Testing Report

Matthew James Nestor
a1132338

Group 2

November 4, 2013

1 Introduction

Herein details the testing cases for the Graph class, and where necessary, the classes which Graph uses. The Graph class is responsible for creating a graph data-structure on the basis of the Map data-structure. It is populated with vertices for all intersections, road-ends, road-closures, obstacles, and disaster-zone/road collisions that are reachable from the robot's initial position, and edges for all roads which link them. It also contains methods that search the graph data-structure, in particular, **AStarSearch(Vertex, Vertex)**. Furthermore, it contains methods which link transitions between vertices to actions the robot can take (e.g. by supplying a vertex with an action, such as travel **NORTH**, the vertex will return the next vertex that would be reached if that action were taken, if that vertex exists). The Graph class can be found in the below location. All other classes that Graph uses are in the same package, with the exception of the Map classes, which are in the package **HostSide.mapDataStructure**.

`https://version-control.adelaide.edu.au/svn/sep2012-2/trunk/code/SelectiveRepeatVersion/SEPProject/HostSide/src/artificialIntelligenceUnit/Graph.java`

2 Testing Description and Rationale

All tests were performed using JUnit test cases and designed with the aid of the elcEMMA code coverage tool. According to elcEMMA, the JUnit tests cover %84.3 of the Graph class code, %81.1 of all code in the **artificialIntelligenceUnit** package, and %40.0 of all code in the HostSide project. Below are the test cases, with explanations.

1. **testCollisionCalculation()**: This test case tests the method **Graph.testCollision(Point, Point, Point, double)**. In order to populate the graph data-structure, it is necessary to determine which roads, if any, intersect with disaster zones, and where those intersections occur. This calculation is performed by **Graph.testCollision(Point, Point, Point, double)**.
2. **testIsInBetween()**: This test case tests the **Graph.isInBetween(Point, Point, Point)** method, which tests whether a point lies on a line between two points.
3. **testAStarSearch()**: This test case tests the **Graph.AStarSearch(Vertex, Vertex)** method, which searches the graph data-structure and returns the solution as an **ArrayList<Vertex>**. The solution is the shortest path from the LHS argument to the RHS argument, or **null** if no path exists.
4. **testUpdateGraph()**: This tests the **Graph.updateGraph()**, **Action.getAction()**, and **Action.getDistance()** methods. **Graph.updateGraph()** updates the classes local variables, then calls **Graph.AStarSearch(Vertex, Vertex)**. This returns a **Deque<Action>** object, each **Action** in which contains data about the next action the robot must take in order to reach the next goal.
5. **testMisc()**: Tests for various GET, SET, and REMOVE methods.

A Testing Code: GraphTest.java

The testing code can be found in the following location.

`https://version-control.adelaide.edu.au/svn/sep2012-2/trunk/code/SelectiveRepeatVersion/SEPProject/HostSide/src/artificialIntelligenceUnit/GraphTest.java`

```
1  package artificialIntelligenceUnit;
2
3  import static org.junit.Assert.*;
4
5  import java.awt.Point;
6  import java.util.ArrayList;
7  import java.util.Deque;
8  import java.util.LinkedList;
9
10 import junit.framework.Assert;
11
12 import mapDataStructure.Map;
13 import mapDataStructure.RobotMap;
14
15 import org.junit.Before;
16 import org.junit.Test;
17
18 import controllerFSM.ControllerFSM;
19
20 /**
21  * @author Matthew Nestor
22  * @filename GraphTest.java
23  * @package artificialIntelligenceUnit
24  * @project HostSide
25  * @date 03/11/2013
26  */
27
28 public class GraphTest {
29   private Graph graph;
30   private Map map;
31
32   @Before
33   public void before(){
34     map = new RobotMap();
35     map.loadMap("map6.xml");
36     graph = new Graph(map);
37     graph.initialiseGraph();
38   }
39
40   @Test
41   public void testCollisionCalculation() {
42     graph.print();
43     CollisionResult result = graph.testCollision(new Point(0, 0),
44         new Point(2, 0),
45         new Point(-1, 0),
46         2.0);
47     result.print();
48     Assert.assertEquals(true, result.collision());
49     Assert.assertEquals(false, result.twoCollisions());
50     Assert.assertEquals(new Point(1, 0), result.firstCollision());
51
52
53     result = graph.testCollision(new Point(0, 0),
54         new Point(2, 0),
55         new Point(3, 0),
56         2.0);
57     result.print();
58     Assert.assertEquals(true, result.collision());
59     Assert.assertEquals(false, result.twoCollisions());
60     Assert.assertEquals(new Point(1, 0), result.firstCollision());
61
62     result = graph.testCollision(new Point(0, 0),
63         new Point(10, 0),
64         new Point(5, 1),
65         2.0);
66     result.print();
67     Assert.assertEquals(true, result.collision());
68     Assert.assertEquals(true, result.twoCollisions());
```

```
69      Assert.assertEquals(new Point(3, 0), result.firstCollision());
70      Assert.assertEquals(new Point(6, 0), result.secondCollision());
71
72      result = graph.testCollision(new Point(0, 0),
73          new Point(10, 0),
74          new Point(5, 1),
75          1.0);
76      result.print();
77      Assert.assertEquals(true, result.collision());
78      Assert.assertEquals(false, result.twoCollisions());
79      Assert.assertEquals(new Point(5, 0), result.firstCollision());
80
81      result = graph.testCollision(new Point(10, 0),
82          new Point(10, 10),
83          new Point(10, 20),
84          5.0);
85      result.print();
86      Assert.assertEquals(false, result.collision());
87      Assert.assertEquals(false, result.twoCollisions());
88
89      result = graph.testCollision(new Point(120, 20),
90          new Point(120, 40),
91          new Point(120, 80),
92          27.0);
93      result.print();
94      Assert.assertEquals(false, result.collision());
95      Assert.assertEquals(false, result.twoCollisions());
96
97      result = graph.testCollision(new Point(120, 20),
98          new Point(120, 0),
99          new Point(120, 80),
100         27.0);
101     result.print();
102     Assert.assertEquals(false, result.collision());
103     Assert.assertEquals(false, result.twoCollisions());
104
105     result = graph.testCollision(new Point(120, 40),
106         new Point(120, 80),
107         new Point(120, 80),
108         27.0);
109     result.print();
110     Assert.assertEquals(true, result.collision());
111     Assert.assertEquals(false, result.twoCollisions());
112     Assert.assertEquals(new Point(120, 53), result.firstCollision());
113
114     result = graph.testCollision(new Point(0, 0),
115         new Point(0, 10),
116         new Point(5, 5),
117         5);
118     result.print();
119     Assert.assertEquals(true, result.collision());
120     Assert.assertEquals(false, result.twoCollisions());
121     Assert.assertEquals(new Point(0, 5), result.firstCollision());
122
123     result = graph.testCollision(new Point(0, 0),
124         new Point(0, 10),
125         new Point(5, 5),
126         100);
127     result.print();
128     Assert.assertEquals(false, result.collision());
129     Assert.assertEquals(false, result.twoCollisions());
130
131     result = graph.testCollision(new Point(5, 10),
132         new Point(5, 1),
133         new Point(5, 0),
134         5);
135     result.print();
136     Assert.assertEquals(true, result.collision());
137     Assert.assertEquals(false, result.twoCollisions());
138     Assert.assertEquals(new Point(5, 5), result.firstCollision());
139
140     result = graph.testCollision(new Point(5, 1),
141         new Point(5, 10),
142         new Point(5, 0),
143         5);
144     result.print();
```

```
145     Assert.assertEquals(true, result.collision());
146     Assert.assertEquals(false, result.twoCollisions());
147     Assert.assertEquals(new Point(5, 5), result.firstCollision());
148
149     result = graph.testCollision(new Point(5, 1),
150         new Point(5, -10),
151         new Point(5, 0),
152         5);
153     result.print();
154     Assert.assertEquals(true, result.collision());
155     Assert.assertEquals(false, result.twoCollisions());
156     Assert.assertEquals(new Point(5, -5), result.firstCollision());
157
158     result = graph.testCollision(new Point(5, 10),
159         new Point(5, -10),
160         new Point(5, 0),
161         5);
162     result.print();
163     Assert.assertEquals(true, result.collision());
164     Assert.assertEquals(true, result.twoCollisions());
165     Assert.assertEquals(new Point(5, 5), result.firstCollision());
166     Assert.assertEquals(new Point(5, -5), result.secondCollision());
167   }
168
169   @Test
170   public void testIsInBetween(){
171     Assert.assertEquals(true, graph.isInBetween(new Point(120, 20), new Point(120,
          40), new Point(120, 40)));
172     Assert.assertEquals(true, graph.isInBetween(new Point(50, 20), new Point(120,
          20), new Point(100, 20)));
173     Assert.assertEquals(false, graph.isInBetween(new Point(120, 20), new Point(120,
           40), new Point(120, 41)));
174     Assert.assertEquals(false, graph.isInBetween(new Point(120, 20), new Point(120,
           40), new Point(100, 38)));
175     Assert.assertEquals(false, graph.isInBetween(new Point(120, 20), new Point(150,
           20), new Point(151, 20)));
176     Assert.assertEquals(false, graph.isInBetween(new Point(120, 20), new Point(150,
           20), new Point(100, 21)));
177   }
178
179   @Test
180   public void testAStarSearch(){
181     Vertex start = graph.getInitialVertex();
182     Vertex goal = graph.goals.peek();
183     ArrayList<Vertex> solution = graph.AStarSearch(start, goal);
184     Assert.assertEquals(solution, graph.getSolution());
185     Assert.assertEquals(new Point(29, 28), solution.remove(0).getCoordinate());
186     Assert.assertEquals(new Point(29, 21), solution.remove(0).getCoordinate());
187     Assert.assertEquals(new Point(29, 1), solution.remove(0).getCoordinate());
188   }
189
190   @Test
191   public void testUpdateGraph(){
192     Deque<Action> actionList = graph.updateGraph(); //calls AStarSearch and returns
           AStarSearch array
193     Action nextAction = actionList.remove();
194     Assert.assertEquals("NORTH", nextAction.getAction());
195     Assert.assertEquals(20, nextAction.getDistance());
196     nextAction = actionList.remove();
197     Assert.assertEquals("NORTH", nextAction.getAction());
198     Assert.assertEquals(7, nextAction.getDistance());
199     graph.print();
200   }
201
202   @Test
203   public void testMisc() throws InterruptedException{
204     Vertex start = graph.getInitialVertex();
205     LinkedList<Edge> outEdges = graph.outEdgesOf(start);
206     for(Edge e: outEdges){
207       Vertex v = e.toVertex();
208       Assert.assertEquals(e, graph.getEdge(start, v));
209       Assert.assertEquals(true, graph.containsEdge(e));
210     }
211     Vertex fakeVertex = new Vertex(21, new Point(1, 1));
212     Assert.assertEquals(false, graph.containsEdge(start, fakeVertex));
213     Assert.assertEquals(null, graph.getEdge(start, fakeVertex));
```

```
214     Vertex nextVertex = graph.transition(start, "NORTH");
215     graph.removeEdge(start, nextVertex);
216     graph.removeEdge(nextVertex, start);
217     Assert.assertEquals(false, graph.containsEdge(start, nextVertex));
218     Vertex goal = graph.getVertex(new Point(55, 21));
219     Assert.assertEquals(true, goal.isGoal());
220     goal = graph.getVertex(new Point(-1, -2));
221     Assert.assertEquals(null, goal);
222     Vertex v2 = graph.transition(nextVertex, "SOUTH");
223     Assert.assertEquals(null, v2);
224     int tempInt = graph.stepCost(nextVertex, "SOUTH");
225     Assert.assertEquals(-1, tempInt);
226   }
```