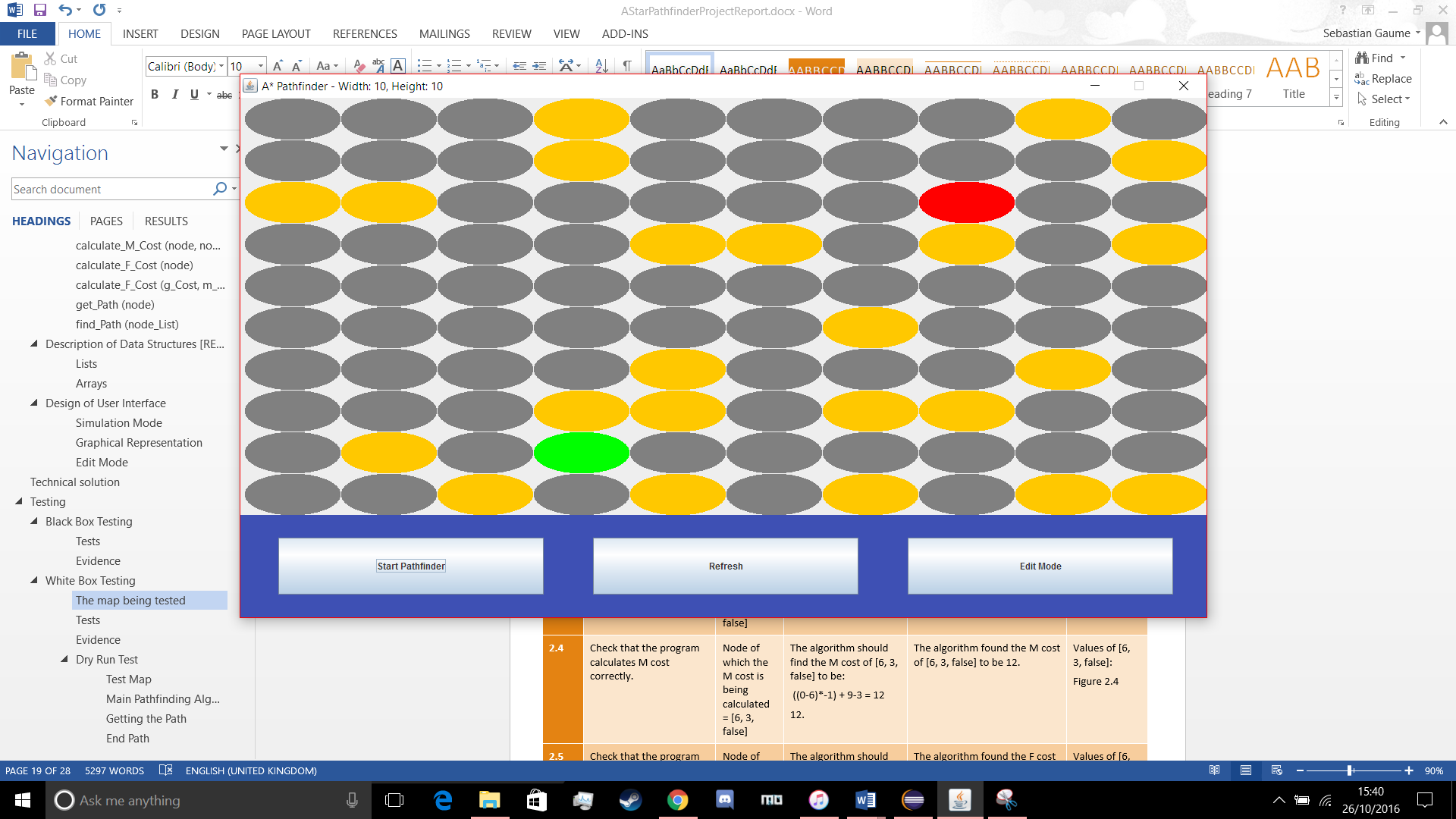


Figure 1.10

## White Box Testing

#### The map being tested



#### Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Series | Description of Test | Test Data | Expected Result | Actual Result | Evidence |
| 2.1 | Check that the Start and End Nodes are found correctly, and the Start Node becomes the Current Node, when the algorithm starts. | Start Node = [3, 8, false]  End Node = [7, 2, false] | current\_Node.toString() should output [3, 8, false] and end\_Node.toString() should output [7, 2, false]. | current\_Node.toString() outputs [3, 8, false] and end\_Node.toString() outputs [7, 2, false]. | current\_Node.toString():  Figure 2.1  end\_Node.toString():  Figure 2.2 |
| 2.2 | Check that the program finds the Adjacent Nodes correctly. | Current Node = [3, 8, false] | The algorithm should find a list of nodes that contains the nodes [3, 9, false], [3, 7, true], [2, 8, false] and [4, 8, false]. | The adjacent nodes list contains [3, 9, false], [3, 7, true], [2, 8, false] and [4, 8, false]. | adjacent\_Nodes.toString():  Figure 2.3 |
| 2.3 | Check that the program calculates G cost correctly. | Node of which the G cost is being calculated = [4, 8, false] | The algorithm should find the G cost of [4, 8, false] to be 1. | The algorithm found the G cost of [4, 8, false] to be 1. | Values of [4, 8, false]:  Figure 2.4 |
| 2.4 | Check that the program calculates M cost correctly. | Node of which the M cost is being calculated = [4, 8, false] | The algorithm should find the M cost of [4, 8, false] to be:  7-4 + (2-8)\*(-1) = 9  9. | The algorithm found the M cost of [4, 8, false] to be 9. | Values of [4, 8, false]:  Figure 2.4 |
| 2.5 | Check that the program calculates F cost correctly. | Node of which the F cost is being calculated = [4, 8, false] | The algorithm should find the F cost [4, 8, false] to be:  9 + 1 = 10  10. | The algorithm found the F cost of [4, 8, false] to be 10. | Values of [4, 8, false]:  Figure 2.4 |
| 2.6 | Check that the program finds the node with the lowest F cost on the open list correctly sets it as the Current Node. | Open List = [[2, 8, false][3, 9, false][4, 8, false]] | The algorithm should find that [4, 8, false] has the lowest F cost in the open list. | The algorithm found [4, 8, false] as the lowest F cost Node on the Open List. | current\_Node.toString():  Figure 2.5 |
| 2.7 | Check that once the program has finished it shows a correct path | Start Node = [3, 8, false]  End Node = [7, 2, false] | The algorithm should find a valid path from the start node to the end node that will be displayed on the visualizer, and the title of the window should change to reflect that a path has been found. | A valid path is displayed on the visualizer and ‘PATH FOUND’ is added to the title of the window. | Final Path:  Figure 2.6 |

#### Evidence

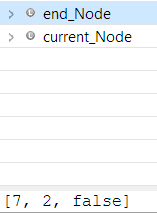


Figure 2.2

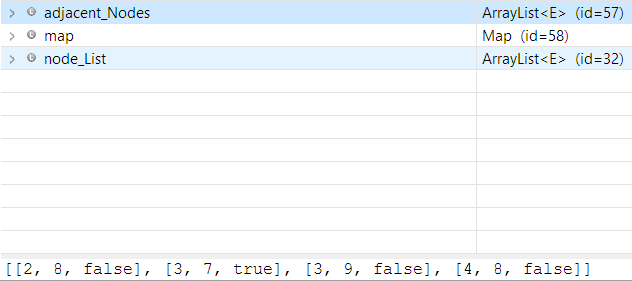


Figure 2.

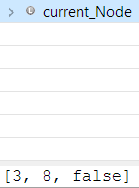


Figure 2.1

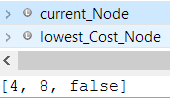


Figure 2.5

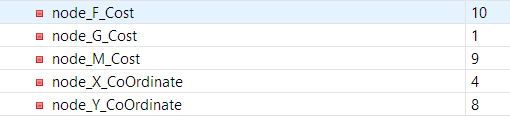


Figure 2.

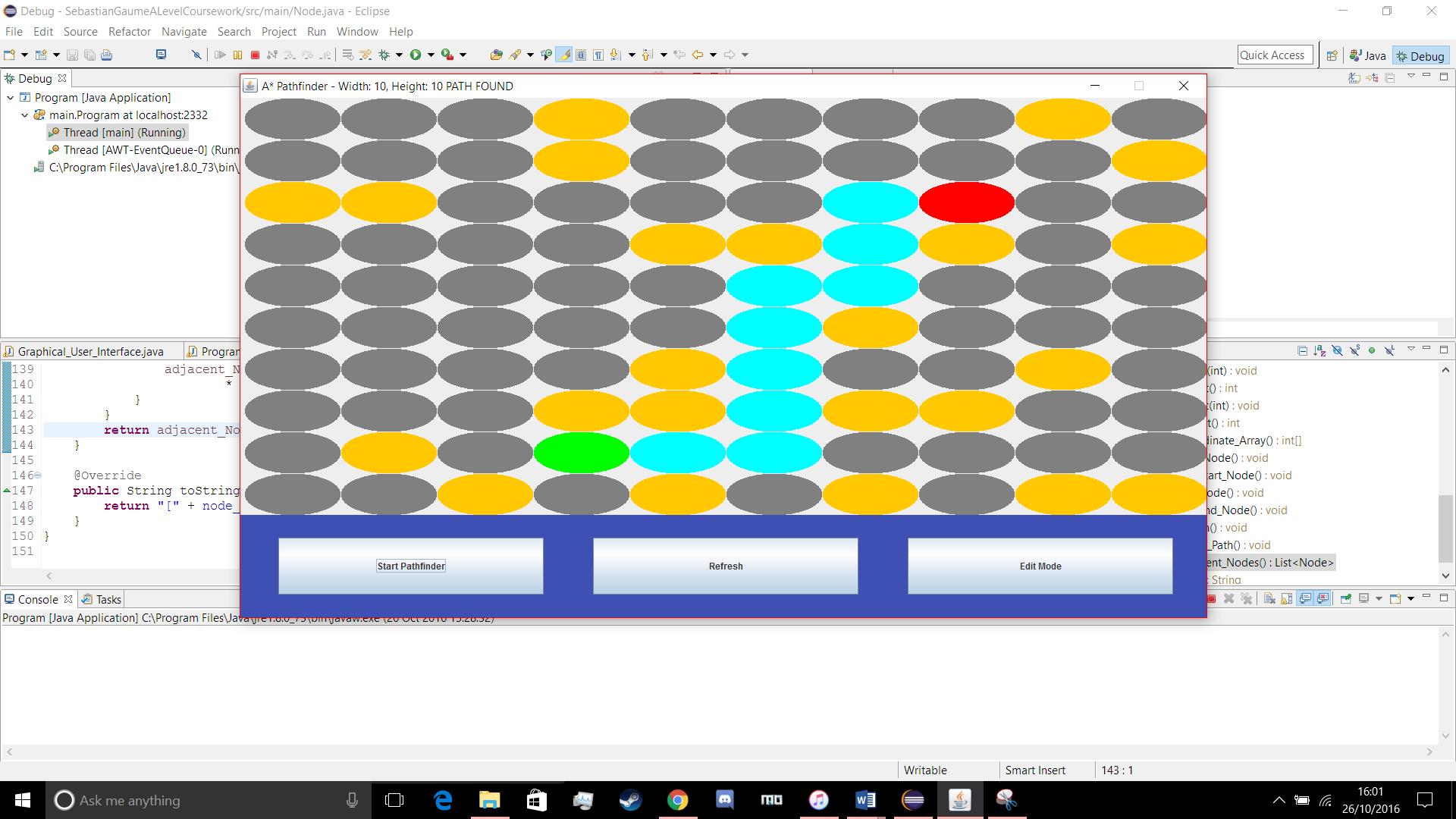


Figure 2.6

#### Dry Run Test

###### Test Map

Key:

S – Start Node

E – End Node

X – Wall Node

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | S |  | X |  |  |
| 1 |  |  | X |  |  |
| 2 |  | X |  |  |  |
| 3 |  | X |  | E |  |
| 4 |  |  |  |  |  |

###### Main Pathfinding Algorithm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Main Loop Number | Current Node | Open List | Closed List | Explanation |
| 1 | [0, 0] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | null | null | null | null | | The start node has been set as the current node and added to the open list. |
| 1 | [0, 0] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | null | null | null | null | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | The start node has the lowest cost in the open list, so it is switched to the closed list. |
| 1 | [0, 0] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 1 | 0 | 0 | | 1 | 0 | 0 | 0 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | The valid neighbours of the current node have been added to the open list. |
| 2 | [1, 0] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 1 | 0 | 0 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | [1, 0] and [0, 1] both have equal F costs, so the algorithm picks one, sets it as the current node and switches it to the closed list. |
| 2 | [1, 0] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 1 | 0 | 0 | | 1 | 1 | 1 | 0 | | 0 | 2 | 1 | 0 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | The valid neighbours of [1, 0] have been added to the open list. |
| 3 | [0, 1] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 1 | 1 | 1 | 0 | | 0 | 2 | 1 | 0 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | All the nodes on the open list have the same F cost, so the algorithm picks one, sets it as the current node and switches it to the closed list. |
| 4 | [0, 1] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 1 | 1 | 1 | 0 | | 0 | 2 | 1 | 0 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | [0, 1] has no adjacent nodes that aren’t already part of the open list, and since the node that is in the open list doesn’t have a lower F cost via the current node their parent node does not change. |
| 5 | [0, 2] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 1 | 1 | 1 | 0 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | Both nodes on the open list have the same F cost, so one is selected, made the current node and switched to the closed list. |
| 5 | [0, 2] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | The valid neighbours of the current node are added to the open list. |
| 6 | [1, 1] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 3 | 0 | 2 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | Both nodes on the open list have the same F cost, so one is selected, made the current node and switched to the closed list. |
| 6 | [1, 1] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 3 | 0 | 2 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | The current node has no valid neighbours so nothing is added to the open list. |
| 7 | [0, 3] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | null | null | null | null | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | The only node on the open list is made the current node and switched to the closed list. |
| 7 | [0, 3] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 4 | 0 | 3 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | The valid neighbour of the current node was added to the open list. |
| 8 | [0, 4] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | null | null | null | null | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | 0 | 4 | 0 | 3 | | The only node on the open list is made the current node and switched to the closed list. |
| 8 | [0, 4] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 1 | 4 | 0 | 4 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | 0 | 4 | 0 | 3 | | The valid neighbour of the current node was added to the open list. |
| 9 | [1, 4] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | null | null | null | null | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | 0 | 4 | 0 | 3 | | 1 | 4 | 0 | 4 | | The only node on the open list is made the current node and switched to the closed list. |
| 9 | [1, 4] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 2 | 4 | 1 | 4 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | 0 | 4 | 0 | 3 | | 1 | 4 | 0 | 4 | | The valid neighbour of the current node was added to the open list. |
| 10 | [2, 4] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | null | null | null | null | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | 0 | 4 | 0 | 3 | | 1 | 4 | 0 | 4 | | 2 | 4 | 1 | 4 | | The only node on the open list is made the current node and switched to the closed list. |
| 10 | [2, 4] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 2 | 3 | 2 | 4 | | 3 | 4 | 2 | 4 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | 0 | 4 | 0 | 3 | | 1 | 4 | 0 | 4 | | 2 | 4 | 1 | 4 | | The valid neighbours of the current node are added to the open list. |
| 11 | [2, 3] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 3 | 4 | 2 | 4 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | 0 | 4 | 0 | 3 | | 1 | 4 | 0 | 4 | | 2 | 4 | 1 | 4 | | 2 | 3 | 2 | 4 | | Both nodes on the open list have the same F cost, so one is selected, made the current node and switched to the closed list. |
| 11 | [2, 3] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 3 | 4 | 2 | 4 | | 2 | 2 | 2 | 3 | | 3 | 3 | 2 | 3 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | 0 | 4 | 0 | 3 | | 1 | 4 | 0 | 4 | | 2 | 4 | 1 | 4 | | 2 | 3 | 2 | 4 | | The valid neighbours of the current node are added to the open list. |
| 12 | [3, 3] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 3 | 4 | 2 | 4 | | 2 | 2 | 2 | 3 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | 0 | 4 | 0 | 3 | | 1 | 4 | 0 | 4 | | 2 | 4 | 1 | 4 | | 2 | 3 | 2 | 4 | | 3 | 3 | 2 | 3 | | The node on the open list with the lowest F cost (the end node) is made the current node and switched to the closed list |
| 12 | [3, 3] | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 3 | 4 | 2 | 4 | | 2 | 2 | 2 | 3 | | 3 | 2 | 3 | 3 | | 4 | 3 | 3 | 3 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 1 | 0 | | 1 | 1 | 1 | 0 | | 0 | 3 | 0 | 2 | | 0 | 4 | 0 | 3 | | 1 | 4 | 0 | 4 | | 2 | 4 | 1 | 4 | | 2 | 3 | 2 | 4 | | 3 | 3 | 2 | 3 | | The adjacent nodes of the current node are added to the open list, and though one is already on the open list, since its new F cost would be higher than it’s current F cost it’s parent node is not changed. |
| 13 | N/A | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 3 | 4 | 2 | 4 | | 2 | 2 | 2 | 3 | | 3 | 2 | 3 | 3 | | 4 | 3 | 3 | 3 | | |  |  |  |  | | --- | --- | --- | --- | | x | y | px | py | | 0 | 0 | null | null | | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 0 | 2 | 0 | 1 | | 1 | 1 | 0 | 1 | | 0 | 3 | 0 | 2 | | 0 | 4 | 0 | 3 | | 1 | 4 | 0 | 4 | | 2 | 4 | 1 | 4 | | 2 | 3 | 2 | 4 | | 3 | 3 | 2 | 3 | | The end node is in the closed list, so the program leaves the main loop of the pathfinder and begins to reverse engineer the path. |

###### Getting the Path

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Times Recurred | Current Node | Path | Is the Current Node The Start Node? | Explanation |
| 1 | [3, 3] | |  |  | | --- | --- | | x | y | | 2 | 3 | | No | The algorithm starts from the end node and is called recursively with the parent of the current node until the start node is reached. |
| 2 | [2, 3] | |  |  | | --- | --- | | x | y | | 2 | 3 | | 2 | 4 | | No | With each recursion a new list is created, and the current node’s parent node is added to it. |
| 3 | [2, 4] | |  |  | | --- | --- | | x | y | | 2 | 3 | | 2 | 4 | | 1 | 4 | | No |  |
| 4 | [1, 4] | |  |  | | --- | --- | | x | y | | 2 | 3 | | 2 | 4 | | 1 | 4 | | 0 | 4 | | No |  |
| 5 | [0, 4] | |  |  | | --- | --- | | x | y | | 2 | 3 | | 2 | 4 | | 1 | 4 | | 0 | 4 | | 0 | 3 | | No |  |
| 6 | [0, 3] | |  |  | | --- | --- | | x | y | | 2 | 3 | | 2 | 4 | | 1 | 4 | | 0 | 4 | | 0 | 3 | | 0 | 2 | | No |  |
| 7 | [0, 2] | |  |  | | --- | --- | | x | y | | 2 | 3 | | 2 | 4 | | 1 | 4 | | 0 | 4 | | 0 | 3 | | 0 | 2 | | 0 | 1 | |  |  |
| 8 | [1, 0] | |  |  | | --- | --- | | x | y | | 2 | 3 | | 2 | 4 | | 1 | 4 | | 0 | 4 | | 0 | 3 | | 0 | 2 | | 0 | 1 | | 0 | 0 | | No |  |
| 9 | [0, 0] | |  |  | | --- | --- | | x | y | | 2 | 3 | | 2 | 4 | | 1 | 4 | | 0 | 4 | | 0 | 3 | | 0 | 2 | | 0 | 1 | | 0 | 0 | | Yes | When the start node is reached all of the lists of nodes are added together to create a list containing the whole path. |

###### End Path

Key:

S – Start Node

E – End Node

X – End Node

P – Node in the Path

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | S |  | X |  |  |
| 1 | P |  | X |  |  |
| 2 | P | X |  |  |  |
| 3 | P | X | P | E |  |
| 4 | P | P | P |  |  |

# Appraisal

## Comparison of Project Performance Against My General and Specific Objectives

My comments will be written in this colour.

#### General Objectives

To create a computerized, interactive system that demonstrates the use of the A\* pathfinding algorithm and helps promote understanding of how the algorithm works for 8-14 year olds. The system will have a Graphical User Interface, a graphical representation of the algorithm in progress and the ability for students to edit the map that the algorithm will be pathfinding through.

I have created an interactive pathfinder that allows students to see how fast and effective a pathfinder using the A\* pathfinding algorithm, and how it reacts in different situations. The system has a graphical user interface, which gives students the ability to edit the map that the pathfinder path finds through, however it does not show the algorithm in progress as the algorithm goes so quickly that any attempt to show it in progress wold just cause the visualizer to be re-drawn so often that the JPanel would not be able to keep up.

#### Specific Objectives

1. A Graphical User Interface will be added

A Graphical User Interface was implemented that runs when the program is started

* 1. The Graphical User Interface will have a simulation mode

The Graphical User Interface has a simulation mode with the intended menu

* + 1. In simulation mode the G.U.I. will display the graphical representation of the map

In simulation mode the G.U.I. displays a graphical representation of the map above the menu

* + 1. The program will launch in simulation mode

The program starts in simulation mode when launched

* + 1. In simulation mode there will be a button to run the algorithm

In simulation mode there is a button that runs the algorithm on the left of the menu

* + 1. In simulation mode there will be a button to switch to the editing mode

In simulation mode there is a button to switch to the editing mode on the right of the menu

* 1. The Graphical User Interface will have an editing mode

The G.U.I. has an edit mode which can accessed from the simulation mode

* + 1. In the editing mode the G.U.I. will display the graphical representation of the map

In edit mode the G.U.I. displays a graphical representation of the map above the menu

* + 1. In the editing mode the user will be able to select a node or path location from the graphical representation

In edit mode the user cannot select individual nodes due to the difficulty of implementing such a system, especially as the size of the map increases – there could be upwards of 10000 nodes, and since the nodes get smaller the more there are, it becomes simply impractical to get input from each one. Instead a randomization button was added, which regenerates the map randomly

* + - 1. While a node is selected the user will be able to change the type of the node

See above

* + - 1. While a path location is selected the user will be able to add/remove the path

See above

* + 1. In editing mode the user will be able to change the dimensions of the map (in nodes)

In edit mode the user can increase or decrease the width and height 5 nodes at a time by using the buttons, at the same time a new random map is generated

* + - 1. The default dimensions will be 20 nodes by 20 nodes

The default map dimensions are 20 nodes by 20 nodes when the program is started

* + 1. In the editing mode there will be a button to switch to simulation mode

In edit mode there is a button to switch to simulation mode on the far right of the menu

1. A graphical representation of the map will be added

A graphical representation of the map was added and is visible at all times, above the menu

* 1. Each node of the graph will be represented by a coloured circle

Each node of the map is represented by a coloured circle on the graphical representation

* + 1. Nodes that are part of the open list will be coloured white

Doing this would require an update each time a node is added to the open list, and given how many times that happens every second the JPanel would not be able to keep up, instead all nodes in the open list are treated as if they are normal nodes

* + 1. Nodes that are part of the closed list will be coloured black

Doing this would require an update each time a node is added to the closed list, and given how many times that happens every second the JPanel would not be able to keep up, instead all nodes in the closed list are treated as if they are normal nodes

* + 1. Wall nodes will be coloured orange

Wall nodes are coloured orange on the graphical representation

* + 1. The start node will be coloured green

The start node is coloured green on the graphical representation

* + 1. The end node will be coloured red

The end node is coloured red on the graphical representation

* + 1. Normal nodes (Non-Wall, Non-Start/End nodes) will be coloured grey

Normal nodes are coloured grey on the graphical representation

1. The A\* pathfinding algorithm will be implemented

The A\* pathfinding algorithm is implemented as the algorithm used to find the path

* 1. The algorithm will be based upon the algorithm that LogicOut currently teaches (see appendixes)

The algorithm is based on the Manhattan version of the A\* pathfinding algorithm that LogicOut teaches

* 1. The algorithm will use lists of nodes

The algorithm uses two lists of nodes as part of the algorithm

* + 1. The algorithm will use a ‘closed list’ to track all the nodes that it has already checked

The algorithm uses a ‘closed list’ to track all the nodes that it has already checked in the algorithm

* + 1. The algorithm will use an ‘open list’ to track all the nodes that it can check next

The algorithm uses an ‘open list’ to track all nodes that can be checked next in the algorithm

* 1. The algorithm will find a path from the designated start node to the designated end node

The algorithm finds a path from the designated start node to the designate end node when it is begun through the menu

1. The user will be able to run the algorithm

The user can run the algorithm by using the leftmost button in the simulation mode menu

* 1. The user will be able to press a button to start the algorithm

There is a button in simulation mode that can start the algorithm – the leftmost button

* + 1. When the algorithm starts it will be run based on how the graph has been edited

When the algorithm starts it runs based on the most recent version of the map

1. The algorithm will run on LogicOut’s hardware

The algorithm will run on LogicOut’s hardware as specified by my client’s requests

* 1. The algorithm will run on Windows 8 64-bit

The algorithm will run on any system with java installed

* 1. The algorithm will run on a computer with 6GB of RAM, an 8-core intel CPU clocked at 2.67GHz and a GeForce GTX 285

The algorithm will run on a computer with 6GB of RAM, an 8-Core Intel CPU clocked at 2.67GHz and a GeForce GTX 285, as specified by my client

## User Feedback authenticated by Assesor

See Appendixes.

## Analysis of User Feedback

The user feedback was fairly positive, Nick liked the simplicity of the layout of the G.U.I. and specifically liked the ability to edit the size of the map. He was also very pleased that the algorithm works and displays the output clearly. He did point out that a more in-depth editing mode would have been nice, I feel that this would take some time to implement. On the other hand, it wouldn’t be too difficult to implement the networking features he requested, given a bit more time.

## Possible Extensions

#### Login System

The first extension I would add is the login system my client, Nick, requested, as the only reason it was not included in the final system was that I lacked the skill to implement it in a reasonable amount of time, but given extension time I wold be able to learn how to implement a login system, and so I would.

#### Specific Node selction and Editing

In my objectives I stated that ‘In the editing mode the user will be able to select a node or path location from the graphical representation’, which was not implemented in my system, as I couldn’t think of a way to handle such a system at the larger map sizes without having the selection process be impractical. Given time, however, I would likely be able to come up with a solution that would allow for the system to be more interactive and still be practical, even at the largest map dimension – 200 by 200 nodes.

#### Node Stretching / Spacing

Currently as you change the dimensions of the map the nodes are scaled and spaced to make sure they fit inside the area given to the visualizer, however at some dimensions the scaling and spacing is implemented in such a way that there is free space around the graphical representation, and while I was not able to find a good solution in the time available, given more time I would hopefully be able to find a method of stretching, scaling and spacing the nodes such that there is no free space around the graphical representation.

#### Real-Time graphcial Representation of the algorithm in progress

In my objectives I stated that I wanted to colour nodes different colours if they were in the open/closed list in real-time, however the algorithm I designed ran so fast that if I had rendered it in real time the JPanel I was using for the graphical representation would not have been able to keep up with the algorithm and would have just flickered until the algorithm finished. Given more time, however, I could come up with a way to slow down the algorithm to a point where displaying it’s progress in real time is possible, and possibly even give the user control over how fast the algorithm attempts to run.

#### Design new Buttons for the G.U.I.

The java swing library’s buttons are, in my opinion, not at all visually appealing and given time I would have liked to redesign them in such a way that users would find more appealing

## Appraisal Appendices