Path Finding Algorithm Analysis, Spring 2020

Group 2 - Object Oriented Software Design

Adam Corbin (acorbin3@fau.edu)

April 2020

Contents

1 Path Finding Algorithm Analysis, Spring 2020 **3**
- 1.1 Group 2 - Object Oriented Software Design . . . 3
- 1.2 Adam Corbin - April 2020 . . . 3

2 Functional Specification **3**

3 Description **3**
- 3.1 List of things that this program will do . . . 3
- 3.2 Intended users . . . 4

4 Application Requirements **4**
- 4.0.1 Functional Requirements . . . 4
- 4.0.2 UI Requirements . . . 4
- 4.1 Use Cases . . . 4
  - 4.1.1 UC-1: User selects a path . . . 4
  - 4.1.2 UC-2: User selects a path . . . 4
  - 4.1.3 UC-3: Computer generates path . . . 4
  - 4.1.4 UC-4: Computer generates path . . . 4
  - 4.1.5 UC-5: User selects a different algorithm . . . 5
  - 4.1.6 User selects points from history . . . 5
  - 4.1.7 UC-6: System populates algorithm comparator . . . 5
- 4.2 Use Cases updated after GUI elements . . . 5
  - 4.2.1 UC-1: User selects a path . . . 5
  - 4.2.2 UC-2: User selects a path . . . 5
  - 4.2.3 UC-3: Computer generates path . . . 5
  - 4.2.4 UC-4: Computer generates path . . . 5
  - 4.2.5 UC-5: User selects a different algorithm . . . 5
  - 4.2.6 UC-6: System populates algorithm comparator . . . 6
  - 4.2.7 UC-7: Generate New Graph . . . 6
  - 4.2.8 UC-8: Generate graph from text fields . . . 6

5 User Interface **6**
- 5.1 Mockup . . . 6

6 Design Specification **6**
- 6.1 CRC cards . . . 6
  - 6.1.1 Tile . . . 6
  - 6.1.2 Path . . . 6
  - 6.1.3 Algorithm . . . 6
  - 6.1.4 Grid . . . 8
  - 6.1.5 GridTile . . . 8
  - 6.1.6 AlgorithmController . . . 8
  - 6.1.7 GUIView . . . 8
- 6.2 UML Diagrams . . . 8
  - 6.2.1 Patterns . . . 8
  - 6.2.2 Class diagrams . . . 9
  - 6.2.3 Algorithms package uml . . . 11
  - 6.2.4 View package uml . . . 12
  - 6.2.5 Model package uml . . . 13
  - 6.2.6 Sequence Diagrams . . . 13

6.2.7 State Diagrams . . . 17

7 Screenshots **22**

8 Code **27**
8.1 Algorithms Package . . . 27
8.1.1 Algorithm.java . . . 27
8.1.2 AStar.java . . . 28
8.1.3 BDS.java . . . 31
8.1.4 DFS.java . . . 33
8.2 Controller package . . . 36
8.2.1 AlgorithmController.java . . . 36
8.3 Model package . . . 40
8.3.1 AlgorithmResult.java . . . 40
8.3.2 AlgorithmStatModel.java . . . 40
8.3.3 GridModel.java . . . 45
8.3.4 GridTile.java . . . 48
8.3.5 Path.java . . . 48
8.3.6 Tile.java . . . 50
8.4 Simulation Package . . . 50
8.4.1 Simulation.java . . . 50
8.5 View Package . . . 52
8.5.1 GridView.java . . . 52
8.5.2 GUIView.java . . . 55
8.6 JUnit Tests . . . 59
8.6.1 AlgorithmStatModelTest.java . . . 59
8.6.2 GridTileTest.java . . . 62
8.6.3 PathTest.java . . . 62
8.6.4 TileTest.java . . . 64

9 Glossary **64**

1 Path Finding Algorithm Analysis, Spring 2020

1.1 Group 2 - Object Oriented Software Design

1.2 Adam Corbin - April 2020

2 Functional Specification

3 Description

This application will dive into the different types of path finding algorithms games use in order to come up with the generated path when a character or player selects a destination position. This project will have a visual progression of the algorithms so that a human can see how well they preform between each other. I would like to evaluate different situations to find positives and negatives between the algorithms such as best case and worst-case scenarios. Statistic analysis will also be done to evaluate how well they rank between each other.

3.1 List of things that this program will do

- Visual graph representing at least 1 or more algorithms running over time
- A way to view the results between the different algorithm
- A way for a human to pick start and destination points.

- A way to auto pick start and destination points

This will run on Windows 10 OS using Swing

3.2 Intended users

- Any game developers who want to consider different options for path finding.
- Anyone who wants to understand how path finding algorithms work.

4 Application Requirements

4.0.1 Functional Requirements

1. **F.R-1**: The system shall generate a path given a start point, and end point, and the selected algorithm
2. **F.R-2**: The system shall let the user know if a path is not possible to be created
3. **F.R-3**: The system shall provide statistics on the different algorithms
4. **F.R-4**: The system shall find worst case scenario for each algorithm given the scenario
5. **F.R-5**: The system shall run a simulation to compare the differences between the algorithms using simple statistics
6. **F.R-6**: The system shall be able to generate a new random graph
7. **F.R-7**: The system shall be able to generate a new random start and end position

4.0.2 UI Requirements

1. **UI.R-1**: The UI shall be able to select a starting point and an end point
2. **UI.R-2**: The UI shall have a button to start the simulation
3. **UI.R-3**: The UI shall be able to see the algorithms behave over time where a human can visually see
4. **UI.R-4**: The UI shall have the ability to auto pick the 2 points
5. **UI.R-5**: The UI shall have the ability to select which algorithms to use for the simulation
6. **UI.R-6**: The UI shall have a table comparing the different algorithms
7. **UI.R-7**: The UI shall have multiple size graphs to pick from

4.1 Use Cases

4.1.1 UC-1: User selects a path

1. The user selects the start and end points and uses default algorithm
2. The system computes the path generation
3. The system displays a successful path

4.1.2 UC-2: User selects a path

4.1.2.1 Variation #1: Invalid path

1. After step 2 , the system displays the path could not be found

4.1.3 UC-3: Computer generates path

1. The user selects for the system to pick 2 random points on the graph
2. The system picks 2 random points
3. The system computes the path generation
4. The system displays a successful path

4.1.4 UC-4: Computer generates path

4.1.4.1 Variation #1: Invalid path

1. After step 2, the system displays that the path could not be found

4.1.5 UC-5: User selects a different algorithm

1. The user selects the start and end points
2. The user changes the default algorithm to another selection
3. The system computes the path generation
4. The system displays the successful path

4.1.6 User selects points from history

1. The user picks from this history list to switch back to
2. The system regenerates the path from the history
3. The system displays the generated path

4.1.7 UC-6: System populates algorithm comparator

1. The user selects all the algorithms to run for the simulation
2. The user picks the 2 points
3. The system generates the paths for each algorithm
4. The system displays a ranking order between each algorithm with some statistics.

4.2 Use Cases updated after GUI elements

4.2.1 UC-1: User selects a path

1. The user enters the coordinates in the start position text field and end position text filed
2. The user selects the first algorithm JComboBox
3. The user presses the Generate Path button
4. The system computes the path generation
5. The system displays a successful path on the grid JComponent

4.2.2 UC-2: User selects a path

4.2.2.1 Variation #1: Invalid path

1. After step 2 , the system displays no the path on the graph

4.2.3 UC-3: Computer generates path

1. The user presses the generate Generate Random Path button
2. The system picks 2 random points and fills in the start and end JTextField with the positions
3. The user presses the Compute Path buttons
4. The system computes the path generation
5. The system displays a successful path on the grid JComponent

4.2.4 UC-4: Computer generates path

4.2.4.1 Variation #1: Invalid path

1. After step 2, the system displays that the path could not be found

4.2.5 UC-5: User selects a different algorithm

1. The user enters the coordinates in the start position text field and end position text filed
2. The user checks the a different algorithm JComboBox than the first
3. The user presses the Generate Path buttons
4. The system computes the path generation
5. The system displays a successful path on the grid JComponent

4.2.6 UC-6: System populates algorithm comparator

1. The user enters the coordinates in the start position text field and end position text filed
2. The user presses the Compute Comparison buttons
3. The system generates the paths for each algorithm
4. The system displays the results between each algorithm with some statistics in the algorithm results JTable

4.2.7 UC-7: Generate New Graph

1. The user presses the New Graph button
2. the system will clear the graph JComponent
3. the system will then repopulate GridModel blocking elements
4. The system will then repaint the graph JComponent

4.2.8 UC-8: Generate graph from text fields

1. The user enters the coordinates in the start position text field and end position text filed
2. The system automatically computes the path generation
3. The system displays a successful path on the grid JComponent

5 User Interface

5.1 Mockup

Link: https://gomockingbird.com/projects/v95sylo/4gXVnC

6 Design Specification

6.1 CRC cards

6.1.1 Tile

- Responsibilities
  - Coordinate on a map that is used to represent a node or a position
- Collaborators
  - Path

6.1.2 Path

- Responsibilities
  - An ordered list of tiles that will be used to represent chain of tiles to get from start to finish
- Collaborators
  - Tile
  - Algorithm
  - AlgorithmController

6.1.3 Algorithm

- Responsibilities
  - Steps to find a path between a departure and destination position
- Collaborators
  - AlgorithmController
  - Path

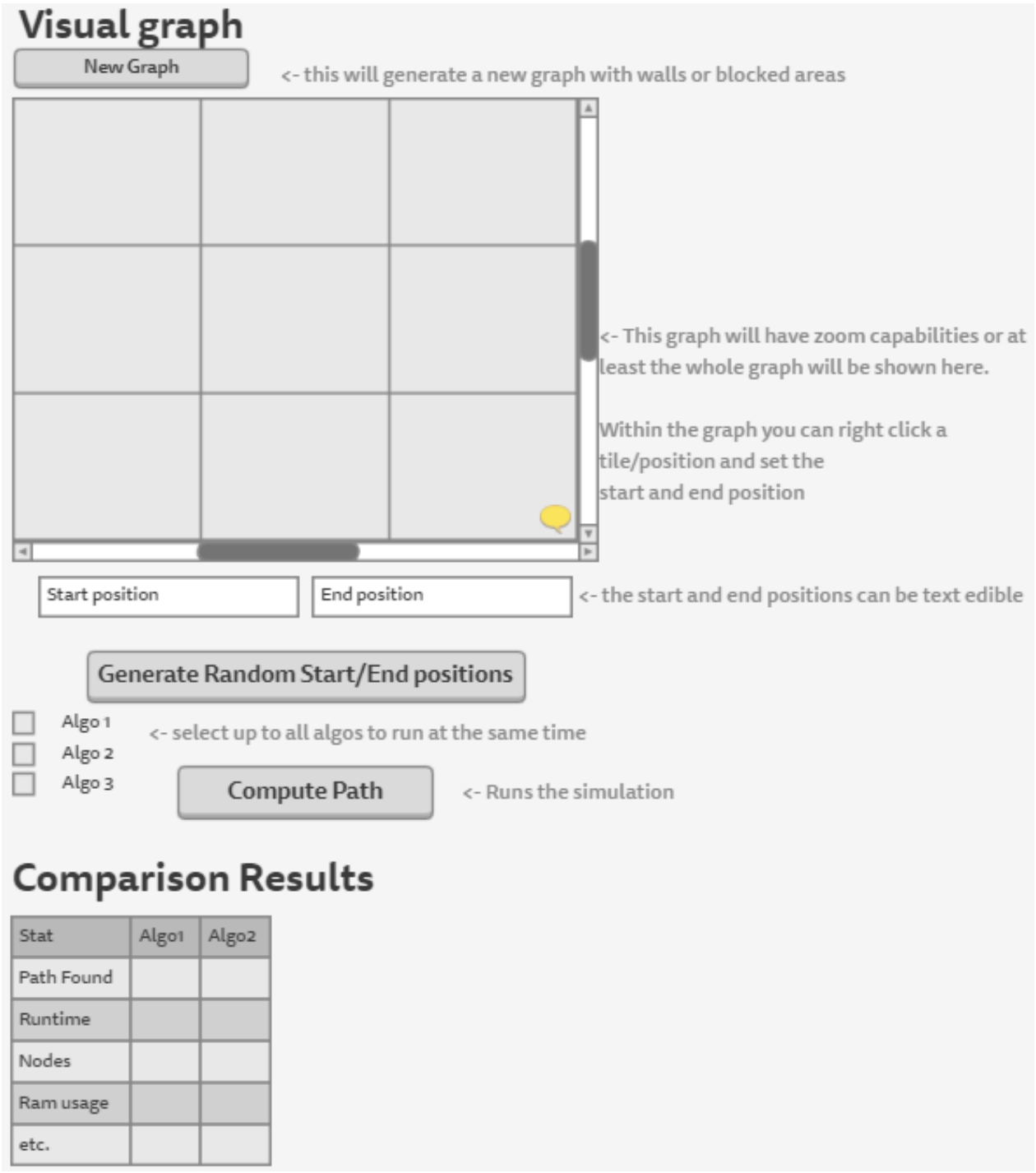


Figure 1: mockup

6.1.4 Grid

- Responsibilities
  - Showing start and end tiles of a path
  - showing the different paths for the algorithms
  - Showing the visited tiles of the algorithms
- Collaborators
  - GUIView
  - Path
  - Tile

6.1.5 GridTile

- Responsibilities
  - Keeping track of what colors should the tile display in the grid
  - collision flag of the tile
- Collaborators
  - Grid

6.1.6 AlgorithmController

- Responsibilities
  - Keeping track which algorithms are selected
  - Running the simulations
  - Keeps track of past history of the departure and destination tiles
  - Simulator that runs the algorithms and collects the comparative results
  - Keeps track of the tiles visited while going through the algorithm
  - Keeps track of optimized path for each algorithm
- Collaborators
  - GUIView
  - Grid
  - Algorithm

6.1.7 GUIView

- Responsibilities
  - Displaying the grid
  - Displaying the start and end positions
  - Selectable algorithms
  - Displaying the comparison results of the different algorithms
  - Selectable history of past start and end positions

- Collaborators
  - AlgorithmController

6.2 UML Diagrams

6.2.1 Patterns

- Observer pattern All the buttons, text fields, and JComboBoxes with Action listeners

Name in Design Pattern	Actual Name
Subject	JButton & JComboBox, JTextField
Observer	ActionListener

Name in Design Pattern	Actual Name
ConcreteObserver	AlgorithmController has 4 buttons, 2 JComboBoxes, and 2 JTextFields with action listeners it implemented
attach()	addActionListener
notify()	actionPerformed

- Singleton - The GUIView uses a Singleton pattern. I couldnt find a table in the book for this pattern.
- Composite pattern - GridTiles with parent as an object. This is used to build the path where GridTiles link between each other

Name in Design Pattern	Actual Name
Primitive	GridTile
Composite	GridTile it self
Leaf	Tile
method()	getParent()

- Adapter pattern - With AlgotStatsModel updating table

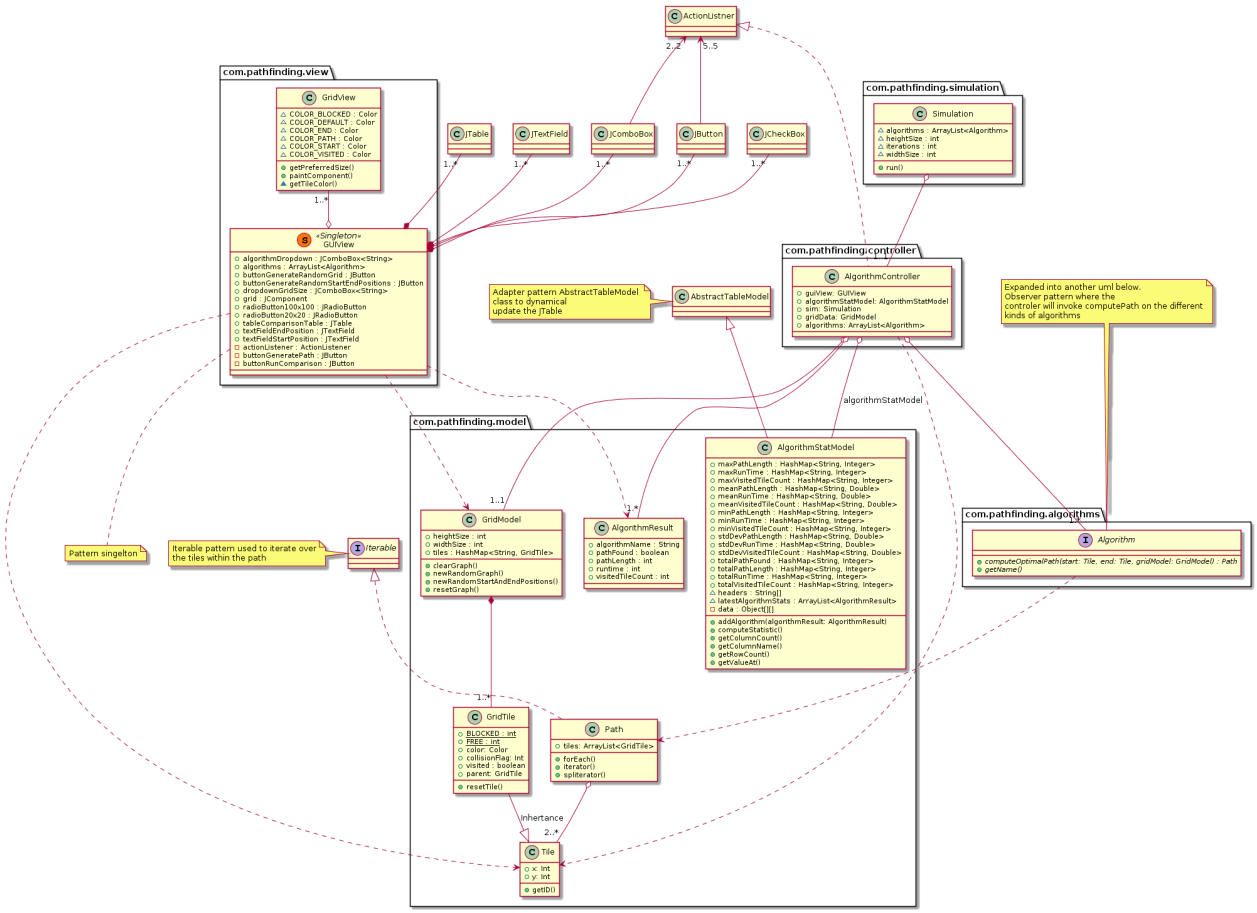
Name in Design Pattern	Actual Name
Adaptee	JTable
Target	AlgorithmStatModel
Adapter	AbstractTableModel
Client	The class that wants to add rows to a table
targetMethod()	fireTableDataChanged()
adapteeMethod()	getValueAt(),getColumnCount(), getRowCount(), getColumnName()

- Strategy pattern - GridBag

Name in Design Pattern	Actual Name
Context	GUIView
Strategy	LayoutManager
ConcreteStrategy	GridBagLayout
doWork()	A method of the LayoutManager interface type such as layoutContainer

6.2.2 Class diagrams

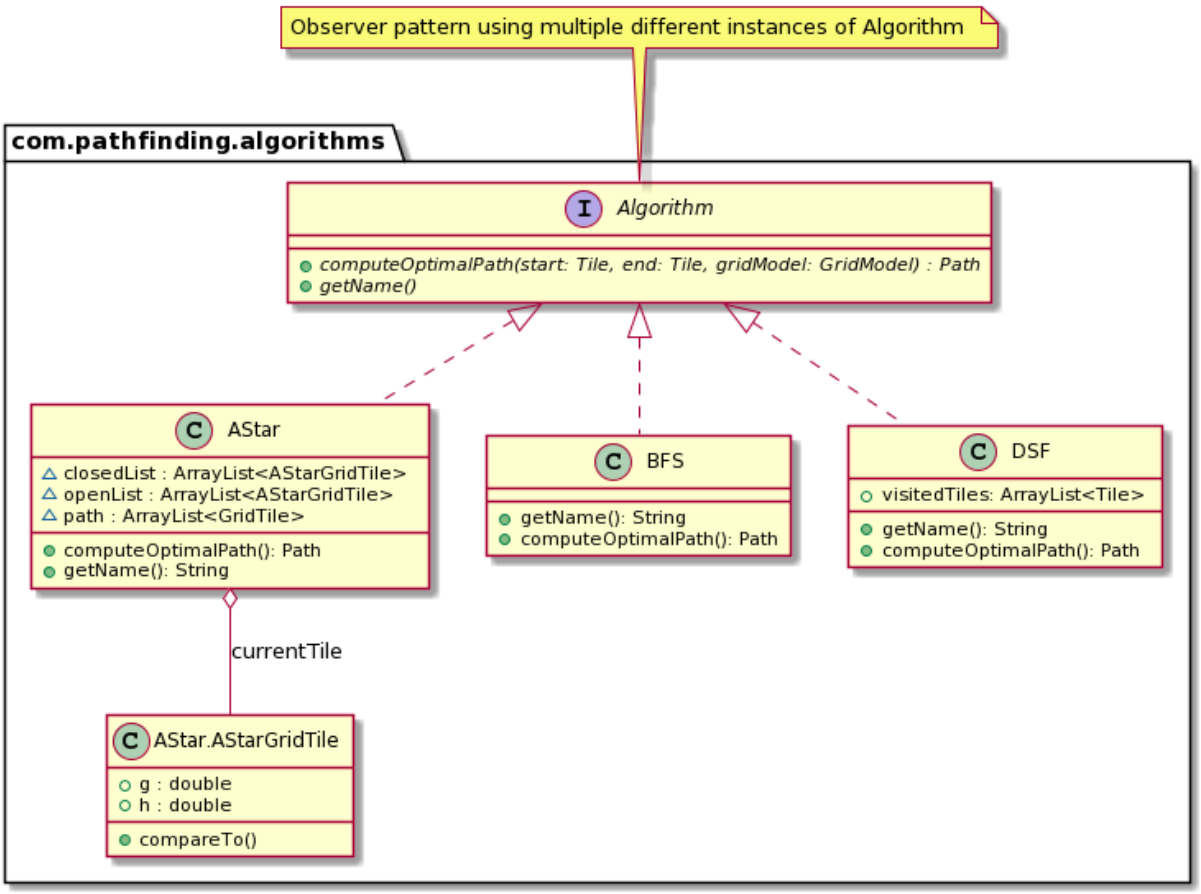
6.2.2.1 Overall UML Diagram



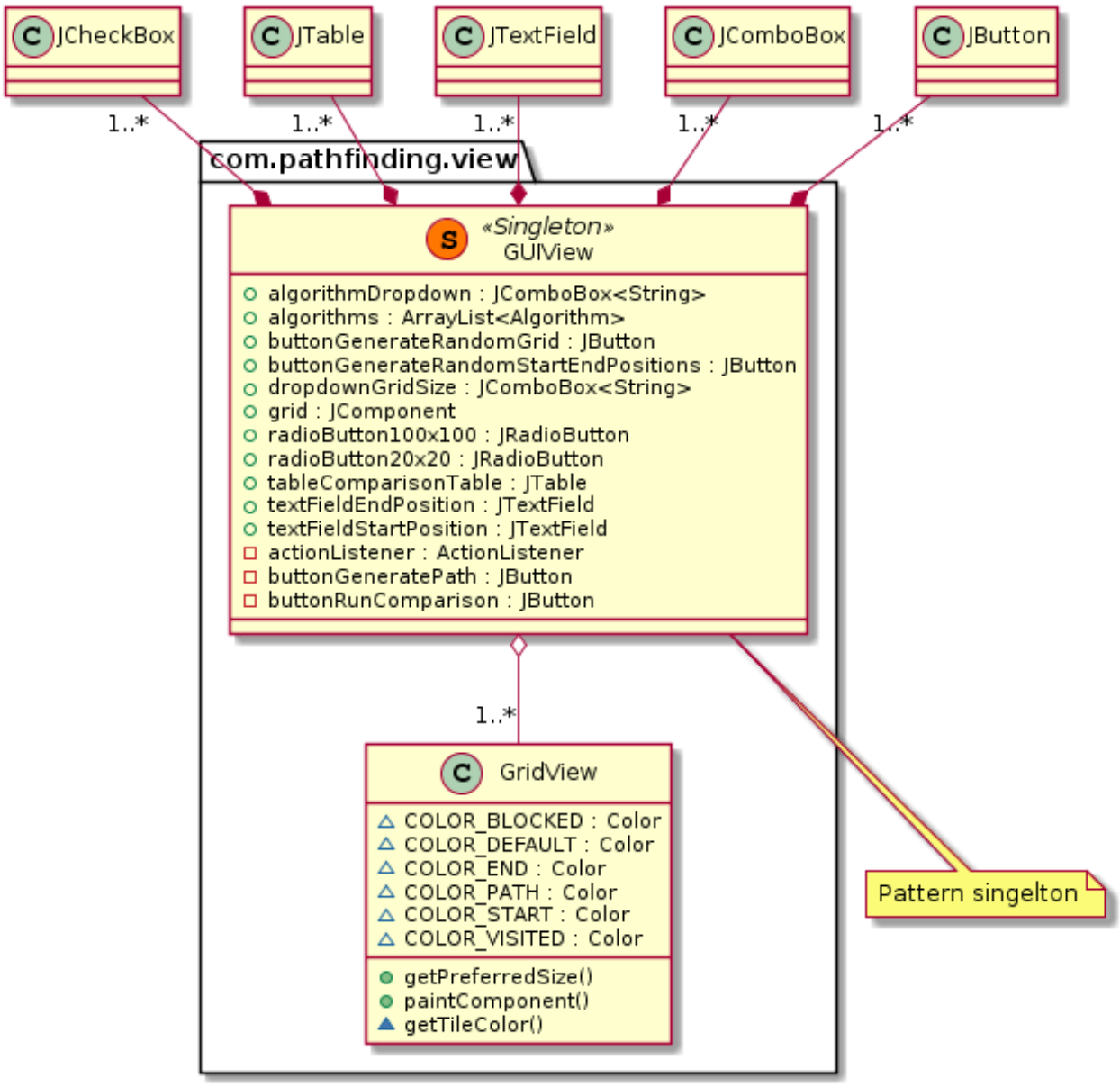
Clear picture in link below. Also opening PDF All uml diagrams are clickable to a PNG that can be zoomed in:

http://www.plantuml.com/plantuml/png/dLXjRziu3FwkNq7qlhJBOdG5XW514InhagsxRWtIh5-6uQ3OIgARBHcMtAJBTR_
IUAyU0MAsXH9aO-eXtJVAqqJdKOXKJ-vY6b28rWJya88yKEg5FWoycAg5pCk0YxcNYG35deJ7h8rWHQiEd0hj7bwpfvQC6V
ONmSSZ522V0e3bd4X5Pmgrb2fyPjIwDtrO5Uaj3dEwxqzSU41sRBBhgVoMH0_GLkq09kRBHCx8kAmyhx2n5ICmZPJ9FSDM
2WWLIm0BPqz0_d44g8N3EpEiEHwmGKOWOcRC160yKQEeD8Bvp91ssZCL1fgp0HeBlfA8mOTUD09wkE2iHxA53rtO2fboA
XHY05CEGpjs9kuYgYM0XuQ5s8D_MvC4Kq-RvU6aOgo_coI2uO7o-q0_yLmd1aWTwiI79pgq3dwXQVA7nTGRpQwWMZe-
X9HyJ6cUr8zXoJ4KWenhHWxaKFH4CH32Ic1pcgwo6JI8tU3cn6j2Rjka0Msci9GMzgT89zVN4LANB5F2d1H5TZQeri9qG0ixcB
F3HNDyb0xwxAa1Gfd1gHXbserHqbBJS5HXc2DP5FgduceDJaRMQZV5zZVUV1gmH1Mi-Vs7MmYHivNUk5z6g-
mNMqISEMn3G3L5lVlflypNttyuVqZIGL1OdDDDMtIxARXmHdZfurGpipbkoVtpV07x7v9b6WcRvgxOT8AgT2oNEvgFD4nP
nOHHlRfH5mZ8IzBNWMDE8-3KaQ1wR9cNRssQf2fQCXVx9aHdssms854AgaCiXuQatTpYyb4FSnjgciYixjKD41SzQl0VVA7-
SviMUTmJvUu8aOert7Y8F3oe6WPKm6gnWf6hiD1Yw1edo1YE000rCMogK7v7wgMF9KWMc96bdwn9T29zYZ4kmXOOW-
dkHFbcTW9nAPzVs0psRCz322On4ojfoLhLnLuAjl06uRtAU4xgQGOLURf_oHioN2yEtMF01fqPrsKIczGaPQYaTu8NYKuYt0
4Ca-LdO5quetLNilkcR4osB___27LMefryMvBf7ozaT5KVf2hzTvszuG9s9vjCrcxItfpVtS___gSSCR81ehhM7mnxt-
ucjMk3h_KsezxaK-t2mwGthdVrUFjmEzaAmHRNA65unWaBp2XHtBwOOcLXqh4KhBYKlur-HbsX6oyFOU4EzVeQpXzU__pl
gJSWLq-tDsn7XIpqTaRptjprf2O_oIZFHjjVSYG8-lmohbwTNRa0uuSNePs-x__kWOzsCDzis0yr5Vu-
NraVD05T7GZ81aC8_25eDNxh1WvoqbJYUTq3_sKWkuMpGTFxhVUVtuaEQa_iHDkjkPCu9e5s6hUxSFRMyzwULKZAS5J
vgO947qCIcPE4cHhO1D8bimVoVHoEJLPk3WqvZ_lFIsCExUUGqZjuUquxttQ3wmcaQv-5g9kGzzYeSkFy7

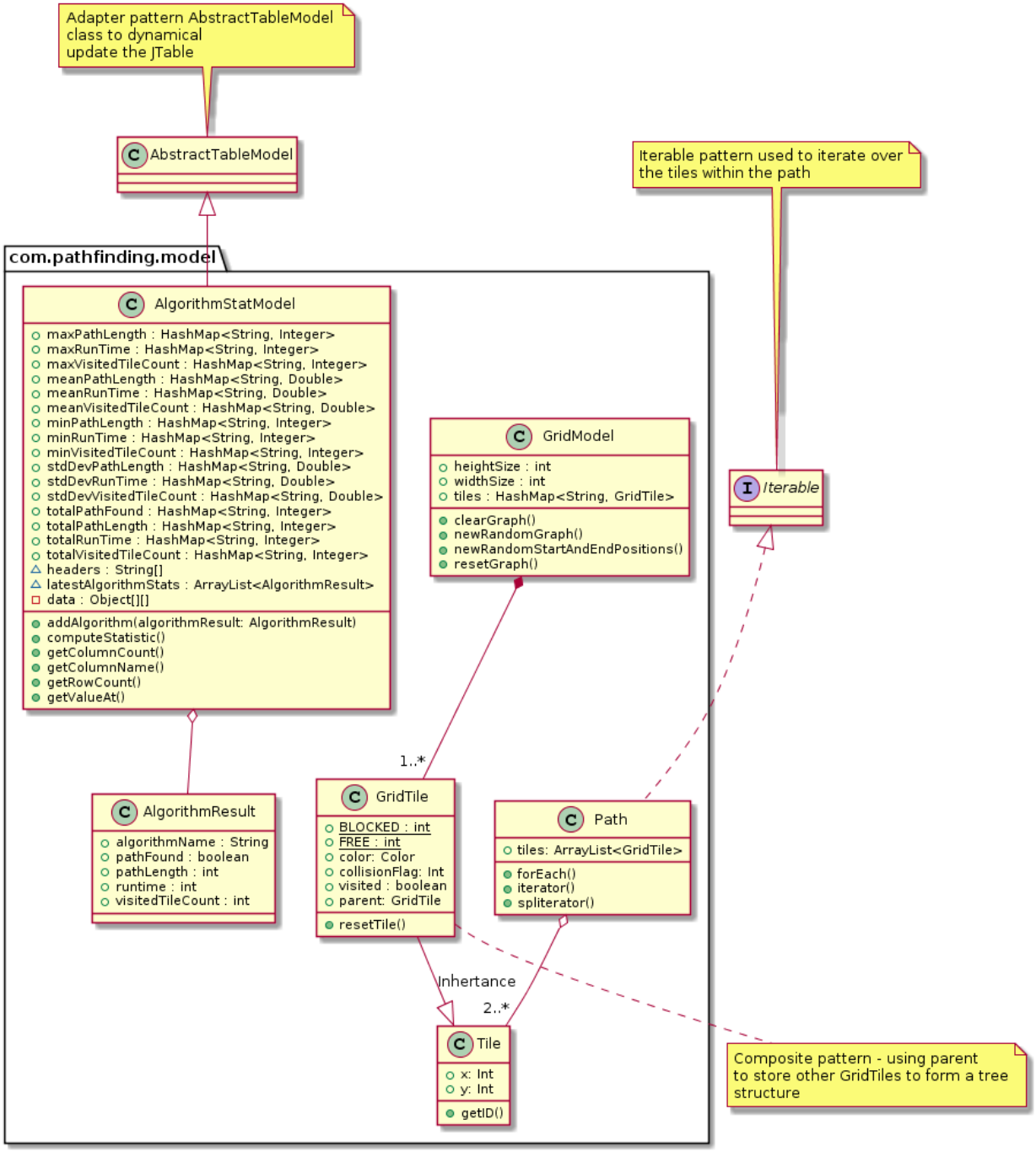
6.2.3 Algorithms package uml



6.2.4 View package uml

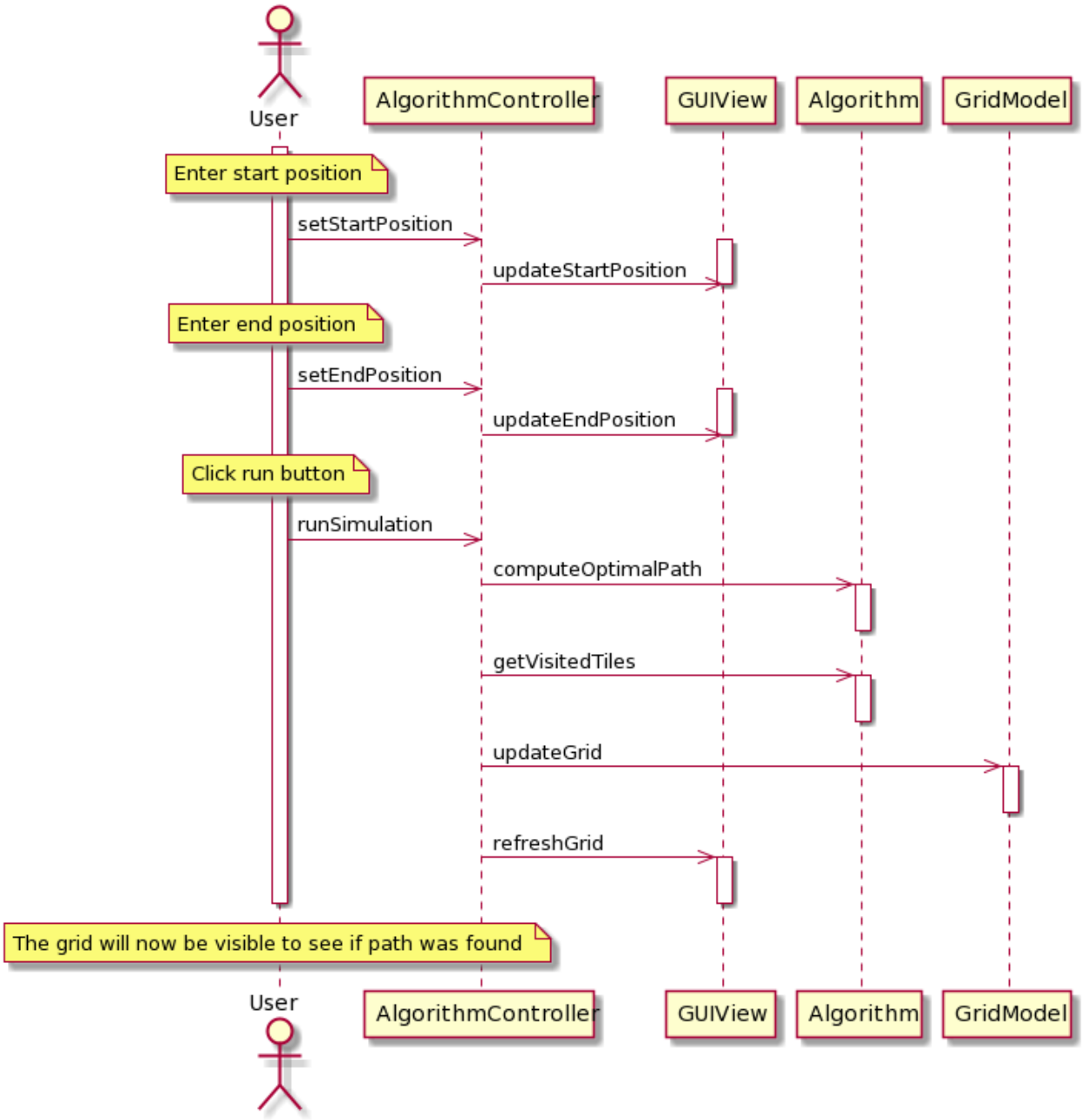


6.2.5 Model package uml

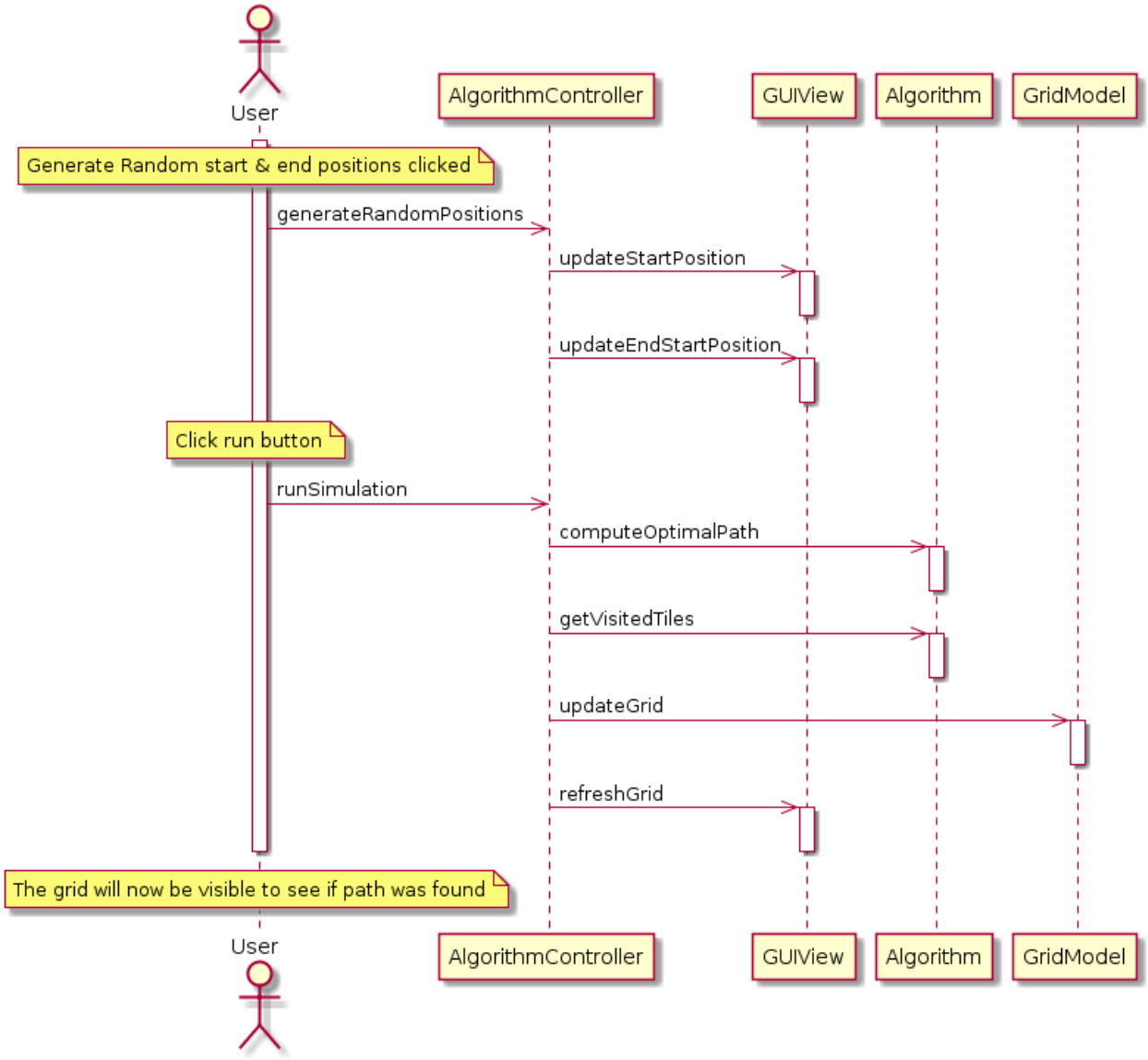


6.2.6 Sequence Diagrams

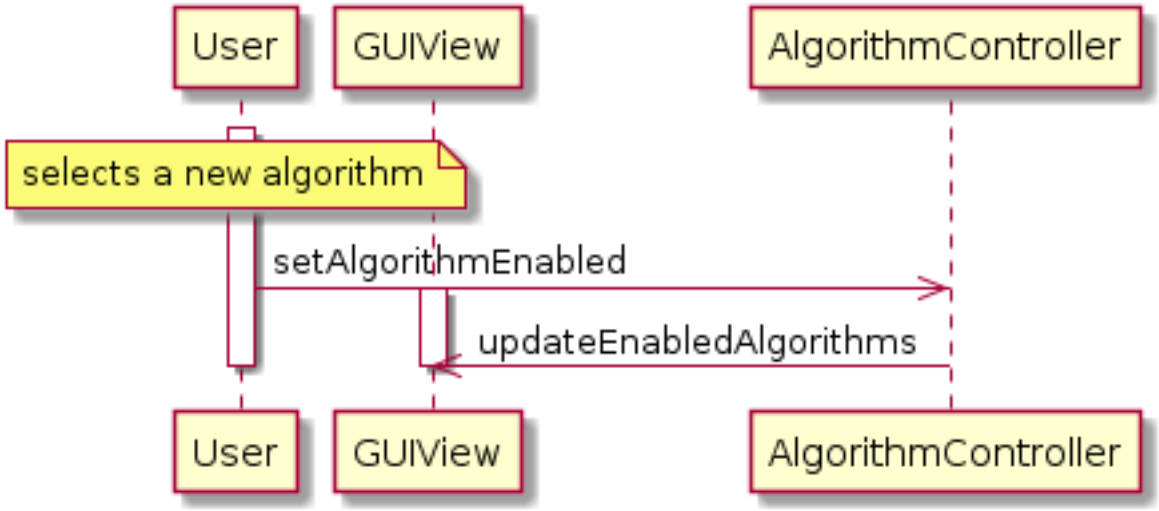
6.2.6.1 Path found



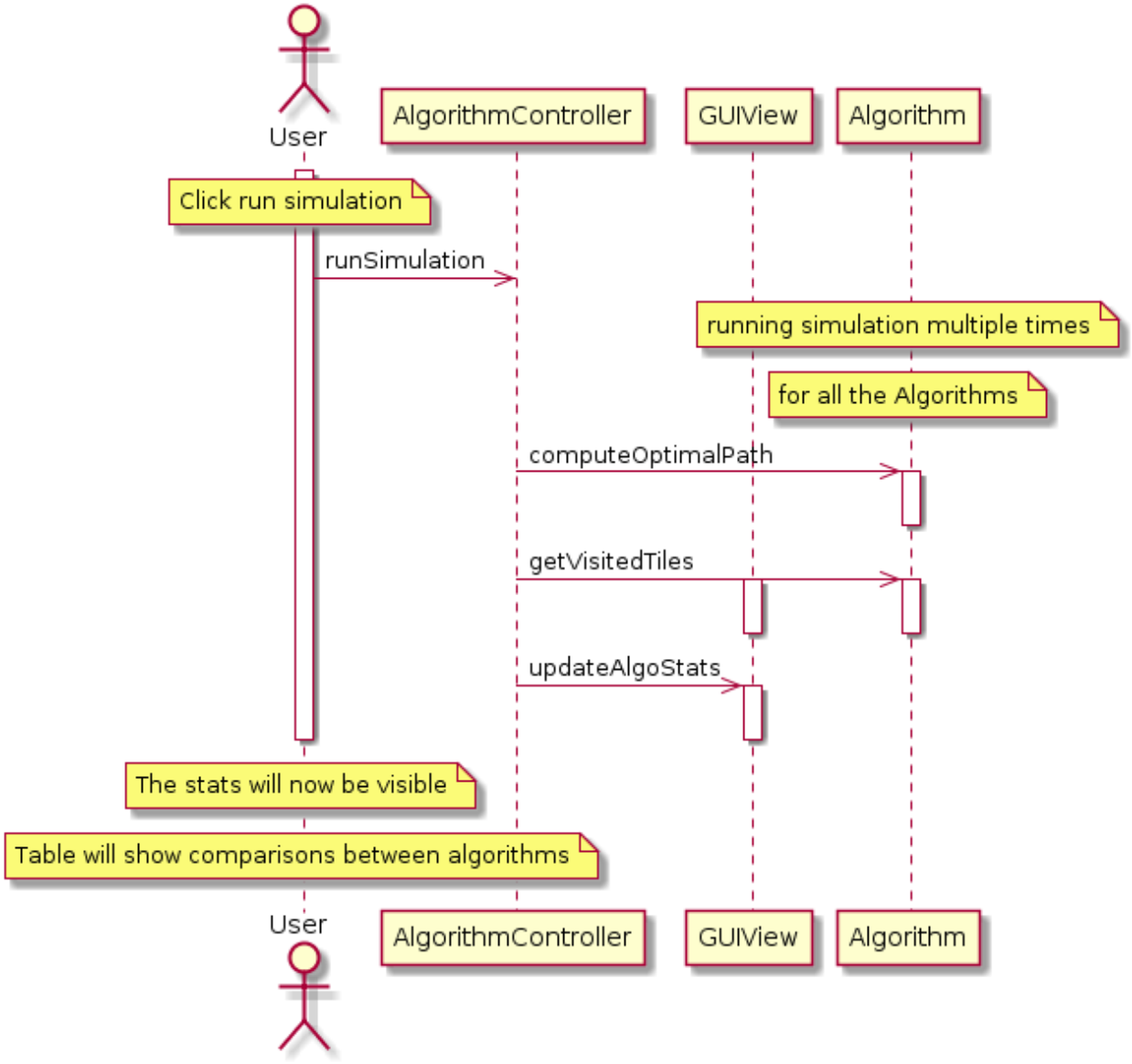
6.2.6.2 Computer generated path found



6.2.6.3 Selecting different Algorithm

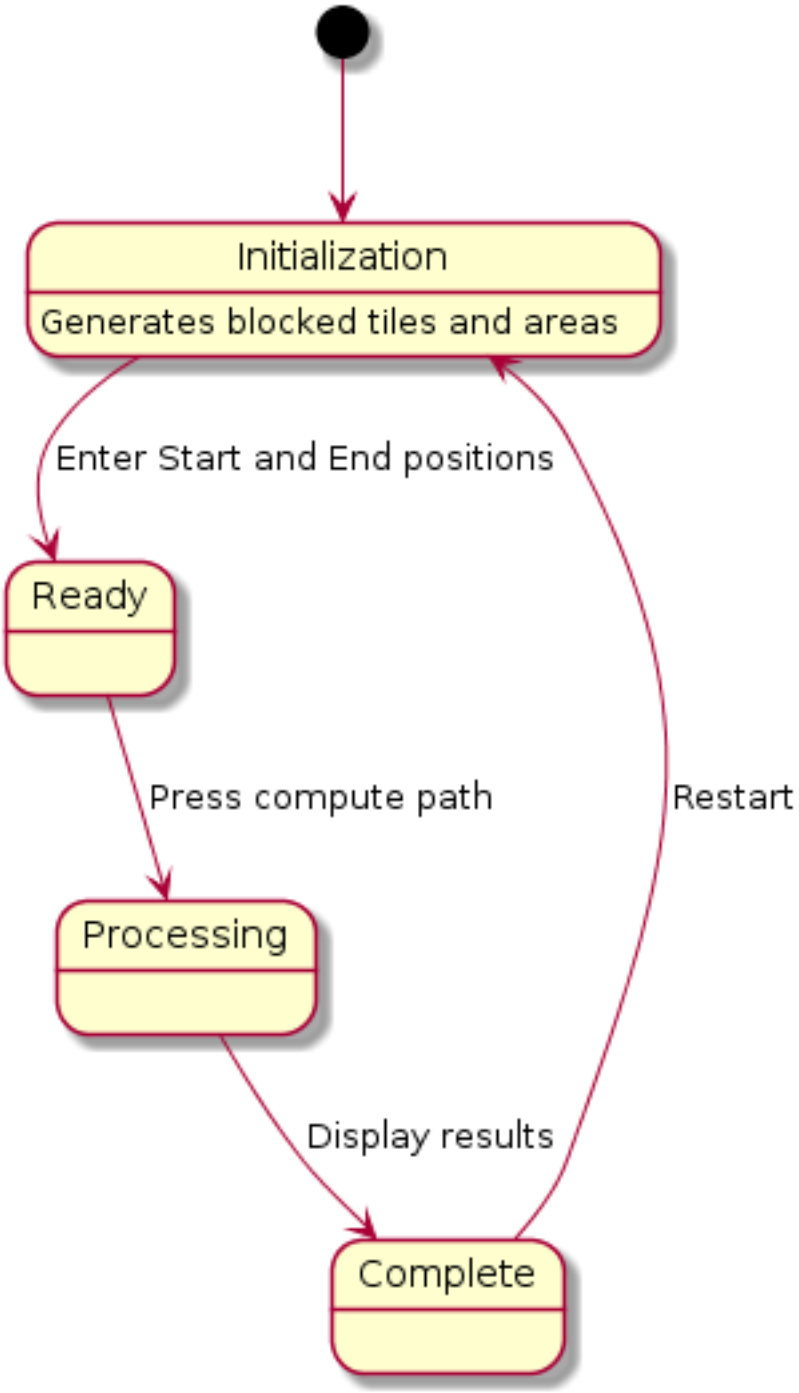


6.2.6.4 Compare algorithms

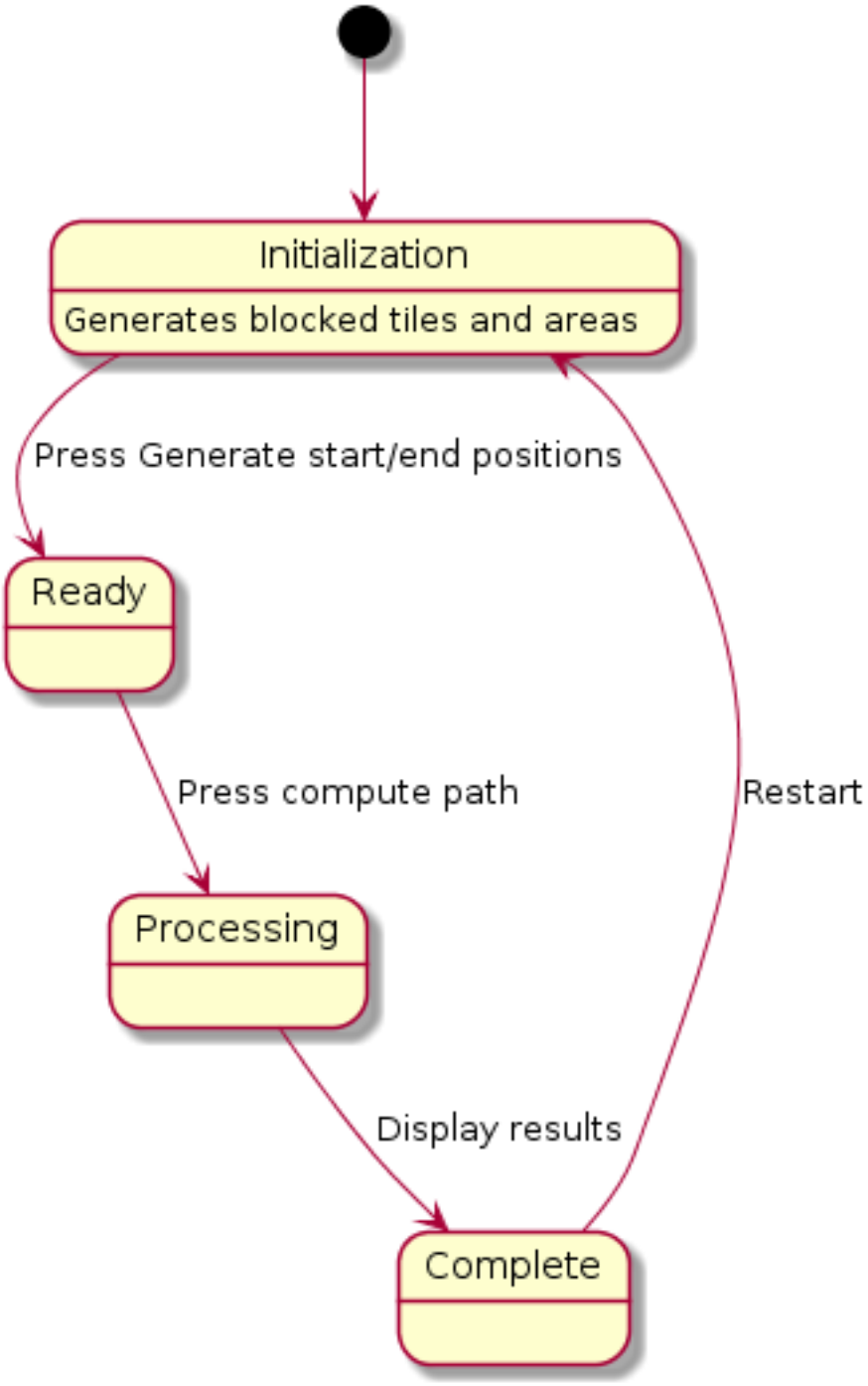


6.2.7 State Diagrams

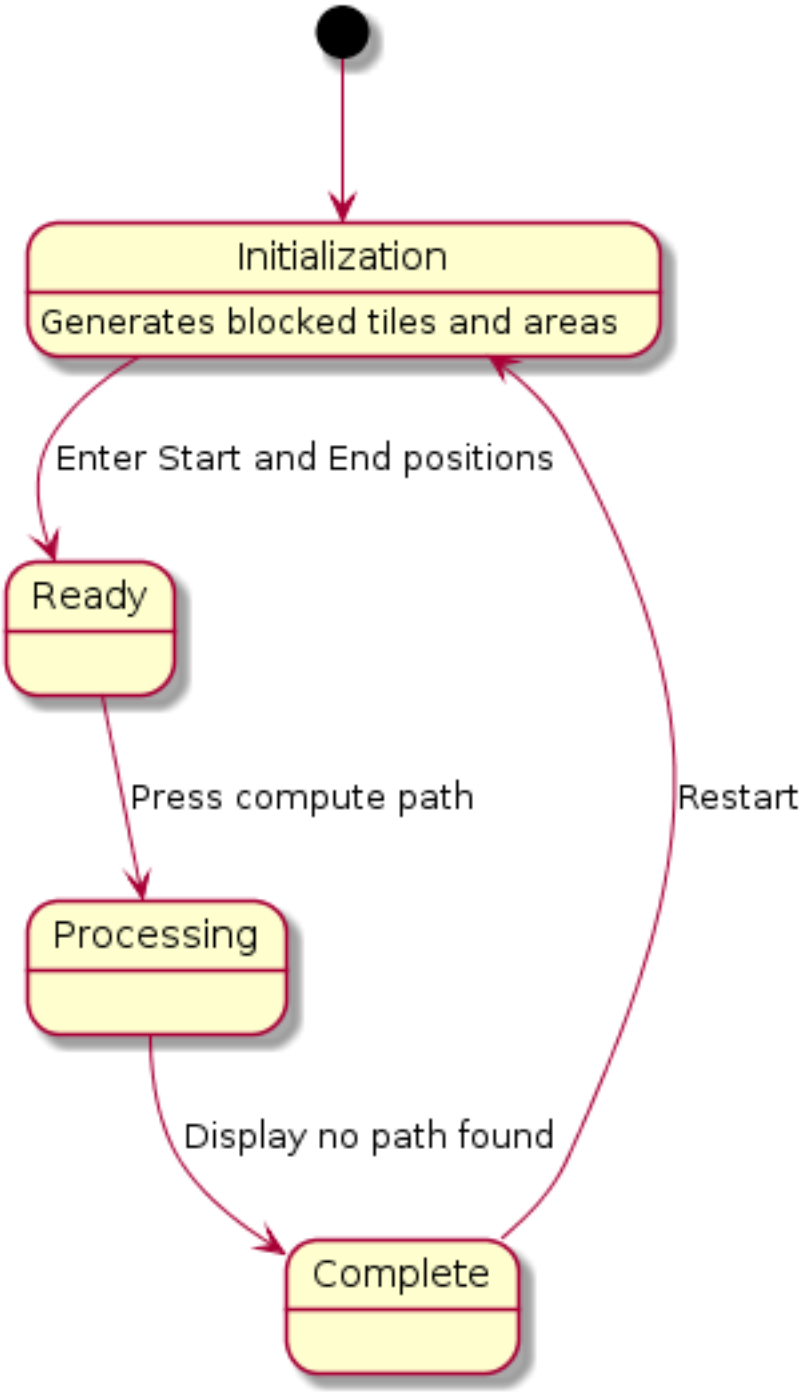
6.2.7.1 Path Found



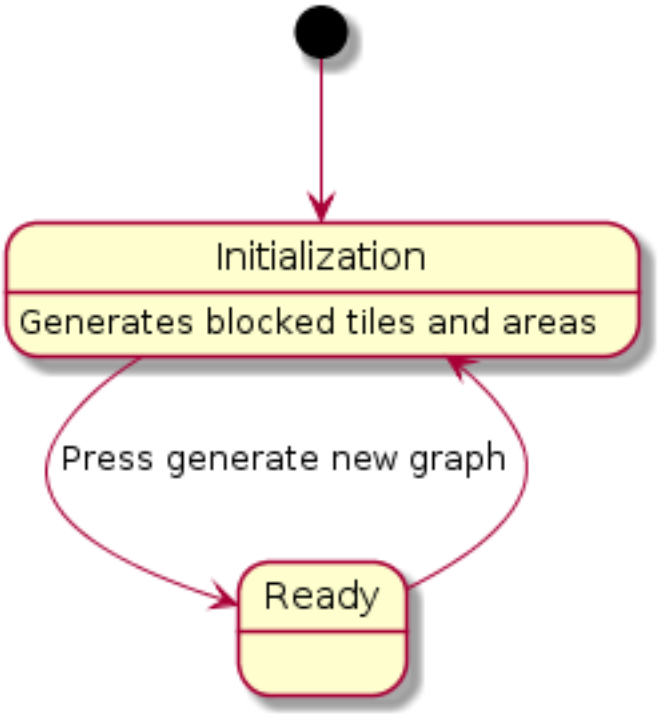
6.2.7.2 Computer Generated Path found



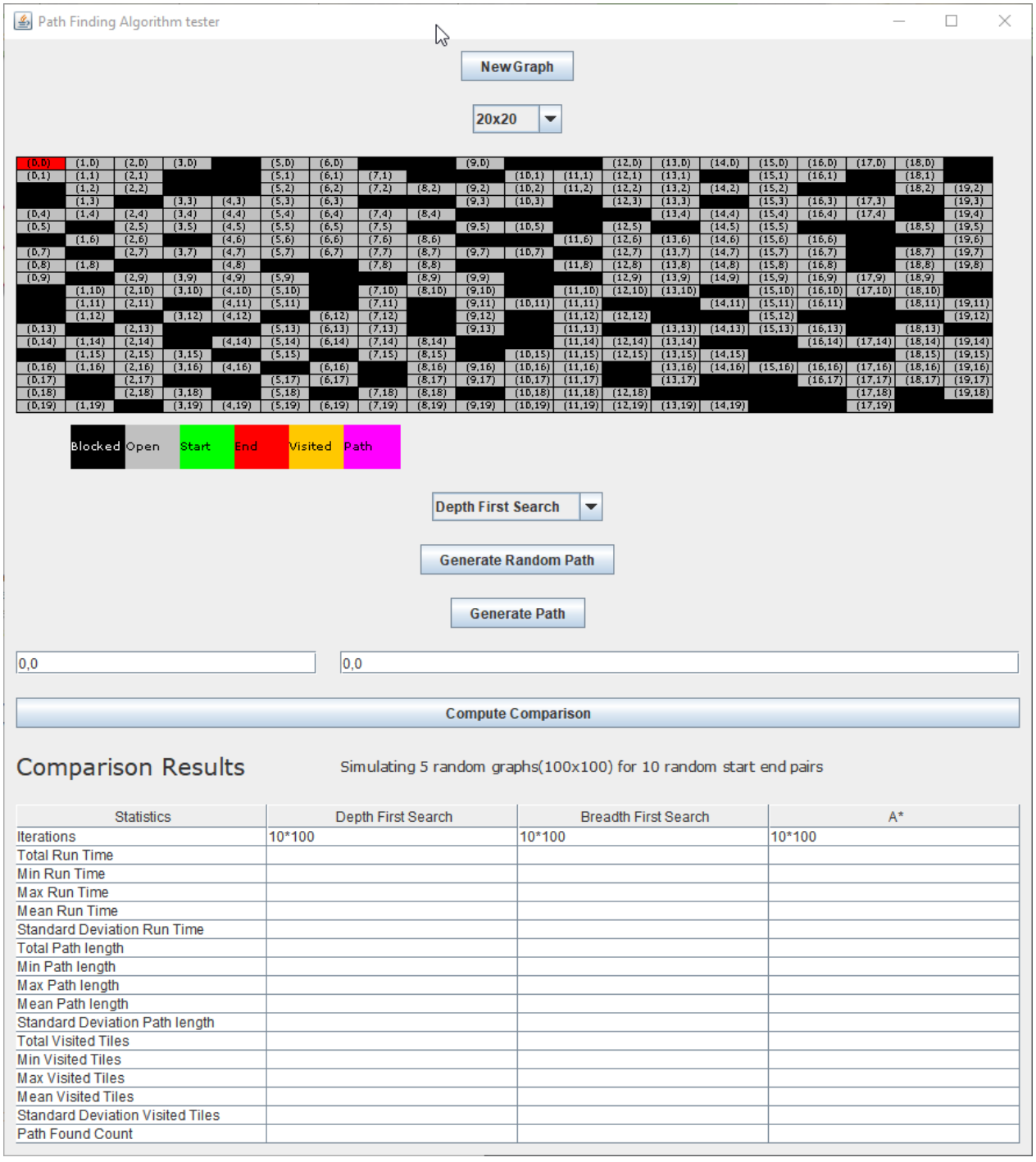
6.2.7.3 Path not found found

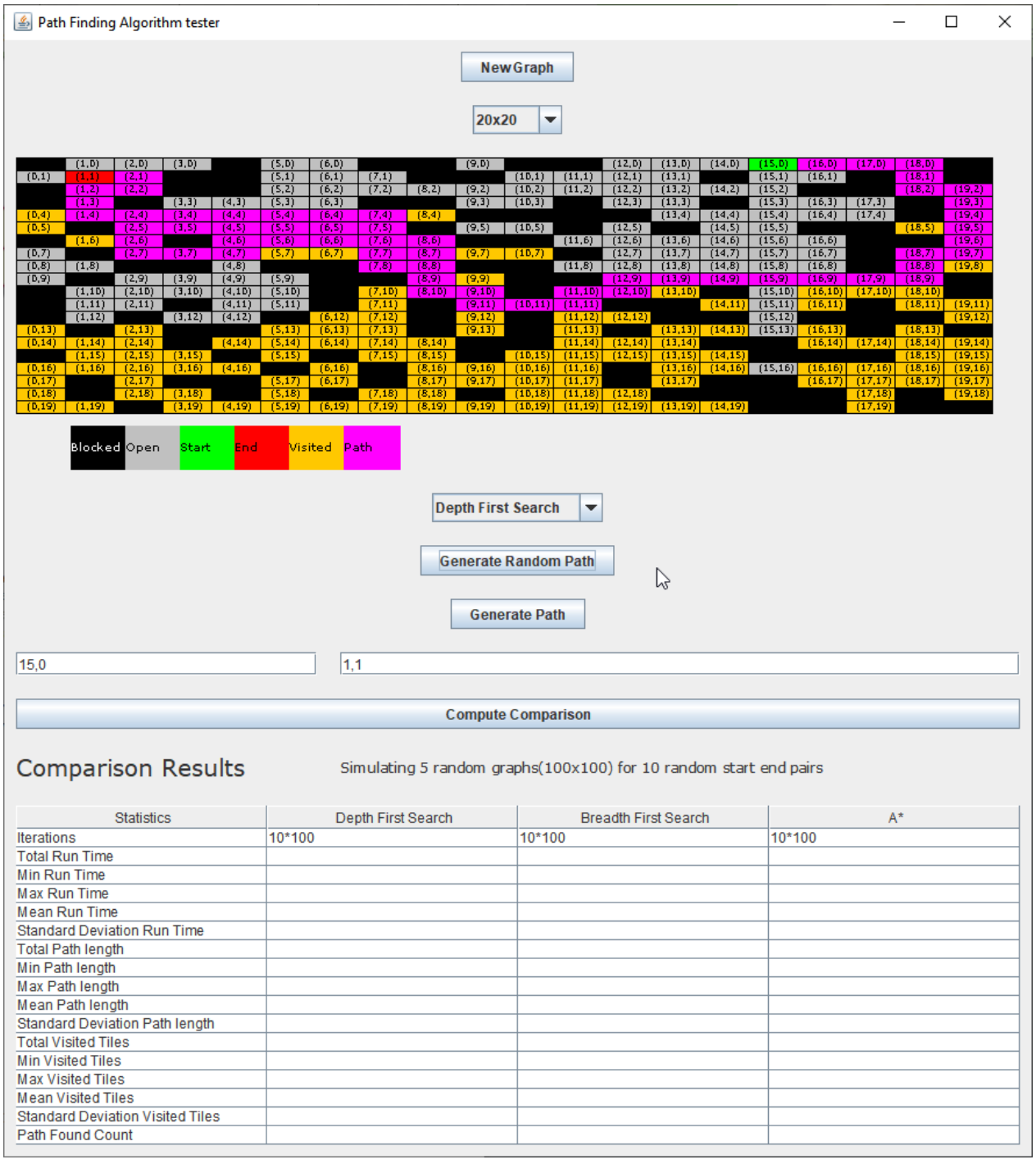


6.2.7.4 Generate new Graph


Initialization
Generates blocked tiles and areas
Press generate new graph
Ready

7 Screenshots

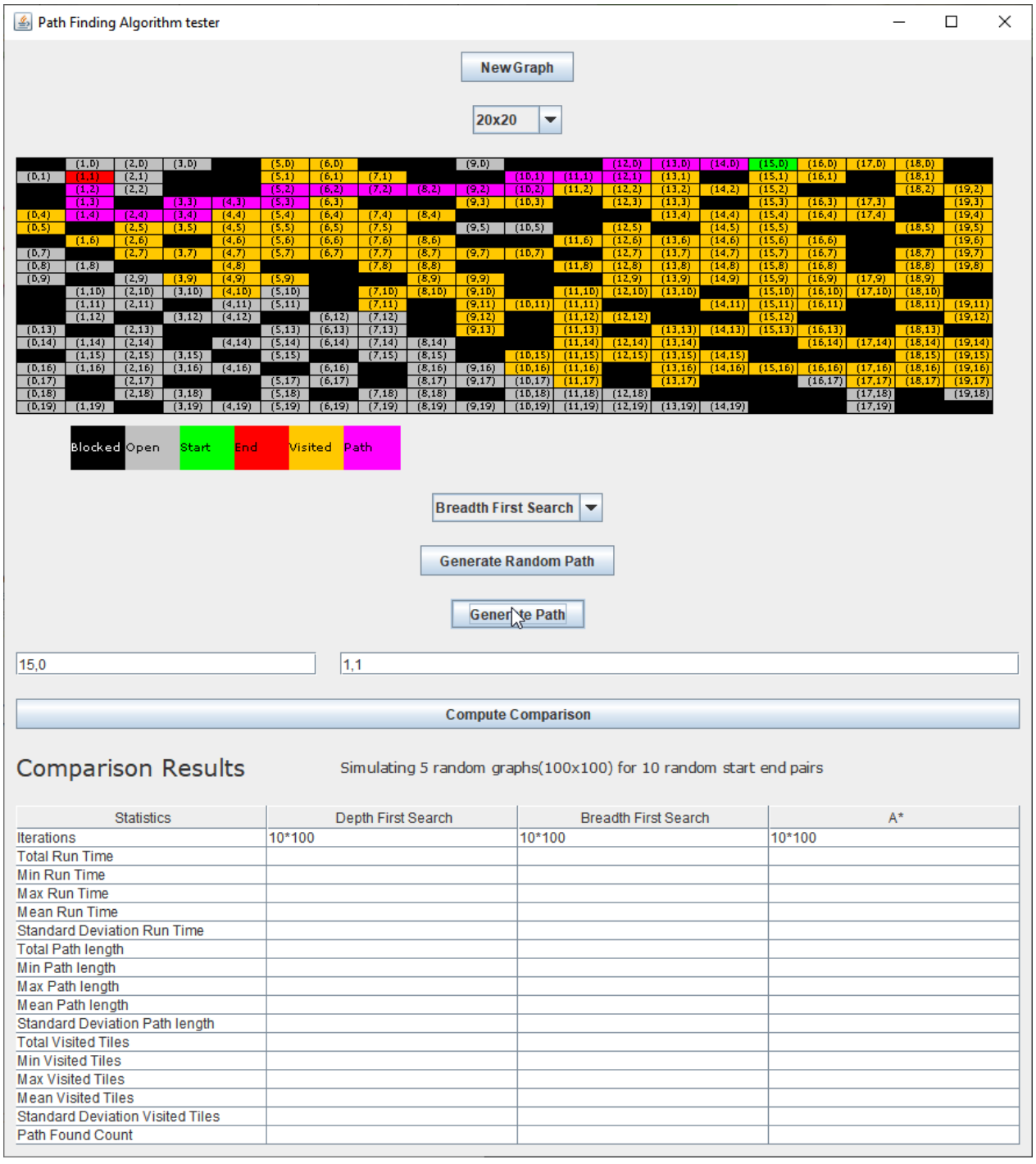


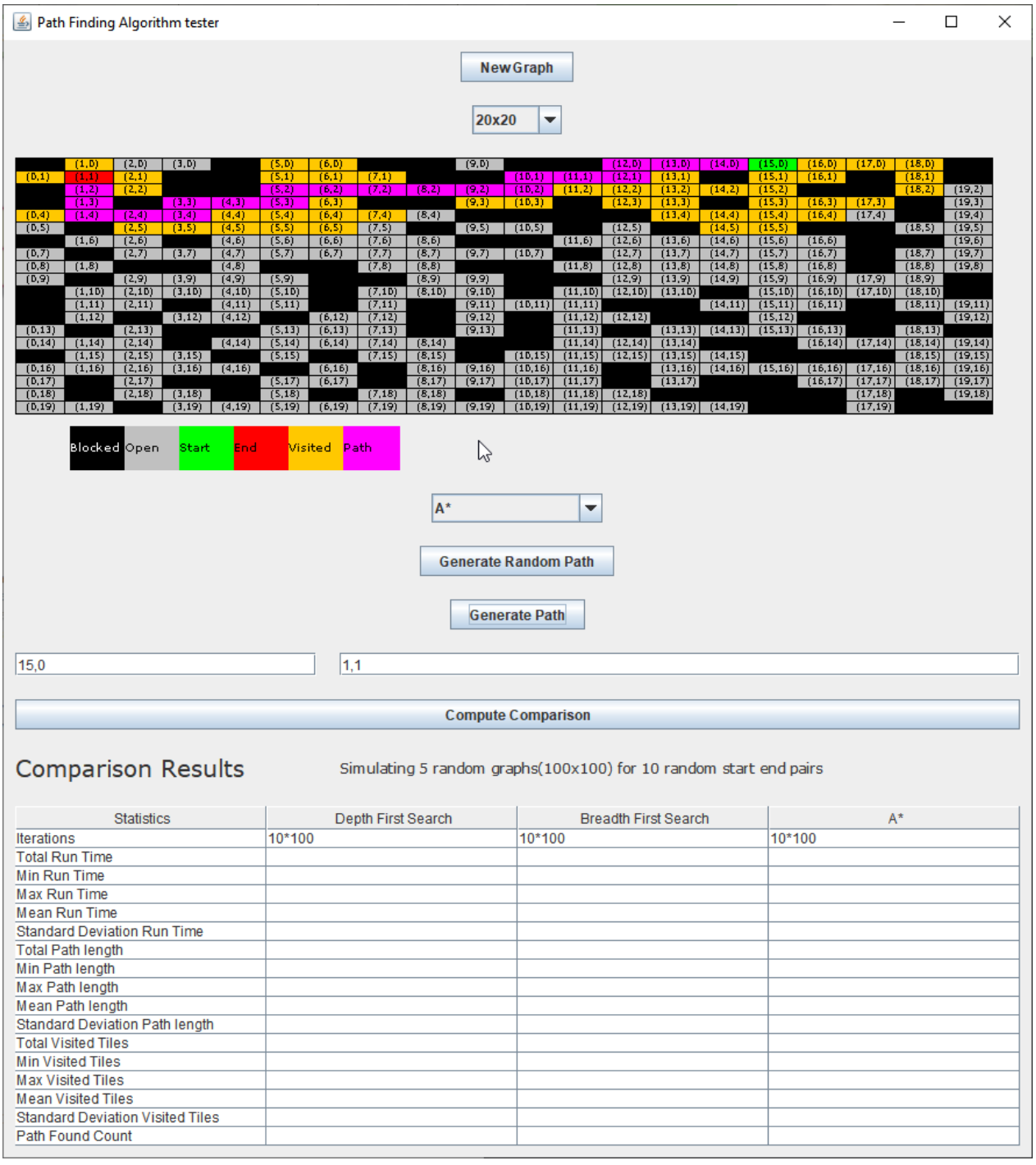


Comparison Results

Simulating 5 random graphs(100x100) for 10 random start end pairs

Statistics	Depth First Search	Breadth First Search	A*
Iterations	10*100	10*100	10*100
Total Run Time			
Min Run Time			
Max Run Time			
Mean Run Time			
Standard Deviation Run Time			
Total Path length			
Min Path length			
Max Path length			
Mean Path length			
Standard Deviation Path length			
Total Visited Tiles			
Min Visited Tiles			
Max Visited Tiles			
Mean Visited Tiles			
Standard Deviation Visited Tiles			
Path Found Count			

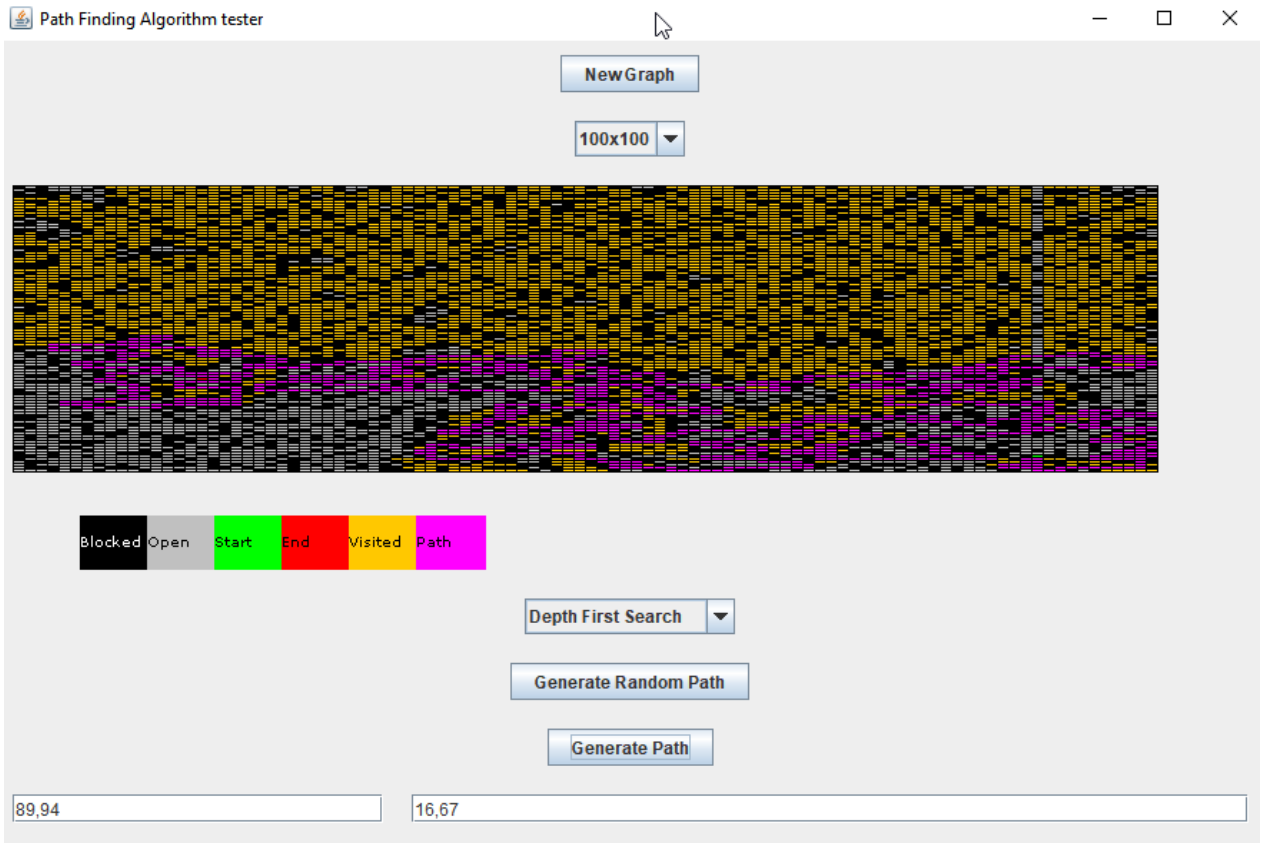

Path Finding Algorithm tester
New Graph
20x20
Blocked
Open
Start
End
Visited
Path
Breadth First Search
Generate Random Path
Generate Path
15,0
1,1
Compute Comparison
Comparison Results
Simulating 5 random graphs(100x100) for 10 random start end pairs
Statistics
Depth First Search
Breadth First Search
A*
Iterations
10*100
10*100
10*100
Total Run Time
Min Run Time
Max Run Time
Mean Run Time
Standard Deviation Run Time
Total Path length
Min Path length
Max Path length
Mean Path length
Standard Deviation Path length
Total Visited Tiles
Min Visited Tiles
Max Visited Tiles
Mean Visited Tiles
Standard Deviation Visited Tiles
Path Found Count

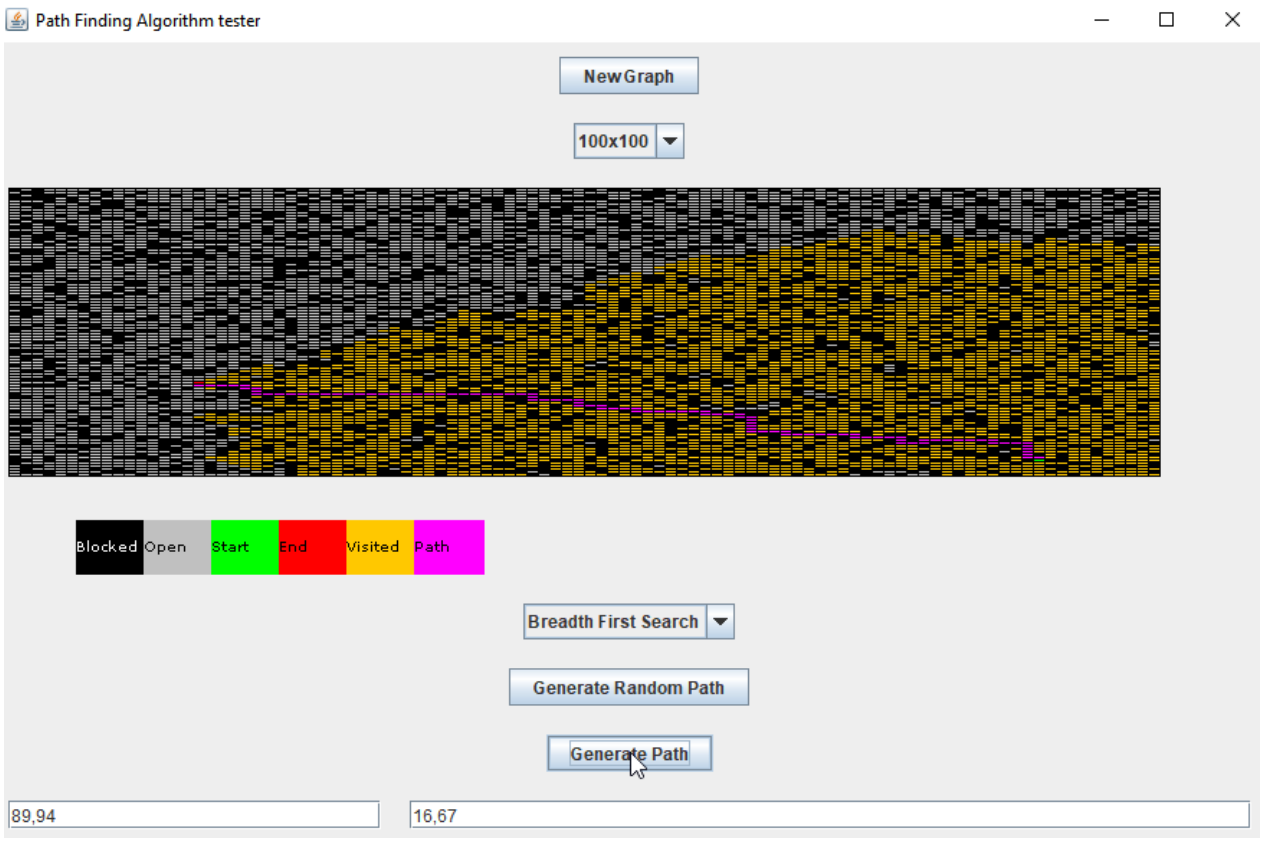


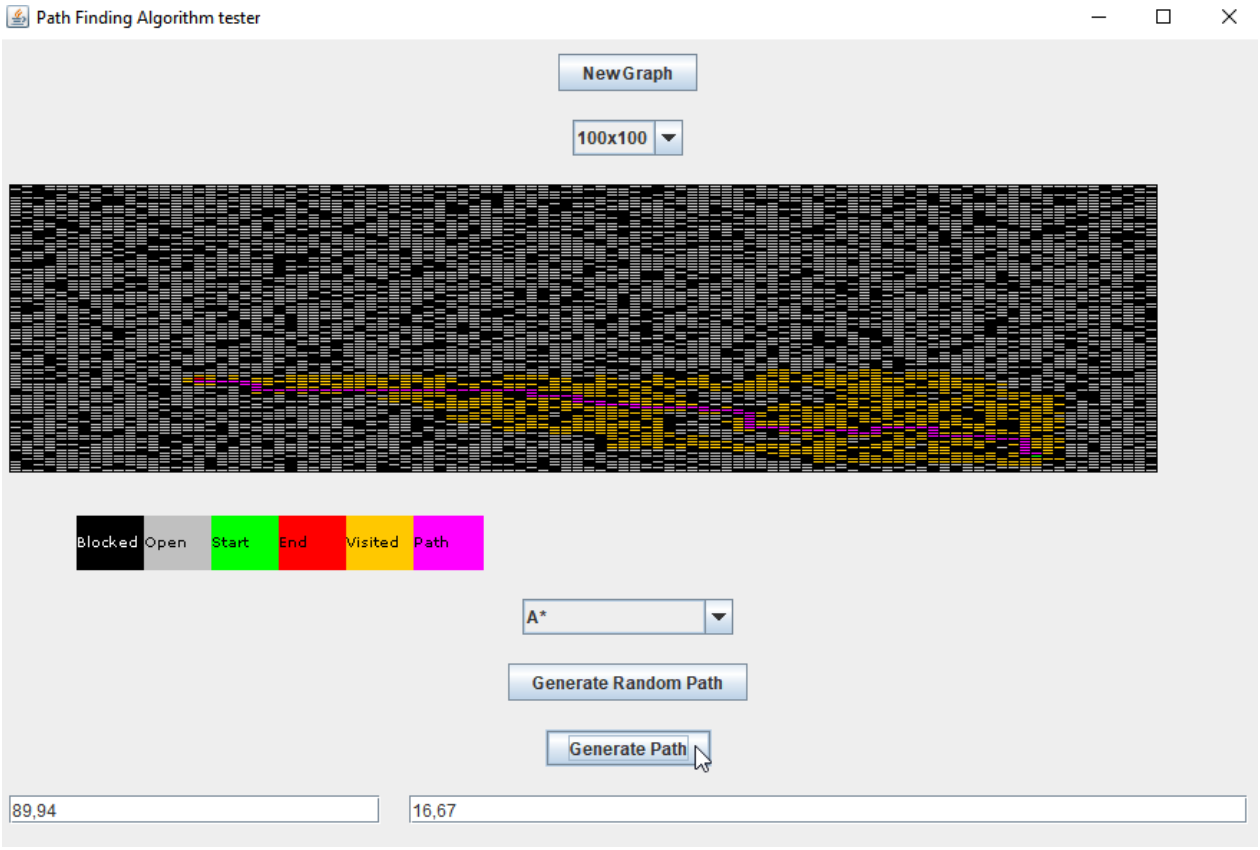
Comparison Results

Simulating 5 random graphs(100x100) for 10 random start end pairs

Statistics	Depth First Search	Breadth First Search	A*
Iterations	10*100	10*100	10*100
Total Run Time			
Min Run Time			
Max Run Time			
Mean Run Time			
Standard Deviation Run Time			
Total Path length			
Min Path length			
Max Path length			
Mean Path length			
Standard Deviation Path length			
Total Visited Tiles			
Min Visited Tiles			
Max Visited Tiles			
Mean Visited Tiles			
Standard Deviation Visited Tiles			
Path Found Count			


Path Finding Algorithm tester
New Graph
100x100
Blocked
Open
Start
End
Visited
Path
Depth First Search
Generate Random Path
Generate Path
89,94
16,67


Path Finding Algorithm tester
New Graph
100x100
Blocked
Open
Start
End
Visited
Path
Breadth First Search
Generate Random Path
Generate Path
89,94
16,67



Compute Comparison

Comparison Results

Simulating 5 random graphs(100x100) for 10 random start end pairs

Statistics	Depth First Search	Breadth First Search	A*
Iterations	10*100	10*100	10*100
Total Run Time	325	353	4,689
Min Run Time	0	1	0
Max Run Time	39	33	448
Mean Run Time	6.5	7.06	93.78
Standard Deviation Run Time	7.114	7.173	125.641
Total Path length	31,940	2,180	2,196
Min Path length	0	0	0
Max Path length	2,080	141	145
Mean Path length	638.8	43.6	43.92
Standard Deviation Path length	644.646	38.728	39.056
Total Visited Tiles	215,350	207,551	142,608
Min Visited Tiles	96	989	108
Max Visited Tiles	6,797	6,929	6,929
Mean Visited Tiles	4,307	4,151.02	2,852.16
Standard Deviation Visited Tiles	2,433.721	2,362.927	3,037.775
Path Found Count	64	64	64

8 Code

8.1 Algorithms Package

8.1.1 Algorithm.java

```
1 package com.pathfinding.algorithms;
2
```

```
3  import com.pathfinding.model.GridModel;
4  import com.pathfinding.model.Path;
5  import com.pathfinding.model.Tile;
6
7  /**
8   * This interface is use to set the infrastructure for all Algorithm types
9   */
10 public interface Algorithm {
11     String getName();
12
13     Path computeOptimalPath(Tile start, Tile end, GridModel gridModel);
14 }
```

8.1.2 AStar.java

```
1  package com.pathfinding.algorithms;
2
3  import com.pathfinding.model.GridModel;
4  import com.pathfinding.model.GridTile;
5  import com.pathfinding.model.Path;
6  import com.pathfinding.model.Tile;
7
8  import java.util.ArrayList;
9  import java.util.Collections;
10 import java.util.List;
11
12 /**
13  * This Algorithm will implment the A* path finding method. Its suppose to be the most
       efficent of the set.
14  */
15 public class AStar implements Algorithm {
16
17     ArrayList<AStarGridTile> openList = new ArrayList<>();
18     ArrayList<AStarGridTile> closedList = new ArrayList<>();
19     ArrayList<GridTile> path = new ArrayList<>();
20     AStarGridTile currentTile;
21     Tile end;
22
23     /**
24      * Looks in a list to see if tile is found
25      *
26      * @param array list of tiles
27      * @param tile  request tile to see if in list
28      * @return true == in list, false == not in list
29      */
30     private static boolean findNeighborInList(List<AStarGridTile> array, AStarGridTile
           tile) {
31         for (AStarGridTile listTile : array) {
32             if (listTile.x == tile.x && listTile.y == tile.y) {
33                 return true;
34             }
35         }
36         return false;
37     }
38
```

```
39      @Override
40      public String getName() {
41          return "A*";
42      }
43
44      /**
45       * This method will find the path between start and end tiles
46       * <p>
47       * Reference to this source: https://rosettacode.org/wiki/A*_search_algorithm#Java
48       *
49       * @param start     - which tile to start from
50       * @param end       - which tile to go to and finish
51       * @param gridModel - The predefined model that the algo will run on
52       * @return - The path between start and end. If no path found it will return an
             empty list
53       * @precondition - gridModel is expected to be already initialized.
54       */
55      @Override
56      public Path computeOptimalPath(Tile start, Tile end, GridModel gridModel) {
57          //Clear out all instance variables between different runs
58          openList.clear();
59          closedList.clear();
60          path.clear();
61          gridModel.resetGraph();
62
63          this.end = end;
64          currentTile = new AStarGridTile(null, gridModel.tiles.get(start.getID()), 0, 0);
65          closedList.add(currentTile);
66          addNeighborsToOpenList(gridModel);
67
68          while (this.currentTile.x != this.end.x || this.currentTile.y != this.end.y) {
69              if (this.openList.isEmpty()) { // Nothing to examine
70                  return new Path();
71              }
72              this.currentTile = this.openList.get(0); // get first node (lowest f score)
73              this.openList.remove(0); // remove it
74              this.closedList.add(this.currentTile); // and add to the closed
75              addNeighborsToOpenList(gridModel);
76          }
77          this.path.add(0, this.currentTile);
78          while (this.currentTile.x != start.x || this.currentTile.y != start.y) {
79              this.currentTile = this.currentTile.parent;
80              this.path.add(0, this.currentTile);
81          }
82          return new Path(this.path);
83
84      }
85
86      /**
87       * This method will add all the tiles around the current tile
88       *
89       * @param gridModel used to look at dimentions of the graph to ensure we dont go out
             of bounds
90       */
```

```
91      private void addNeighborsToOpenList(GridModel gridModel) {
92          AStarGridTile gridTile;
93          for (int x = -1; x <= 1; x++) {
94              for (int y = -1; y <= 1; y++) {
95                  int observedX = x + currentTile.x;
96                  int observedY = y + currentTile.y;
97
98                  if (x != 0 && y != 0) {
99                      continue; // skip if diagonal movement
100                 }
101
102                 if (((x != 0 || y != 0) // not this.currentTile
103                         && this.currentTile.x + x >= 0 && this.currentTile.x + x <
                                gridModel.widthSize // check maze boundaries
104                         && this.currentTile.y + y >= 0 && this.currentTile.y + y <
                                gridModel.heightSize)) {
105                     gridTile = new AStarGridTile(currentTile,
                            gridModel.tiles.get(observedX + "," + observedY), currentTile.g,
                            distance(x, y));
106
107                     if (gridTile.collisionFlag == GridTile.FREE // check if square is
                            walkable
108                             && !findNeighborInList(this.openList, gridTile) &&
                                !findNeighborInList(this.closedList, gridTile)) { // if
                                not already done
109
110                         gridTile.g = gridTile.parent.g + 1.; // Horizontal/vertical cost
                                = 1.0
111                         gridModel.tiles.get(observedX + "," + observedY).visited = true;
112                         this.openList.add(gridTile);
113                     }
114                 }
115             }
116         }
117         Collections.sort(this.openList);
118     }
119
120     /**
121      * Calculate distance between this.currentTile and xend/yend
122      *
123      * @param dx destination x
124      * @param dy destination y
125      * @return the distance between tile(dx,dy) and the current tile
126      */
127     private double distance(int dx, int dy) {
128         return Math.abs(this.currentTile.x + dx - this.end.x) +
                Math.abs(this.currentTile.y + dy - this.end.y);
129     }
130
131     /**
132      * Wrapper class to GridTile to support A* computations to save off g and h
133      */
134     static class AStarGridTile extends GridTile implements Comparable {
135         public AStarGridTile parent;
```

```
        public double g;
        public double h;

        public AStarGridTile(AStarGridTile parent, GridTile tile, double g, double h) {
            super(tile);
            this.collisionFlag = tile.collisionFlag;
            this.parent = parent;
            this.g = g;
            this.h = h;
        }

        /**
         * Compare by f value (g + h)
         *
         * @param o input tile to compute if its a good tile to look at
         * @return - f
         */
        //
        @Override
        public int compareTo(Object o) {
            AStarGridTile that = (AStarGridTile) o;
            return (int) ((this.g + this.h) - (that.g + that.h));
        }
    }
}
```

8.1.3 BDS.java

```
package com.pathfinding.algorithms;

import com.pathfinding.model.GridModel;
import com.pathfinding.model.GridTile;
import com.pathfinding.model.Path;
import com.pathfinding.model.Tile;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

/**
 * Implementation of Breadth First Search
 */
public class BFS implements Algorithm {

    @Override
    public String getName() {
        return "Breadth First Search";
    }

    /**
     * BFS takes a look at all the squares around given square before looking at any
         deeper squares.
     *
     * @param start     - which tile to start from
     * @param end       - which tile to go to and finish
```

```
     * @param gridModel - The predefined model that the algo will run on
     * @return - The path between start and end. If no path found it will return an
         empty list
     * @precondition - gridModel is expected to be already initialized.
     */
    @Override
    public Path computeOptimalPath(Tile start, Tile end, GridModel gridModel) {

        gridModel.resetGraph();
        Queue<GridTile> queue = new LinkedList<>();
        queue.add(gridModel.tiles.get(start.getID()));
        int level = 0;

        while (!queue.isEmpty()) {
            Tile pos = queue.remove();
            int row = pos.x;
            int col = pos.y;

            GridTile currentTile = gridModel.tiles.get(row + "," + col);
            if (row < 0 || col < 0 || row >= gridModel.heightSize || col >=
                gridModel.widthSize || currentTile.visited)
                continue;

            //Break out of loop if we found our end tile
            if (currentTile.x == end.x && currentTile.y == end.y) {
                ArrayList<GridTile> tilesForPath = new ArrayList<>();
                GridTile parent = currentTile.parent;
                tilesForPath.add(currentTile);
                while (parent != null) {
                    tilesForPath.add(parent);
                    parent = parent.parent;
                }
                queue.clear();
                return new Path(tilesForPath);
            }
            //Ensure to not set visited on start or end tiles
            if (currentTile.collisionFlag == GridTile.FREE && !(currentTile.x == start.x
                && currentTile.y == start.y)) {
                currentTile.visited = true;
            }

            // Only push items on the stack if they are free
            if (col > 0) {
                GridTile left = gridModel.tiles.get(row + "," + (col - 1));
                if (left.collisionFlag == GridTile.FREE) {
                    try {
                        left.parent = currentTile.clone();
                    } catch (CloneNotSupportedException e) {
                        e.printStackTrace();
                    }
                    queue.add(left); //go left

                }
            }
```

```
78
79              if (col + 1 < gridModel.widthSize) {
80                  GridTile right = gridModel.tiles.get(row + "," + (col + 1));
81                  if (right.collisionFlag == GridTile.FREE) {
82                      try {
83                          right.parent = currentTile.clone();
84                      } catch (CloneNotSupportedException e) {
85                          e.printStackTrace();
86                      }
87                      queue.add(right); //go right
88                  }
89              }
90
91              if (row > 0) {
92                  GridTile up = gridModel.tiles.get((row - 1) + "," + col);
93                  if (up.collisionFlag == GridTile.FREE) {
94                      try {
95                          up.parent = currentTile.clone();
96                      } catch (CloneNotSupportedException e) {
97                          e.printStackTrace();
98                      }
99                      queue.add(up); //go up
100                 }
101             }
102
103             if (row + 1 < gridModel.heightSize) {
104                 GridTile down = gridModel.tiles.get((row + 1) + "," + col);
105                 if (down.collisionFlag == GridTile.FREE) {
106                     try {
107                         down.parent = currentTile.clone();
108                     } catch (CloneNotSupportedException e) {
109                         e.printStackTrace();
110                     }
111                     queue.add(down); //go down
112                 }
113             }
114         }
115         return new Path(new ArrayList<>());
116     }
117
118 }
```

8.1.4 DFS.java

```
1  package com.pathfinding.algorithms;
2
3  import com.pathfinding.model.GridModel;
4  import com.pathfinding.model.GridTile;
5  import com.pathfinding.model.Path;
6  import com.pathfinding.model.Tile;
7
8  import java.util.ArrayList;
9  import java.util.Stack;
10
11 /**
```

```java
 * Implementation of Depth First Search
 */
public class DFS implements Algorithm {

    @Override
    public String getName() {
        return "Depth First Search";
    }

    /**
     * Another resource:
         https://medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40
     * 1. Add root node to the stack.
     * 2. Loop on the stack as long as it's not empty.
     * 1. Get the node at the top of the stack(current), mark it as visited, and remove
         it.
     * 2. For every non-visited child of the current node, do the following:
     * 1. Check if it's the goal node, If so, then return this child node.
     * 2. Otherwise, push it to the stack.
     * 3. If stack is empty, then goal node was not found!
     *
     * @param start     - which tile to start from
     * @param end       - which tile to go to and finish
     * @param gridModel - The predefined model that the algo will run on
     * @return - The path between start and end. If no path found it will return an
         empty list
     * @precondition - gridModel is expected to be already initialized.
     */
    @Override
    public Path computeOptimalPath(Tile start, Tile end, GridModel gridModel) {
        gridModel.resetGraph();
        boolean[][] visited = new boolean[gridModel.heightSize][gridModel.widthSize];

        Stack<Tile> stack = new Stack<>();
        stack.push(start);

        while (!stack.empty()) {
            Tile pos = stack.pop();
            int row = pos.x;
            int col = pos.y;

            if (row < 0 || col < 0 || row >= gridModel.heightSize || col >=
                gridModel.widthSize || visited[row][col])
                continue;

            visited[row][col] = true;
            GridTile currentTile = gridModel.tiles.get(row + "," + col);


            //Break out of loop if we found our end tile
            if (currentTile.x == end.x && currentTile.y == end.y) {
                ArrayList<GridTile> tilesForPath = new ArrayList<>();
                GridTile parent = currentTile.parent;
                tilesForPath.add(currentTile);
```

```
62                  while (parent != null) {
63                      tilesForPath.add(parent);
64                      parent = parent.parent;
65                  }
66                  stack.clear();
67                  return new Path(tilesForPath);
68              }
69              //Ensure to not set visited on start or end tiles
70              if (currentTile.collisionFlag == GridTile.FREE && (currentTile.x != start.x
                    && currentTile.y != start.y))
71                  currentTile.visited = true;
72
73
74              // Only push items on the stack if they are free
75              if (col > 0) {
76                  GridTile left = gridModel.tiles.get(row + "," + (col - 1));
77                  if (left.collisionFlag == GridTile.FREE) {
78                      try {
79                          left.parent = currentTile.clone();
80                      } catch (CloneNotSupportedException e) {
81                          e.printStackTrace();
82                      }
83                      stack.push(left); //go left
84                  }
85              }
86
87              if (col + 1 < gridModel.widthSize) {
88                  GridTile right = gridModel.tiles.get(row + "," + (col + 1));
89                  if (right.collisionFlag == GridTile.FREE) {
90                      try {
91                          right.parent = currentTile.clone();
92                      } catch (CloneNotSupportedException e) {
93                          e.printStackTrace();
94                      }
95                      stack.push(right); //go right
96                  }
97              }
98
99              if (row > 0) {
100                 GridTile up = gridModel.tiles.get((row - 1) + "," + col);
101                 if (up.collisionFlag == GridTile.FREE) {
102                     try {
103                         up.parent = currentTile.clone();
104                     } catch (CloneNotSupportedException e) {
105                         e.printStackTrace();
106                     }
107                     stack.push(up); //go up
108
109                 }
110             }
111
112             if (row + 1 < gridModel.heightSize) {
113                 GridTile down = gridModel.tiles.get((row + 1) + "," + col);
114                 if (down.collisionFlag == GridTile.FREE) {
```

```
                    try {
                        down.parent = currentTile.clone();
                    } catch (CloneNotSupportedException e) {
                        e.printStackTrace();
                    }
                    stack.push(down); //go down
                }
            }


        }
        return null;
    }

}
```

8.2 Controller package

8.2.1 AlgorithmController.java

```
package com.pathfinding.controller;

import com.pathfinding.algorithms.AStar;
import com.pathfinding.algorithms.Algorithm;
import com.pathfinding.algorithms.BFS;
import com.pathfinding.algorithms.DFS;
import com.pathfinding.model.AlgorithmStatModel;
import com.pathfinding.model.GridModel;
import com.pathfinding.model.GridTile;
import com.pathfinding.simulation.Simulation;
import com.pathfinding.view.GUIView;

import javax.swing.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.ArrayList;

/**
 * This is the master controller in the MVC pattern
 * It will handle the Observer pasterns between the GUI and Model with the Action
     listeners
 * It will also handle the creation of all the Models and Views
 */
public class AlgorithmController {
    GridModel gridModel;

    ArrayList<Algorithm> algorithms = new ArrayList<>();
    GUIView guiView;
    AlgorithmStatModel algorithmStatModel;
    Simulation sim;

    int selectedAlgoIndex = 0;

    AlgorithmController() {
        //Create gridModel
```

```
35
36          gridModel = new GridModel(20, 20);
37          algorithms.add(new DFS());
38          algorithms.add(new BFS());
39          algorithms.add(new AStar());
40          String[] columnNames = new String[algorithms.size() + 1];
41          columnNames[0] = "Statistics";
42          for (int i = 0; i < algorithms.size(); i++) {
43              columnNames[i + 1] = algorithms.get(i).getName();
44          }
45          algorithmStatModel = new AlgorithmStatModel(columnNames);
46          sim = new Simulation(10, algorithms, 100, 100, algorithmStatModel);
47          guiView = GUIView.getInstance(gridModel, algorithms, algorithmStatModel);
48
49
50          guiView.addALForNewGrid(e -> {
51              guiView.gridModel.newRandomGraph();
52              guiView.grid.invalidate();
53              guiView.grid.validate();
54              guiView.grid.repaint();
55
56              //Reset start and end
57              guiView.gridModel.startPosition.x = 0;
58              guiView.gridModel.startPosition.y = 0;
59              guiView.gridModel.endPosition.x = 0;
60              guiView.gridModel.endPosition.y = 0;
61              guiView.textFieldStartPosition.setText(0 + "," + 0);
62              guiView.textFieldEndPosition.setText(0 + "," + 0);
63          });
64
65          guiView.algorithmDropdown.addActionListener(e -> selectedAlgoIndex =
                guiView.algorithmDropdown.getSelectedIndex());
66
67          guiView.dropdownGridSize.addActionListener(e -> {
68              if (guiView.dropdownGridSize.getSelectedIndex() == 0) {
69                  guiView.gridModel.heightSize = 20;
70                  guiView.gridModel.widthSize = 20;
71              } else {
72                  guiView.gridModel.heightSize = 100;
73                  guiView.gridModel.widthSize = 100;
74              }
75              guiView.gridModel.newRandomGraph();
76              guiView.grid.invalidate();
77              guiView.grid.validate();
78              guiView.grid.repaint();
79          });
80
81          guiView.textFieldStartPosition.addKeyListener(new KeyListener() {
82              @Override
83              public void keyTyped(KeyEvent e) {
84                  checkTextFieldUpdate(guiView.textFieldStartPosition,
                        guiView.gridModel.startPosition);
85                  gridModel.path =
                        algorithms.get(selectedAlgoIndex).computeOptimalPath(gridModel.startPosition,
```

```
                    gridModel.endPosition, gridModel);
            }

            @Override
            public void keyPressed(KeyEvent e) {
                checkTextFieldUpdate(guiView.textFieldStartPosition,
                    guiView.gridModel.startPosition);
                gridModel.path =
                    algorithms.get(selectedAlgoIndex).computeOptimalPath(gridModel.startPosition,
                    gridModel.endPosition, gridModel);
            }

            @Override
            public void keyReleased(KeyEvent e) {
                checkTextFieldUpdate(guiView.textFieldStartPosition,
                    guiView.gridModel.startPosition);
                gridModel.path =
                    algorithms.get(selectedAlgoIndex).computeOptimalPath(gridModel.startPosition,
                    gridModel.endPosition, gridModel);
                guiView.grid.invalidate();
            }
        });

        guiView.textFieldEndPosition.addKeyListener(new KeyListener() {
            @Override
            public void keyTyped(KeyEvent e) {
                checkTextFieldUpdate(guiView.textFieldEndPosition,
                    guiView.gridModel.endPosition);
                gridModel.path =
                    algorithms.get(selectedAlgoIndex).computeOptimalPath(gridModel.startPosition,
                    gridModel.endPosition, gridModel);
                guiView.grid.invalidate();
            }

            @Override
            public void keyPressed(KeyEvent e) {
                checkTextFieldUpdate(guiView.textFieldEndPosition,
                    guiView.gridModel.endPosition);
                gridModel.path =
                    algorithms.get(selectedAlgoIndex).computeOptimalPath(gridModel.startPosition,
                    gridModel.endPosition, gridModel);
                guiView.grid.invalidate();
            }

            @Override
            public void keyReleased(KeyEvent e) {
                checkTextFieldUpdate(guiView.textFieldEndPosition,
                    guiView.gridModel.endPosition);
                gridModel.path =
                    algorithms.get(selectedAlgoIndex).computeOptimalPath(gridModel.startPosition,
                    gridModel.endPosition, gridModel);
                guiView.grid.invalidate();
            }
        });
```

```
124
125         guiView.addALForGenerateRandomStartEndPositions(e -> {
126             gridModel.newRandomStartAndEndPositions();
127             guiView.textFieldStartPosition.setText(gridModel.startPosition.x + "," +
                    gridModel.startPosition.y);
128             guiView.textFieldEndPosition.setText(gridModel.endPosition.x + "," +
                    gridModel.endPosition.y);
129
130             gridModel.path =
                    algorithms.get(selectedAlgoIndex).computeOptimalPath(gridModel.startPosition,
                    gridModel.endPosition, gridModel);
131             //update view
132             guiView.grid.invalidate();
133             guiView.grid.validate();
134             guiView.grid.repaint();
135         });
136
137         guiView.addALForButtonGeneratePath(e -> {
138             gridModel.path =
                    algorithms.get(selectedAlgoIndex).computeOptimalPath(gridModel.startPosition,
                    gridModel.endPosition, gridModel);
139             //update view
140             guiView.grid.invalidate();
141             guiView.grid.validate();
142             guiView.grid.repaint();
143         });
144
145         guiView.addALForButtonRunComparison(e -> {
146             sim.run();
147             algorithmStatModel.computeStatistic();
148             guiView.tableComparisonTable.invalidate();
149             guiView.tableComparisonTable.validate();
150             guiView.tableComparisonTable.repaint();
151         });
152
153     }
154
155     public static void main(String[] args) {
156         AlgorithmController algorithmController = new AlgorithmController();
157
158     }
159
160     /**
161      * This parses the change from the text field, parses to make sure we have correct
             coordinates. It will catch any
162      * incorrect patterns of "int,int" and print to the console. Once parsed it will
             then update the model and then
163      * re-validate the View
164      *
165      * @param jTextField input filed to parse
166      * @param position   the model to update
167      */
168     private void checkTextFieldUpdate(JTextField jTextField, GridTile position) {
169         String fieldText = jTextField.getText();
```

```
        String[] split = fieldText.split(",");
        if (split.length == 2) {
            try {
                int x = Integer.parseInt(split[0]);
                int y = Integer.parseInt(split[1]);
                position.x = x;
                position.y = y;
                guiView.grid.invalidate();
                guiView.grid.validate();
                guiView.grid.repaint();
            } catch (NumberFormatException e) {
                e.printStackTrace();
                System.out.println("Error: parsing integer");
            }
        } else {
            System.out.println("Error: parsing the text filed: " + fieldText);
        }
    }

}
```

8.3 Model package

8.3.1 AlgorithmResult.java

```
package com.pathfinding.model;

/**
 * Simple class to represent the data to be collected when comparing Algorithms
 */
public class AlgorithmResult {
    public String algorithmName;
    public int runtime;
    public int pathLength;
    public boolean pathFound;
    public int visitedTileCount;

    public AlgorithmResult(String algorithmName, int runtime, int pathLength, boolean
        pathFound, int visitedTileCount) {

        this.algorithmName = algorithmName;
        this.runtime = runtime;
        this.pathLength = pathLength;
        this.pathFound = pathFound;
        this.visitedTileCount = visitedTileCount;
    }

}
```

8.3.2 AlgorithmStatModel.java

```
package com.pathfinding.model;

import javax.swing.table.AbstractTableModel;
import java.text.DecimalFormat;
import java.util.ArrayList;
```

```java
import java.util.HashMap;

/**
 * Class to represent the model to compare the different algorithms. This is a Model in the MVC pattern
 * when an update occurs, the GUI Jtable will automatically update
 * Pattern : Adapter pattern
 */
public class AlgorithmStatModel extends AbstractTableModel {
    public HashMap<String, Integer> totalRunTime = new HashMap<>(); // Key is the algorithm name
    public HashMap<String, Integer> minRunTime = new HashMap<>();
    public HashMap<String, Integer> maxRunTime = new HashMap<>();
    public HashMap<String, Double> meanRunTime = new HashMap<>();
    public HashMap<String, Double> stdDevRunTime = new HashMap<>();

    public HashMap<String, Integer> totalPathLength = new HashMap<>();
    public HashMap<String, Integer> maxPathLength = new HashMap<>();
    public HashMap<String, Integer> minPathLength = new HashMap<>();
    public HashMap<String, Double> meanPathLength = new HashMap<>();
    public HashMap<String, Double> stdDevPathLength = new HashMap<>();

    public HashMap<String, Integer> totalVisitedTileCount = new HashMap<>();
    public HashMap<String, Integer> minVisitedTileCount = new HashMap<>();
    public HashMap<String, Integer> maxVisitedTileCount = new HashMap<>();
    public HashMap<String, Double> meanVisitedTileCount = new HashMap<>();
    public HashMap<String, Double> stdDevVisitedTileCount = new HashMap<>();

    public HashMap<String, Integer> totalPathFound = new HashMap<>();

    ArrayList<AlgorithmResult> latestAlgorithmStats = new ArrayList<>();

    String[] headers;
    private Object[][] data = {
            {"Iterations", "5*10", "5*10", "5*10", "10*100"},

            {"Total Run Time", "", "", "", ""},
            {"Min Run Time", "", "", "", ""},
            {"Max Run Time", "", "", "", ""},
            {"Mean Run Time", "", "", "", ""},
            {"Standard Deviation Run Time", "", "", "", ""},

            {"Total Path length", "", "", "", ""},
            {"Min Path length", "", "", "", ""},
            {"Max Path length", "", "", "", ""},
            {"Mean Path length", "", "", "", ""},
            {"Standard Deviation Path length", "", "", "", ""},

            {"Total Visited Tiles", "", "", "", ""},
            {"Min Visited Tiles", "", "", "", ""},
            {"Max Visited Tiles", "", "", "", ""},
            {"Mean Visited Tiles", "", "", "", ""},
            {"Standard Deviation Visited Tiles", "", "", "", ""},

```

```
58              {"Path Found Count", "", "", "", ""},
59
60      };
61
62      public AlgorithmStatModel(String[] headers) {
63          this.headers = headers;
64      }
65
66      public void addAlgorithm(AlgorithmResult algorithmResult) {
67          latestAlgorithmStats.add(algorithmResult);
68      }
69
70      /**
71       * This will run all the computations for the given algorithm results for the
              following
72       * 1. Total runtime
73       * 2. min runtime
74       * 3. max runtime
75       * 4. mean runtime
76       * 5. standard deviation runtime
77       * <p>
78       * 1. Total Path length
79       * 2. min Path length
80       * 3. max Path length
81       * 4. mean Path length
82       * 5. standard deviation Path length
83       * <p>
84       * 1. Total Tiles visited
85       * 2. min Tiles visited
86       * 3. max Tiles visited
87       * 4. mean Tiles visited
88       * 5. standard deviation Tiles visited
89       *
90       * @Precondition - Expected to have at least 1 algorithm results
91       * @Postcondition - the results will be computed and stored in the hashmaps. The
              JTable which this model is
92       * adapted too will also update automatically.
93       */
94      public void computeStatistic() {
95
96          for (AlgorithmResult algorithmResult : latestAlgorithmStats) {
97              String algorithmName = algorithmResult.algorithmName;
98
99              //update totals, ensure they exist first
100             if (!totalPathFound.containsKey(algorithmName)) {
101                 totalPathFound.put(algorithmName, algorithmResult.pathFound ? 1 : 0);
102                 totalVisitedTileCount.put(algorithmName,
                        algorithmResult.visitedTileCount);
103                 totalPathLength.put(algorithmName, algorithmResult.pathLength);
104                 totalRunTime.put(algorithmName, algorithmResult.runtime);
105             } else {
106                 totalPathFound.put(algorithmName, totalPathFound.get(algorithmName) +
                        (algorithmResult.pathFound ? 1 : 0));
107                 totalVisitedTileCount.put(algorithmName,
```

```
                    totalVisitedTileCount.get(algorithmName) +
                    algorithmResult.visitedTileCount);
108                 totalPathLength.put(algorithmName, totalPathLength.get(algorithmName) +
                    algorithmResult.pathLength);
109                 totalRunTime.put(algorithmName, totalRunTime.get(algorithmName) +
                    algorithmResult.runtime);
110             }
111
112             //Update min
113             if (!minRunTime.containsKey(algorithmName)) {
114                 minRunTime.put(algorithmName, algorithmResult.runtime);
115                 minPathLength.put(algorithmName, algorithmResult.pathLength);
116                 minVisitedTileCount.put(algorithmName, algorithmResult.visitedTileCount);
117
118             } else {
119                 if (minRunTime.get(algorithmName) > algorithmResult.runtime) {
120                     minRunTime.put(algorithmName, algorithmResult.runtime);
121                 }
122                 if (minPathLength.get(algorithmName) > algorithmResult.pathLength) {
123                     minPathLength.put(algorithmName, algorithmResult.pathLength);
124                 }
125                 if (minVisitedTileCount.get(algorithmName) >
                    algorithmResult.visitedTileCount) {
126                     minVisitedTileCount.put(algorithmName,
                        algorithmResult.visitedTileCount);
127                 }
128             }
129
130             //update max
131             if (!maxRunTime.containsKey(algorithmName)) {
132                 maxRunTime.put(algorithmName, algorithmResult.runtime);
133                 maxPathLength.put(algorithmName, algorithmResult.pathLength);
134                 maxVisitedTileCount.put(algorithmName, algorithmResult.visitedTileCount);
135
136             } else {
137                 if (maxRunTime.get(algorithmName) < algorithmResult.runtime) {
138                     maxRunTime.put(algorithmName, algorithmResult.runtime);
139                 }
140                 if (maxPathLength.get(algorithmName) < algorithmResult.pathLength) {
141                     maxPathLength.put(algorithmName, algorithmResult.pathLength);
142                 }
143                 if (maxVisitedTileCount.get(algorithmName) <
                    algorithmResult.visitedTileCount) {
144                     maxVisitedTileCount.put(algorithmName,
                        algorithmResult.visitedTileCount);
145                 }
146             }
147
148             if (!totalPathFound.containsKey(algorithmName)) {
149                 if (algorithmResult.pathFound) {
150                     totalPathFound.put((algorithmName), 1);
151                 }
152             } else {
153                 if (algorithmResult.pathFound) {
```

```
154                     totalPathFound.put(algorithmName, totalPathFound.get(algorithmName)
                            + 1);
155                 }
156             }
157         }
158         //Compute Statistic
159         for (String algoName : totalRunTime.keySet()) {
160             int currentAlgoTotalRunTime = totalRunTime.get(algoName);
161             int numberOfIterations = latestAlgorithmStats.size() /
                    totalRunTime.keySet().size();
162             double meanRunTime = currentAlgoTotalRunTime / (double) numberOfIterations;
163             this.meanRunTime.put(algoName, meanRunTime);
164 
165             int currentAlgoTotalPathLength = totalPathLength.get(algoName);
166             double meanPathLength = currentAlgoTotalPathLength / (double)
                    numberOfIterations;
167             this.meanPathLength.put(algoName, meanPathLength);
168 
169             int currentAlgoTotalVisitedTileCount = totalVisitedTileCount.get(algoName);
170             double meanVisitedTileCount = currentAlgoTotalVisitedTileCount / (double)
                    numberOfIterations;
171             this.meanVisitedTileCount.put(algoName, meanVisitedTileCount);
172 
173             //Compute standard deviation
174             double standardDeviationRunTime = 0;
175             double standardDeviationPathLength = 0;
176             double standardDeviationVisitedCount = 0;
177             for (AlgorithmResult algorithmResult : latestAlgorithmStats) {
178                 String algorithmName = algorithmResult.algorithmName;
179                 if (algoName.equals(algorithmName)) {
180                     standardDeviationRunTime += Math.pow(algorithmResult.runtime -
                            meanRunTime, 2);
181                     standardDeviationPathLength += Math.pow(algorithmResult.pathLength -
                            meanPathLength, 2);
182                     standardDeviationVisitedCount +=
                            Math.pow(algorithmResult.visitedTileCount - meanVisitedTileCount,
                            2);
183                 }
184             }
185             stdDevPathLength.put(algoName, Math.sqrt(standardDeviationPathLength /
                    numberOfIterations));
186             stdDevRunTime.put(algoName, Math.sqrt(standardDeviationRunTime /
                    numberOfIterations));
187             stdDevVisitedTileCount.put(algoName, Math.sqrt(standardDeviationVisitedCount
                    / numberOfIterations));
188         }
189 
190 
191         // Update the data which will then in turn update and adapted Jtables
192         String pattern = "###,###.###";
193         DecimalFormat decimalFormat = new DecimalFormat(pattern);
194         for (int i = 0; i < headers.length - 1; i++) { // First header is just Statistics
195             int row = 1;
196             data[row++][i + 1] = decimalFormat.format(totalRunTime.get(headers[i + 1]));
```

```
            data[row++][i + 1] = decimalFormat.format(minRunTime.get(headers[i + 1]));
            data[row++][i + 1] = decimalFormat.format(maxRunTime.get(headers[i + 1]));
            data[row++][i + 1] = decimalFormat.format(meanRunTime.get(headers[i + 1]));
            data[row++][i + 1] = decimalFormat.format(stdDevRunTime.get(headers[i + 1]));

            data[row++][i + 1] = decimalFormat.format(totalPathLength.get(headers[i +
                1]));
            data[row++][i + 1] = decimalFormat.format(minPathLength.get(headers[i + 1]));
            data[row++][i + 1] = decimalFormat.format(maxPathLength.get(headers[i + 1]));
            data[row++][i + 1] = decimalFormat.format(meanPathLength.get(headers[i +
                1]));
            data[row++][i + 1] = decimalFormat.format(stdDevPathLength.get(headers[i +
                1]));

            data[row++][i + 1] =
                decimalFormat.format(totalVisitedTileCount.get(headers[i + 1]));
            data[row++][i + 1] = decimalFormat.format(minVisitedTileCount.get(headers[i
                + 1]));
            data[row++][i + 1] = decimalFormat.format(maxVisitedTileCount.get(headers[i
                + 1]));
            data[row++][i + 1] = decimalFormat.format(meanVisitedTileCount.get(headers[i
                + 1]));
            data[row++][i + 1] =
                decimalFormat.format(stdDevVisitedTileCount.get(headers[i + 1]));

            data[row][i + 1] = totalPathFound.get(headers[i + 1]);
        }
    }

    @Override
    public String getColumnName(int col) {
        return headers[col];
    }

    @Override
    public int getRowCount() {
        return data.length;
    }

    @Override
    public int getColumnCount() {
        return headers.length;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        return data[rowIndex][columnIndex];
    }
}
```

8.3.3 GridModel.java

```
package com.pathfinding.model;

import java.util.HashMap;
```

```java
import java.util.Random;

/**
 * This class is used to represent the Model for a 2d graph. It has the ability to generate random graphs
 * and random start and end positions
 */
public class GridModel {
    public GridTile startPosition = new GridTile(0, 0);
    public GridTile endPosition = new GridTile(0, 0);
    public int widthSize;
    public int heightSize;
    public HashMap<String, GridTile> tiles = new HashMap<>(); // Index will be position "x,y"
    public Path path = new Path();

    /**
     * Creates a new graph based on given params
     *
     * @param widthSize  - designated width to create graph
     * @param heightSize - designated height to create graph
     */
    public GridModel(int widthSize, int heightSize) {
        Random rand = new Random();
        for (int y = 0; y < heightSize; y++) {
            for (int x = 0; x < widthSize; x++) {
                GridTile newTile;
                if (rand.nextInt(10) > 2) {
                    newTile = new GridTile(x, y, GridTile.FREE);
                } else {
                    newTile = new GridTile(x, y, GridTile.BLOCKED);
                }
                tiles.put(x + "," + y, newTile);
            }

        }
        this.widthSize = widthSize;
        this.heightSize = heightSize;
    }

    /**
     * @postcondition - completely clears out the graph.
     */
    public void clearGraph() {
        this.tiles.clear();
    }

    /**
     * @Postcondition - updates graph to reshuffle all the tiles to be free or blocked.
     */
    public void newRandomGraph() {
        Random rand = new Random();
        for (int y = 0; y < heightSize; y++) {
            for (int x = 0; x < widthSize; x++) {
```

```
                String key = x + "," + y;
                //Add a new tile if we expand size
                if (tiles.get(key) == null) {
                    tiles.put(key, new GridTile(x, y, GridTile.FREE));
                }
                if (rand.nextInt(10) > 2) {
                    tiles.get(key).collisionFlag = GridTile.FREE;
                } else {
                    tiles.get(key).collisionFlag = GridTile.BLOCKED;

                }
                tiles.get(x + "," + y).resetTile();
            }
        }
        path.clear();
    }

    /**
     * @postcondition - will update the start and end position to ensure that the block
         was previously free
     */
    public void newRandomStartAndEndPositions() {
        Random random = new Random();
        int randStartX, randStartY, randEndX, randEndY;

        //Find free tiles
        do {
            randStartX = random.nextInt(widthSize);
            randStartY = random.nextInt(heightSize);
        } while (tiles.get(randStartX + "," + randStartY).collisionFlag ==
            GridTile.BLOCKED);

        do {
            randEndX = random.nextInt(widthSize);
            randEndY = random.nextInt(heightSize);
        } while (tiles.get(randEndX + "," + randEndY).collisionFlag == GridTile.BLOCKED);


        //Update old model start and end with new positions
        startPosition.x = randStartX;
        startPosition.y = randStartY;
        endPosition.x = randEndX;
        endPosition.y = randEndY;

    }

    /**
     * @postcondition - updates the tiles to clear out historical data
     */
    public void resetGraph() {
        for (GridTile tile : tiles.values()) {
            if (!(tile == startPosition || tile == endPosition)) {
                tile.resetTile();
            }
```

```
        }
    }
}
```

8.3.4 GridTile.java

```
package com.pathfinding.model;

/**
 * This class is to represent the base tile in a Grid Model. It will have information that might be used for algorithms
 * such as visited or parents.
 */
public class GridTile extends Tile implements Cloneable {
    public static final int BLOCKED = 1;
    public static final int FREE = 0;
    public int collisionFlag = 0;

    //Historical data used for algorithms
    public GridTile parent = null;
    public boolean visited = false;

    public GridTile(int x, int y, int collisionFlag) {
        super(x, y);
        this.collisionFlag = collisionFlag;
    }

    public GridTile(int x, int y) {
        super(x, y);
    }

    public GridTile(Tile tile) {
        super(tile.x, tile.y);
    }

    /**
     * @return Shallow copy of a GridTie
     * @throws CloneNotSupportedException - possible the object might have an issue cloning
     */
    public GridTile clone() throws CloneNotSupportedException {
        return (GridTile) super.clone();
    }

    /**
     * @postcondition - Clears out historical data for a specific tile
     */
    public void resetTile() {
        parent = null;
        visited = false;
    }
}
```

8.3.5 Path.java

```
package com.pathfinding.model;
```

```
2
3  import java.util.ArrayList;
4  import java.util.Iterator;
5
6  public class Path implements Iterable<GridTile> {
7      int index = 0;
8      private ArrayList<GridTile> tiles = new ArrayList<>();
9
10     public Path(ArrayList<GridTile> tiles) {
11         this.tiles.addAll(tiles);
12     }
13
14     public Path() {
15
16     }
17
18     /**
19      * @param tile - to be added to the tiles list
20      */
21     public void addTile(GridTile tile) {
22         tiles.add(tile);
23     }
24
25     /**
26      * Removing the last item in the list
27      */
28     public void remove() {
29         tiles.remove(tiles.size() - 1);
30     }
31
32     /**
33      * Clear out all of the tiles with in the internal list
34      */
35     public void clear() {
36         tiles.clear();
37     }
38
39     /**
40      * @return the size of the path
41      */
42     public int getSize() {
43         return tiles.size();
44     }
45
46     /**
47      * @return the iterator which would be used to loop over the tiles.
48      */
49     @Override
50     public Iterator<GridTile> iterator() {
51         return tiles.iterator();
52     }
53
54 }
```

8.3.6 Tile.java

```java
package com.pathfinding.model;

/**
 * Base of the GridTile to implemnt a basic 2d point on a grid
 */
public class Tile implements Cloneable {
    public int x, y;

    public Tile(int x, int y) {
        this.x = x;
        this.y = y;
    }

    /**
     * @return simple unique identifier for a tile used in hash sets
     */
    public String getID() {
        return x + "," + y;
    }

}
```

8.4 Simulation Package

8.4.1 Simulation.java

```java
package com.pathfinding.simulation;

import com.pathfinding.algorithms.Algorithm;
import com.pathfinding.model.*;

import java.util.ArrayList;
import java.util.Random;

/**
 * This class is responsible for running the simulation against the different Algorithm.
 * Note: When running large graphs the simulation could take a long time. For better
     results it would be possible
 * to create multiple threads since they can be run in parallel but that might also
     tarnish the run times.
 */
public class Simulation {
    ArrayList<Algorithm> algorithms;
    int iterations;
    int widthSize;
    int heightSize;
    AlgorithmStatModel algorithmStatModel;

    public Simulation(int iterations, ArrayList<Algorithm> algorithms, int widthSize,
        int heightSize, AlgorithmStatModel algorithmStatModel) {
        this.algorithms = algorithms;
        this.iterations = iterations;
        this.heightSize = heightSize;
        this.widthSize = widthSize;
```

```
26          this.algorithmStatModel = algorithmStatModel;
27      }
28
29      /**
30       * The run method will execute the simulation. The following steps happen in this
             method
31       * 1. Generate random start and end paris
32       * 2. Create a new graph to be used in the simulation
33       * 3. execute algorithms on the graph
34       * 4. collection the results
35       *
36       * @preconditon - Ensure that the width and height are greater than 0, more than 1
             algorithm in the list, and that the algorithmStatModel != null
37       * @postcondition - all of the results will be stored in algorithmStatModel
38       */
39      public void run() {
40          ArrayList<Tile[]> randomTilePairs = new ArrayList<>();
41          Random rand = new Random();
42          //Create a list of random start and end
43          for (int i = 0; i < iterations; i++) {
44              Tile[] pair = new Tile[2];
45              pair[0] = new Tile(rand.nextInt(widthSize), rand.nextInt(heightSize));
46              pair[1] = new Tile(rand.nextInt(widthSize), rand.nextInt(heightSize));
47              randomTilePairs.add(pair);
48          }
49          GridModel gridModel = new GridModel(this.widthSize, this.heightSize);
50          //Run over 10 different grid models
51          for (int i = 0; i < 5; i++) {
52              gridModel.newRandomGraph();
53              for (Algorithm algorithm : algorithms) {
54                  for (Tile[] pair : randomTilePairs) {
55                      long startTime = System.currentTimeMillis();
56                      Path path = algorithm.computeOptimalPath(pair[0], pair[1],
                            gridModel);
57                      long endTime = System.currentTimeMillis();
58                      int visitedCount = 0;
59                      for (GridTile tile : gridModel.tiles.values()) {
60                          visitedCount += tile.visited ? 1 : 0;
61                      }
62                      int pathLen = 0;
63                      if (path != null) {
64                          pathLen = path.getSize();
65                      }
66                      AlgorithmResult algorithmResult =
67                              new AlgorithmResult(algorithm.getName(),
68                                      (int) (endTime - startTime),
69                                      pathLen,
70                                      pathLen > 0,
71                                      visitedCount);
72                      this.algorithmStatModel.addAlgorithm(algorithmResult);
73                  }
74              }
75          }
76      }
```

```
}
```

8.5 View Package

8.5.1 GridView.java

```
package com.pathfinding.view;

import com.pathfinding.model.GridModel;
import com.pathfinding.model.GridTile;

import javax.swing.*;
import java.awt.*;


/**
 * This is the View where the main visualization of the Grid will be used
 */
public class GridView extends JComponent {

    final Color COLOR_DEFAULT = Color.lightGray;
    final Color COLOR_BLOCKED = Color.black;
    final Color COLOR_START = Color.green;
    final Color COLOR_END = Color.RED;
    final Color COLOR_VISITED = Color.orange;
    final Color COLOR_PATH = Color.magenta;
    GridModel gridModel;

    GridView(GridModel gridModel) {
        this.gridModel = gridModel;

    }

    /**
     * This function is used to generate the grid. It will compute the columns and rows
         based on the size of the main
     * panel. It will also print the coordinates for a 20x20 graph. It will also display
         a graph
     *
     * @param g used to paint to the screen
     */
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        int i;
        int width, height;
        int rows = gridModel.heightSize;
        int cols = gridModel.widthSize;
        // -10 is to ensure we dont have any clipping into the next cell south
        width = getSize().width - 5;
        height = getSize().height - 33; // Save some space for the key at the bottom

        //draw rectangles
        Color prevColor;
        int rowHt = height / (rows);
        int rowWid = width / (cols);
```

```
48          for (int y = 0; y < rows; y++) {
49              for (int x = 0; x < cols; x++) {
50                  prevColor = g.getColor();
51                  GridTile currentTile = gridModel.tiles.get(x + "," + y);
52                  g.setColor(getTileColor(currentTile, gridModel));
53                  g.fillRect(x * rowWid, y * rowHt, rowWid, rowHt);
54                  g.setColor(prevColor);
55              }
56          }
57
58          // Draw outlines for the squares
59          g.setColor(Color.BLACK);
60          for (int y = 0; y < rows; y++) {
61              for (int x = 0; x < cols; x++) {
62                  g.drawRect(x * rowWid, y * rowHt, rowWid, rowHt);
63              }
64          }
65
66
67          // Only print the text for the small graph
68          if (rows < 21) {
69              g.setFont(new Font("Verdana", Font.PLAIN, 8));
70              // draw cords
71              for (i = 0; i < rows; i++) {
72                  for (int j = 0; j < cols; j++) {
73                      g.drawString("(" + i + "," + j + ")", i * rowWid + rowWid / 4 - 1, j
                            * rowHt + rowHt / 2 + 3);
74                  }
75              }
76
77          }
78
79          g.setFont(new Font("Verdana", Font.PLAIN, 10));
80          // Draw the key
81          prevColor = g.getColor();
82
83          int keySize = 49;
84          int x = 1;
85          //Setting rows and colums to set the key in the same location between
86          rows = 20;
87          cols = 18;
88          rowHt = height / (rows);
89          rowWid = width / (cols);
90          g.setColor(COLOR_BLOCKED);
91          g.fillRect(x * rowWid, (rows + 1) * rowHt, keySize, keySize);
92          g.setColor(Color.white);
93          g.drawString("Blocked", x * rowWid, (rows + 1) * rowHt + rowHt * 2);
94
95
96          x = 2;
97          g.setColor(COLOR_DEFAULT);
98          g.fillRect(x * rowWid, (rows + 1) * rowHt, keySize, keySize);
99          g.setColor(Color.BLACK);
100         g.drawString("Open", x * rowWid, (rows + 1) * rowHt + rowHt * 2);
```

```
101
102         x = 3;
103         g.setColor(COLOR_START);
104         g.fillRect(x * rowWid, (rows + 1) * rowHt, keySize, keySize);
105         g.setColor(Color.BLACK);
106         g.drawString("Start", x * rowWid, (rows + 1) * rowHt + rowHt * 2);
107
108         x = 4;
109         g.setColor(COLOR_END);
110         g.fillRect(x * rowWid, (rows + 1) * rowHt, keySize, keySize);
111         g.setColor(Color.BLACK);
112         g.drawString("End", x * rowWid, (rows + 1) * rowHt + rowHt * 2);
113
114         x = 5;
115         g.setColor(COLOR_VISITED);
116         g.fillRect(x * rowWid, (rows + 1) * rowHt, keySize, keySize);
117         g.setColor(Color.BLACK);
118         g.drawString("Visited", x * rowWid, (rows + 1) * rowHt + rowHt * 2);
119
120         x = 6;
121         g.setColor(COLOR_PATH);
122         g.fillRect(x * rowWid, (rows + 1) * rowHt, keySize, keySize);
123         g.setColor(Color.BLACK);
124         g.drawString("Path", x * rowWid, (rows + 1) * rowHt + rowHt * 2);
125
126         g.setColor(prevColor);
127     }
128
129     /**
130      * @param tile      - used to look up the data to know what the color should be
131      * @param gridModel - used to get the start and end positions to ensure they are not
             colored in. Also to used to make
132      *                  sure the tile is not on the path, or if it is set its correct
             color.
133      * @return the color the given tile should be colored to
134      */
135     Color getTileColor(GridTile tile, GridModel gridModel) {
136         Color updatedColor = COLOR_DEFAULT;
137         if (tile.collisionFlag == GridTile.BLOCKED) {
138             updatedColor = COLOR_BLOCKED;
139         } else if (tile.collisionFlag == GridTile.FREE) {
140             if (!tile.visited) {
141                 updatedColor = COLOR_DEFAULT;
142             } else {
143
144                 updatedColor = COLOR_VISITED;
145             }
146         }
147
148         // Ensure that we set the start and end positions correctly
149         if (tile.x == gridModel.startPosition.x && tile.y == gridModel.startPosition.y) {
150             updatedColor = COLOR_START;
151         }
152         if (tile.x == gridModel.endPosition.x && tile.y == gridModel.endPosition.y) {
```

```
153             updatedColor = COLOR_END;
154         }
155 
156         if (gridModel.path != null) {
157             for (GridTile pathTile : gridModel.path) {
158                 //Dont color the start and end tiles
159                 if (!(pathTile.x == gridModel.startPosition.x && pathTile.y ==
                        gridModel.startPosition.y)
160                         && !(pathTile.x == gridModel.endPosition.x && pathTile.y ==
                            gridModel.endPosition.y)
161                         && pathTile.x == tile.x && pathTile.y == tile.y) {
162                     updatedColor = COLOR_PATH;
163                 }
164             }
165         }
166         return updatedColor;
167     }
168 
169 
170     /**
171      * @return base size requested for the GridView
172      */
173     @Override
174     public Dimension getPreferredSize() {
175         return new Dimension(200, 200);
176     }
177 }
```

8.5.2 GUIView.java

```
1  package com.pathfinding.view;
2  
3  import com.pathfinding.algorithms.Algorithm;
4  import com.pathfinding.model.AlgorithmStatModel;
5  import com.pathfinding.model.GridModel;
6  
7  import javax.swing.*;
8  import java.awt.*;
9  import java.awt.event.ActionListener;
10 import java.util.ArrayList;
11 
12 /**
13  * This is the main View that all the base components are created and positioned.
14  */
15 public class GUIView {
16     private static GUIView guiView;
17     public JComponent grid;
18     public JButton buttonGenerateRandomStartEndPositions;
19     public JButton buttonGenerateRandomGrid;
20     public JComboBox<String> algorithmDropdown, dropdownGridSize;
21     public JTable tableComparisonTable;
22     public JTextField textFieldStartPosition, textFieldEndPosition;
23     public GridModel gridModel;
24     public ArrayList<Algorithm> algorithms;
25     public JRadioButton radioButton20x20, radioButton100x100;
```

```
26      AlgorithmStatModel algorithmStatModel;
27      private JFrame mainContainer;
28      private JButton buttonGeneratePath;
29      private JButton buttonRunComparison;
30      private ActionListener actionListener;
31
32
33      /**
34       * This is the main View that has buttons, tables, labels, JComponents, Dropdown
             options
35       *
36       * @param gridModel          reference to the GridModel used to create the GridView
37       * @param algorithms         - list of algorithms used to populate JComboboxes
38       * @param algorithmStatModel - reference to the model for the Algorithm comparison
             JTable
39       */
40      private GUIView(GridModel gridModel, ArrayList<Algorithm> algorithms,
            AlgorithmStatModel algorithmStatModel) {
41          this.gridModel = gridModel;
42          this.algorithms = algorithms;
43          this.algorithmStatModel = algorithmStatModel;
44          int yPos = 0;
45          mainContainer = new JFrame();
46          GridBagConstraints c = new GridBagConstraints();
47          mainContainer.setLayout(new GridBagLayout());
48          mainContainer.setPreferredSize(new Dimension(900, 1000));
49
50          buttonGenerateRandomGrid = new JButton("New Graph");
51          c.fill = GridBagConstraints.NONE;
52          c.weightx = 1;
53          c.weighty = 0.0;
54          c.gridwidth = 2;
55          c.gridx = 0;
56          c.gridy = yPos++;
57          int pad = 10;
58          c.insets = new Insets(pad, pad, pad, pad);
59          mainContainer.add(buttonGenerateRandomGrid, c);
60
61          radioButton20x20 = new JRadioButton("20x20");
62          radioButton20x20.setActionCommand("20");
63          radioButton100x100 = new JRadioButton("100x100");
64          radioButton20x20.setActionCommand("100");
65
66          String[] sizeList = new String[2];
67          sizeList[0] = "20x20";
68          sizeList[1] = "100x100";
69          dropdownGridSize = new JComboBox<>(sizeList);
70          c.gridy = yPos++;
71          c.gridx = 0;
72          c.fill = GridBagConstraints.NONE;
73          mainContainer.add(dropdownGridSize, c);
74
75
76          grid = new GridView(this.gridModel);
```

```
77          c.fill = GridBagConstraints.BOTH;
78
79          c.weighty = .3;
80          c.weightx = 1;
81          c.gridwidth = 2;
82          c.gridx = 0;
83          c.gridy = yPos++;
84          grid.setVisible(true);
85          mainContainer.add(grid, c);
86          grid.setVisible(true);
87
88          String[] list = new String[algorithms.size()];
89          for (int j = 0; j < algorithms.size(); j++) {
90              list[j] = algorithms.get(j).getName();
91          }
92          algorithmDropdown = new JComboBox<>(list);
93          c.fill = GridBagConstraints.NONE;
94          c.weightx = 0.0;
95          c.weighty = 0.0;
96          c.gridwidth = 2;
97          c.gridx = 0;
98          c.gridy = yPos++;
99          mainContainer.add(algorithmDropdown, c);
100
101         buttonGenerateRandomStartEndPositions = new JButton("Generate Random Path");
102         c.fill = GridBagConstraints.NONE;
103         c.weightx = 0.0;
104         c.weighty = 0.0;
105         c.gridwidth = 2;
106         c.gridx = 0;
107         c.gridy = yPos++;
108         mainContainer.add(buttonGenerateRandomStartEndPositions, c);
109
110         buttonGeneratePath = new JButton("Generate Path");
111         c.fill = GridBagConstraints.NONE;
112         c.weightx = 0.0;
113         c.weighty = 0.0;
114         c.gridwidth = 2;
115         c.gridx = 0;
116         c.gridy = yPos++;
117         mainContainer.add(buttonGeneratePath, c);
118
119         textFieldStartPosition = new JTextField("0,0");
120         c.fill = GridBagConstraints.HORIZONTAL;
121         c.weightx = 0.5;
122         c.gridwidth = 1;
123         c.gridx = 0;
124         c.gridy = yPos;
125         mainContainer.add(textFieldStartPosition, c);
126
127         textFieldEndPosition = new JTextField("0,0");
128         c.fill = GridBagConstraints.HORIZONTAL;
129         c.weightx = 0.5;
130         c.gridwidth = 1;
```

```
131         c.gridx = 1;
132         c.gridy = yPos++;
133         mainContainer.add(textFieldEndPosition, c);
134 
135         c.fill = GridBagConstraints.HORIZONTAL;
136         c.weightx = 0.0;
137         c.weighty = 0.0;
138         c.gridwidth = 2;
139         c.gridx = 0;
140         c.gridy = yPos++;
141 
142         buttonRunComparison = new JButton("Compute Comparison");
143         c.gridy = yPos++;
144         mainContainer.add(buttonRunComparison, c);
145 
146         c.gridy = yPos++;
147         JLabel comparisonResultsLabel = new JLabel("Comparison Results");
148         comparisonResultsLabel.setFont(new Font("Verdana", Font.PLAIN, 20));
149         mainContainer.add(comparisonResultsLabel, c);
150 
151         c.gridx = 1;
152         JLabel comparisonResultsLabelDetail = new JLabel("Simulating 5 random
                graphs(100x100) for 10 random start end pairs");
153         comparisonResultsLabelDetail.setFont(new Font("Verdana", Font.PLAIN, 12));
154         mainContainer.add(comparisonResultsLabelDetail, c);
155         c.gridx = 0;
156 
157 
158         JPanel tablePanel = new JPanel();
159         tablePanel.setLayout(new BorderLayout());
160 
161         tableComparisonTable = new JTable(algorithmStatModel);
162         tablePanel.add(tableComparisonTable.getTableHeader(), BorderLayout.PAGE_START);
163         tablePanel.add(tableComparisonTable, BorderLayout.CENTER);
164         c.gridy = yPos;
165         mainContainer.add(tablePanel, c);
166 
167         mainContainer.setTitle("Path Finding Algorithm tester");
168         mainContainer.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
169         mainContainer.pack();
170         mainContainer.setVisible(true);
171         grid.setVisible(true);
172 
173     }
174 
175     /**
176      * Singelton to get the GUI View
177      *
178      * @param gridModel          reference to the main gridmodel
179      * @param algorithms         referene to the lsit og algorithms
180      * @param algorithmStatModel reference to the model for the Stastics table
181      * @return instance of the GUIView
182      */
183     public static GUIView getInstance(GridModel gridModel, ArrayList<Algorithm>
```

```
        algorithms, AlgorithmStatModel algorithmStatModel) {
        if (guiView == null) {
            guiView = new GUIView(gridModel, algorithms, algorithmStatModel);
        }
        return guiView;
    }


    /**
     * @param actionListener used to add action listner to a button
     */
    public void addALForNewGrid(ActionListener actionListener) {
        buttonGenerateRandomGrid.addActionListener(actionListener);
    }

    /**
     * @param actionListener used to add action listner to a button
     */
    public void addALForGenerateRandomStartEndPositions(ActionListener actionListener) {
        this.actionListener = actionListener;
        buttonGenerateRandomStartEndPositions.addActionListener(actionListener);
    }

    /**
     * @param actionListener used to add action listner to a button
     */
    public void addALForButtonGeneratePath(ActionListener actionListener) {
        buttonGeneratePath.addActionListener(actionListener);
    }

    /**
     * @param actionListener used to add action listner to a button
     */
    public void addALForButtonRunComparison(ActionListener actionListener) {
        buttonRunComparison.addActionListener(actionListener);
    }
}
```

8.6 JUnit Tests

8.6.1 AlgorithmStatModelTest.java

```
package com.pathfinding.model;

import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class AlgorithmStatModelTest {

    AlgorithmStatModel algorithmStatModel;

    /**
     * This is the set up for the algorithmStatModel which will be used in all of the
```

```
        test below. We are formatting
14      * on 3 Algorithms
15      */
16     @Before
17     public void setUp() {
18         String[] headers = {"col1", "algo1", "algo2", "algo3"};
19         algorithmStatModel = new AlgorithmStatModel(headers);
20     }
21
22     /**
23      * Simple check to see that result got added to the list
24      */
25     @Test
26     public void addAlgorithm() {
27         AlgorithmResult algorithmResult = new AlgorithmResult("algo1", 111, 50, true,
               222);
28         algorithmStatModel.addAlgorithm(algorithmResult);
29         assertEquals(algorithmStatModel.latestAlgorithmStats.size(), 1);
30
31     }
32
33     /**
34      * The idea behind this test will populated 2 runs for each algo. Only testing the
            first algo for
35      * all of the rows after the computeStatistic are called.
36      */
37     @Test
38     public void computeStatistic() {
39         AlgorithmResult algorithmResult = new AlgorithmResult("algo1", 111, 50, true,
               222);
40         algorithmStatModel.addAlgorithm(algorithmResult);
41         AlgorithmResult algorithmResult2 = new AlgorithmResult("algo1", 222, 100, true,
               500);
42         algorithmStatModel.addAlgorithm(algorithmResult2);
43         AlgorithmResult algorithmResult3 = new AlgorithmResult("algo2", 123, 345, true,
               567);
44         algorithmStatModel.addAlgorithm(algorithmResult3);
45         AlgorithmResult algorithmResult4 = new AlgorithmResult("algo2", 123, 345, true,
               567);
46         algorithmStatModel.addAlgorithm(algorithmResult4);
47         AlgorithmResult algorithmResult5 = new AlgorithmResult("algo3", 321, 543, true,
               765);
48         algorithmStatModel.addAlgorithm(algorithmResult5);
49         AlgorithmResult algorithmResult6 = new AlgorithmResult("algo3", 321, 543, true,
               765);
50         algorithmStatModel.addAlgorithm(algorithmResult6);
51
52         algorithmStatModel.computeStatistic();
53         assertEquals(6, algorithmStatModel.latestAlgorithmStats.size());
54
55         // Total run time checks
56         assertEquals("333", algorithmStatModel.getValueAt(1, 1));
57         assertEquals("111", algorithmStatModel.getValueAt(2, 1));
58         assertEquals("222", algorithmStatModel.getValueAt(3, 1));
```

```
59          assertEquals("166.5", algorithmStatModel.getValueAt(4, 1));
60          assertEquals("55.5", algorithmStatModel.getValueAt(5, 1));
61
62          //Path length cheks
63          assertEquals("150", algorithmStatModel.getValueAt(6, 1));
64          assertEquals("50", algorithmStatModel.getValueAt(7, 1));
65          assertEquals("100", algorithmStatModel.getValueAt(8, 1));
66          assertEquals("75", algorithmStatModel.getValueAt(9, 1));
67          assertEquals("25", algorithmStatModel.getValueAt(10, 1));
68
69          // Visited tiles check
70          assertEquals("722", algorithmStatModel.getValueAt(11, 1));
71          assertEquals("222", algorithmStatModel.getValueAt(12, 1));
72          assertEquals("500", algorithmStatModel.getValueAt(13, 1));
73          assertEquals("361", algorithmStatModel.getValueAt(14, 1));
74          assertEquals("139", algorithmStatModel.getValueAt(15, 1));
75
76          //Number of paths found
77          assertEquals(4, algorithmStatModel.getValueAt(16, 1));
78      }
79
80      /**
81       * check to make sure the column names can be retrieved correctly
82       */
83      @Test
84      public void getColumnName() {
85          assertEquals("col1", algorithmStatModel.getColumnName(0));
86          assertEquals("algo1", algorithmStatModel.getColumnName(1));
87          assertEquals("algo2", algorithmStatModel.getColumnName(2));
88          assertEquals("algo3", algorithmStatModel.getColumnName(3));
89      }
90
91      /**
92       * Check to make sure the table size can be accessed
93       */
94      @Test
95      public void getRowCount() {
96          //The data object is predefined and doesnt change is size. Only the content
                changes
97          assertEquals(17, algorithmStatModel.getRowCount());
98      }
99
100     /**
101      * Check to make sure the columns are correct
102      */
103     @Test
104     public void getColumnCount() {
105         assertEquals(4, algorithmStatModel.getColumnCount());
106     }
107
108     /**
109      * Simple check of the first row since its predefined
110      */
111     @Test
```

```
112     public void getValueAt() {
113         assertEquals("Total Run Time", algorithmStatModel.getValueAt(1, 0));
114         assertEquals("Min Run Time", algorithmStatModel.getValueAt(2, 0));
115         assertEquals("Max Run Time", algorithmStatModel.getValueAt(3, 0));
116     }
117 }
```

8.6.2 GridTileTest.java

```
1  package com.pathfinding.model;
2
3  import org.junit.Test;
4
5  import static org.junit.Assert.assertFalse;
6  import static org.junit.Assert.assertNull;
7
8  public class GridTileTest {
9
10     /**
11      * Testing to make sure resetTile will clear out the history info such as visited
            and parent
12      */
13     @Test
14     public void resetTile() {
15         GridTile gt = new GridTile(1, 3);
16         gt.visited = true;
17         gt.parent = new GridTile(9, 9);
18         gt.resetTile();
19         assertFalse(gt.visited);
20         assertNull(gt.parent);
21     }
22 }
```

8.6.3 PathTest.java

```
1  package com.pathfinding.model;
2
3  import org.junit.Before;
4  import org.junit.Test;
5
6  import java.util.ArrayList;
7  import java.util.Iterator;
8
9  import static org.junit.Assert.assertEquals;
10
11 public class PathTest {
12
13     Path path;
14     int numberOfInitialTiles = 10;
15
16     /**
17      * Initial set up will create some tiles that will have x and y as the same loop
            index.
18      */
19     @Before
```

```
20      public void setUp() {
21          ArrayList<GridTile> tiles = new ArrayList<>();
22          for (int i = 0; i < numberOfInitialTiles; i++) {
23              tiles.add(new GridTile(i, i));
24          }
25
26          path = new Path(tiles);
27      }
28
29      /**
30       * Testing removing all the elements in the list 1 by 1 and using the getSize to
             ensure we drop by 1 each time
31       */
32      @Test
33      public void remove() {
34          int sizeTest = path.getSize();
35          Iterator<GridTile> iterable = path.iterator();
36
37          while (iterable.hasNext()) {
38              path.remove();
39              assertEquals(sizeTest - 1, path.getSize());
40              sizeTest--;
41          }
42      }
43
44      /**
45       * Testing adding 1 tile and checking to see if the size increased by 1
46       */
47      @Test
48      public void addTile() {
49          int sizeTest = path.getSize();
50          path.addTile(new GridTile(99, 99));
51          assertEquals(sizeTest + 1, path.getSize());
52      }
53
54      /**
55       * Testing to see if the path length using getSize is the same as the initialized on
             our setUp method.
56       */
57      @Test
58      public void getSize() {
59          assertEquals(numberOfInitialTiles, path.getSize());
60      }
61
62      /**
63       * Testing out the iterator to ensure that we can loop over the internal list. Using
             the x and y coordinates
64       * to ensure they are looping in the correct order
65       */
66      @Test
67      public void iterator() {
68          Iterator<GridTile> iterable = path.iterator();
69          int tileIndex = 0;
70          while (iterable.hasNext()) {
```

```
            GridTile tile = iterable.next();
            assertEquals(tile.x, tileIndex);
            assertEquals(tile.y, tileIndex);
            tileIndex++;
        }
    }

    /**
     * Testing out the clear method to ensure that the internal list gets cleared out
         and using getSize to test
     */
    @Test
    public void clear() {
        assertEquals(numberOfInitialTiles, path.getSize());
        path.clear();
        assertEquals(0, path.getSize());
    }
}
```

8.6.4 TileTest.java

```
package com.pathfinding.model;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class TileTest {

    /**
     * Testing the format of the ID which will be "x,y"
     */
    @Test
    public void getID() {

        Tile t = new Tile(1, 5);

        assertEquals("1,5", t.getID());

    }
}
```

9 Glossary

- **Path Algorithm** - A set of instructions to iterate over the points to find a way from point A to point B
- **Start, departure point** - the first node in the path
- **End, destination point** - the last node in the path
- **Node, Tile, Point** - Each term here represents an coordinate in a 2 dimensional graph. They will be used in a group to represent a path
- **Path** - a group of nodes,tiles,points that make up an array of elements used describe how to get from point A to point B
- **Results** - a comparison between multiple algorithms
- **Simulation** - the even when all the algorithms will be run against a graph to compare the results