A Star

Casey Weed

February 7, 2016

Contents

1 AStar

AStar is the primary pane and brings together the functions of Robot, SNode and utilizes the Obstacles for sample polygonal obstacles.

Robot size, shape, and step size can be adjusted. However, the obstacle location, sizes, and orientation cannot be adjusted manually. Instead their properties are randomly assigned. In each iteration of the obstacles spawning process, the obstacle checks whether or not it collides with the start, goal, or robot shapes, if they do the current obstacle is scrapped and another attempt is made.

```
1  import javafx.animation.PathTransition;
2  import javafx.animation.Timeline;
3  import javafx.application.Application;
4  import javafx.event.EventHandler;
5  import javafx.scene.Scene;
6  import javafx.scene.control.Button;
7  import javafx.scene.control.Label;
8  import javafx.scene.control.ToolBar;
9  import javafx.scene.input.KeyEvent;
10 import javafx.scene.layout.BorderPane;
11 import javafx.scene.layout.Pane;
12 import javafx.scene.layout.StackPane;
13 import javafx.scene.paint.Color;
14 import javafx.scene.shape.*;
15 import javafx.stage.Stage;
16 import javafx.util.Duration;
17
18 import java.util.ArrayList;
19 import java.util.Comparator;
20 import java.util.PriorityQueue;
21 import java.util.Random;
22
23 public class AStar extends Application {
24     public static void main(String[] args) {
25         launch(args);
26     }
27
28     @Override
29     public void start(Stage stage) {
30         int initialW = 720;
31         int initialH = 480;
32
33         // controls
34         ToolBar tb = new ToolBar();
35         Button newSceneBtn = new Button("New Scenario");
36         Button tglShape = new Button("Toggle Shape");
37         Label robotSizeLbl = new Label("Robot Size:");
38         Button incRobotSize = new Button("Inc");
39         Button decRobotSize = new Button("Dec");
40         Label stepSizeLbl = new Label("Step Size:");
41         Button incStepSize = new Button("Inc");
42         Button decStepSize = new Button("Dec");
43
44         // add
45         tb.getItems().addAll(
46                 newSceneBtn, tglShape, robotSizeLbl, incRobotSize, decRobotSize,
```

```
47                  stepSizeLbl, incStepSize, decStepSize
48          );
49
50          // a star
51          StackPane sp = new StackPane();
52          sp.setPrefWidth(initialW);
53          sp.setPrefHeight(initialH);
54          sp.minHeight(initialH);
55          sp.minWidth(initialW);
56          AStarSimple as = new AStarSimple();
57          sp.getChildren().add(as);
58          BorderPane tbPane = new BorderPane();
59          tbPane.setTop(tb);
60          sp.getChildren().add(tbPane);
61
62          // buttons & actions
63          newSceneBtn.setOnAction(e -> as.newScenario());
64          incRobotSize.setOnAction(e -> {
65              as.incRobotSize(0.25);
66              as.cleanup();
67          });
68          decRobotSize.setOnAction(e -> {
69              as.decRobotSize(0.25);
70              as.cleanup();
71          });
72          tglShape.setOnMousePressed(e -> {
73              as.toggleShape();
74              as.cleanup();
75          });
76          incStepSize.setOnAction(e -> as.incStepSize(1));
77          decStepSize.setOnAction(e -> as.decStepSize(1));
78
79          // scene & stage
80          final Scene scene = new Scene(sp, initialW, initialH);
81          stage.setTitle("A* Pathfinding");
82          stage.setScene(scene);
83          stage.setResizable(true);
84          stage.show();
85
86          // listeners
87          scene.widthProperty().addListener(l -> {
88              as.cleanup();
89          });
90          scene.heightProperty().addListener(l -> {
91              as.cleanup();
92          });
93
94          // sample key usage, remove when done testing
95          scene.setOnKeyPressed(new EventHandler<KeyEvent>() {
96              @Override
97              public void handle(KeyEvent event) {
98                  switch (event.getCode()) {
99                      case P: as.solve(); break;
100                 }
101             }
102         });
103     }
```

```
104
105     static class AStarSimple extends Pane {
106         // objs
107         public SNode start;
108         public SNode goal;
109         public Robot robot;
110
111         // misc
112         public double robotSize = 1;
113         public int shapeChoice = 0;
114         public int stepSize = 5;
115
116         // obstacles
117         public ArrayList<Polygon> obstacles = new ArrayList<>();
118
119         // Directions for delta x,y when checking neighbors
120         public enum Direction {
121
122             TOPLEFT(-1, -1),
123             TOPRIGHT(1, -1),
124             BOTTOMLEFT(-1, 1),
125             BOTTOMRIGHT(1, 1),
126             LEFT(-1, 0),
127             RIGHT(1, 0),
128             UP(0, -1),
129             DOWN(0, 1);
130
131             public final int dx;
132             public final int dy;
133
134             Direction(int dx, int dy) {
135                 this.dx = dx;
136                 this.dy = dy;
137             }
138         }
139
140         // TODO: When changing size check if you collide, if you do DO NOT increment
141         public void incRobotSize(double step) {
142             double before = robotSize;
143             robotSize += step > 0 ? step : 1;
144             robot.setScale(robotSize);
145
146             // depracated; screen is cleared for any change whatsoever, better to
                    be safe than sorry though
147             for (Polygon o : obstacles) {
148                 if (intersecting(robot.getShape(), o)) {
149                     robotSize = before;
150                     robot.setScale(before);
151                 }
152             }
153         }
154
155         public void decRobotSize(double step) {
156             robotSize -= step > 0 && (robotSize - step) >= 1 ? step : 0;
157             robot.setScale(robotSize);
158         }
159
```

```
160         public void incStepSize(int step) {
161             stepSize += (stepSize + step) <= 10 ? step : 0;
162             System.out.println(stepSize);
163         }
164 
165         public void decStepSize(int step) {
166             stepSize -= step > 0 && (stepSize - step) >= 2 ? step : 0;
167             System.out.println(stepSize);
168         }
169 
170         // TODO: transition shape definitions to Robot class
171         // Equilateral triangle default
172         public Polygon EquilateralTriangle() {
173             return EquilateralTriangle(20);
174         }
175 
176         // Equilateral triangle
177         public Polygon EquilateralTriangle(double scale) {
178             if (scale <= 10) scale = 10;
179             double w = scale;
180             double h = scale;
181             double[] points = {
182                     -0.866 * scale, scale,
183                     0, -0.5 * scale,
184                     0.866 * scale, scale
185             };
186             Polygon triangle = new Polygon(points);
187             triangle.setFill(Color.VIOLET);
188             triangle.toBack();
189             return triangle;
190         }
191 
192         // Basic circle default
193         public Shape BasicCircle() {
194             return BasicCircle(10);
195         }
196 
197         // Basic circle
198         public Shape BasicCircle(double radius) {
199             if (radius <= 10) radius = 10;
200             Circle circle = new Circle(radius);
201             circle.setFill(Color.VIOLET);
202             circle.toBack();
203             return circle;
204         }
205 
206         public class SNodeComparator implements Comparator<SNode> {
207             @Override
208             public int compare(SNode a, SNode b) {
209                 return Double.compare(a.getF(), b.getF());
210             }
211         }
212 
213         public void toggleShape() {
214             if (shapeChoice == 0) {
215                 changeShape(1);
216             } else {
```

```
217                 changeShape(0);
218             }
219         }
220 
221         public void changeShape() {
222             changeShape(shapeChoice);
223         }
224 
225         public void changeShape(int choice) {
226             Shape before = robot.getShape();
227             switch (choice) {
228                 case 0: robot.setShape(BasicCircle()); break;
229                 case 1: robot.setShape(EquilateralTriangle()); break;
230             }
231             getChildren().remove(before);
232             shapeChoice = choice;
233             robot.setXY(start.getX(), start.getY());
234             robot.setScale(robotSize);
235             getChildren().add(robot.getShape());
236         }
237 
238         // constructors and steps for creating pane
239         public AStarSimple() {}
240 
241         public boolean intersecting(Shape a, Shape b) {
242             Shape i = Shape.intersect(a, b);
243             return i.getBoundsInLocal().getHeight() != -1;
244         }
245 
246         public void spawnRobot() {
247             if (robot == null) {
248                 robot = new Robot(start.getX(), start.getY(), BasicCircle());
249             }
250             changeShape();
251             robot.setXY(start.getX(), start.getY());
252         }
253 
254         public void spawnObstacles(int n) {
255             obstacles.clear();
256 
257             // constants for threshold
258             int cx = (int) getWidth() / 2;
259             int cy = (int) getHeight() / 2;
260             Random rng = new Random();
261 
262             for (int i = 0; i < n; ++i) {
263 
264                 // create bounds for obstacles
265                 int cxl = cx / 2; // left
266                 int cxr = cx + cxl; // right
267                 int cyt = cy / 2; // top
268                 int cyb = cy + cyt; // bottom
269 
270                 // create coordinates and scale value for polygon
271                 int x = cxl + rng.nextInt((cxr - cxl) - 1);
272                 int y = cyt + rng.nextInt((cyb - cyt) - 1);
273                 int scale = 5 + rng.nextInt(7);
```

```
274                 int deg = rng.nextInt(360);
275                 Color c = new Color(
276                         (Math.random() * 255) / 255.0,
277                         (Math.random() * 255) / 255.0,
278                         (Math.random() * 255) / 255.0,
279                         1.0
280                 );
281
282                 // until I replace it with reflection, this'll work
283                 Polygon[] pool = new Polygon[] {
284                         new Obstacles.Octagon(x, y, scale),
285                         new Obstacles.Pentagon(x, y, scale)
286                 };
287
288                 // create polygon, check if touching start/end
289                 Polygon o = pool[rng.nextInt(pool.length)];
290                 o.setFill(c);
291                 o.setRotate(deg);
292
293                 // if the obstacle touches anything important we will try this
                        iteration again
294                 if (intersecting(o, start.getShape())
295                         || intersecting(o, goal.getShape())
296                         || intersecting(o, robot.getShape())) {
297                     i--;
298                     continue;
299                 }
300
301                 obstacles.add(o);
302             }
303
304             getChildren().addAll(obstacles);
305         }
306
307         public void spawnSGSNodes() {
308             // constants in integers for nice neat movement
309             int w = (int) (getWidth() / 12);
310             int h = (int) (getHeight() / 10);
311
312             // start in TL, goal in BR
313             start = new SNode(w, h);
314             goal = new SNode((int) getWidth() - w, (int) getHeight() - h);
315             goal.setColor(Color.DARKGOLDENROD);
316
317             // add node shapes to scene for visualization
318             getChildren().addAll(start.getShape(), goal.getShape());
319         }
320
321         public void newScenario() {
322             // cleanup all shapes on the scene
323             getChildren().clear();
324
325             // spawn all required entities for a new scenario
326             spawnSGSNodes();
327             spawnRobot();
328             spawnObstacles(5);
329         }
```

```
330
331         public void cleanup() {
332             getChildren().clear();
333             obstacles.clear();
334             spawnSGSNodes();
335             spawnRobot();
336         }
337
338         public void solve() {
339             if (obstacles.size() == 0) {
340                 return;
341             }
342
343             // our open & closed lists
344             PriorityQueue<SNode> open = new PriorityQueue<>(new SNodeComparator());
345             ArrayList<SNode> closed = new ArrayList<>();
346
347             start.setG(0);
348             start.setF(SNode.distanceTo(start, goal));
349
350             open.add(start);
351
352             while (!open.isEmpty()) {
353                 SNode current = open.poll();
354                 closed.add(current);
355                 robot.setXY(current.getX(), current.getY());
356
357                 // TODO: toggle so you can switch on 'generous' detection of goal
                        (using robot shape)
358                 //  if (current.equals(goal) ||
                        robot.getShape().contains(goal.getPoint2D()) ||
                        robot.hit(goal.getShape())) {
359                 if (current.equals(goal) ||
                        robot.getShape().contains(goal.getPoint2D())
360                         || intersecting(current.getShape(), goal.getShape())
361                         || robot.hit(goal.getShape())) {
362                     goal.setParent(current);
363                     regurgitate(goal);
364                     return;
365                 }
366
367                 for (Direction d : Direction.values()) {
368                     SNode n = new SNode(robot.getX() + (stepSize * d.dx),
                            robot.getY() + (stepSize * d.dy));
369                     if (closed.contains(n) || robot.collides(d, obstacles))
                            continue;
370
371                     // create tentative G
372                     double tempG = current.getG() + SNode.distanceTo(current, n);
373
374                     // if not in open list then add
375                     if (!open.contains(n)) {
376                         open.add(n);
377                     } else if (tempG >= n.getG()) { // not a better path, forget it
378                         continue;
379                     }
380
```

```
381                     // set parent, F, G, and H score with supposed tie breaking
382                     double tempH = SNode.distanceTo(n, goal) * (1.0 + (1.0 /
                            1000.0)); // tie breaking
383                     n.setParent(current);
384                     n.setG(tempG);
385                     n.setF(n.getG() + tempH);
386                     Shape nShape = n.getShape();
387
388                     getChildren().add(nShape);
389                 }
390             }
391         }
392
393         public void regurgitate(SNode n) {
394             // Polyline for visualization, path for transition
395             Polyline line = new Polyline();
396             Path path = new Path();
397
398             // path, move to initial goal point and work backwards
399             path.getElements().add(new MoveTo(goal.getX(), goal.getY()));
400             path.getElements().add(new LineTo(n.getX(), n.getY()));
401
402             // line, same with path, start with end node and work backwards
403             line.getPoints().addAll(
404                     Double.valueOf(n.getX()), Double.valueOf(n.getY())
405             );
406
407             // continue adding pieces of the path until we run into null parent
                    (starting point)
408             while (n.getParent() != null) {
409                 n = n.getParent();
410                 line.getPoints().addAll(
411                         Double.valueOf(n.getX()), Double.valueOf(n.getY())
412                 );
413                 path.getElements().add(new LineTo(n.getX(), n.getY()));
414             }
415
416             // spiffy up our line
417             line.setStrokeWidth(4);
418             line.setStroke(Color.RED);
419             line.toFront();
420             line.setStrokeLineCap(StrokeLineCap.ROUND);
421             getChildren().add(line);
422
423             // create a basic looping transition of the path
424             final PathTransition pathtransition = new PathTransition();
425             pathtransition.setDuration(Duration.seconds(10));
426             pathtransition.setDelay(Duration.seconds(0.5));
427             pathtransition.setPath(path);
428             pathtransition.setNode(robot.getShape());
429             robot.getShape().toFront();
430             pathtransition.setCycleCount(Timeline.INDEFINITE);
431             pathtransition.setAutoReverse(true);
432             pathtransition.play();
433         }
434     }
435 }
```

2 SNode

SNode is a replacement for using raw x & y coordinates. Instead it stores the coordinates, the default movement cost in the four cardinal directions (d), as well as the cost for moving diagonally ($d2$), the previous node and lastly the heuristic value (h).

```
1  import javafx.geometry.Point2D;
2  import javafx.scene.paint.Color;
3  import javafx.scene.shape.Circle;
4  import javafx.scene.shape.Shape;
5
6  public class SNode implements Comparable<SNode> {
7      public static final double d = 10.0;
8      public static final double d2 = 14.0;
9      public static final double defaultSize = 2.0;
10     private final int x, y;
11     private double f, h;
12     private double g = d;
13     private SNode parent = null;
14     private boolean obstacle = false;
15     private Color color = Color.LIGHTBLUE;
16     private Point2D pt;
17
18     public SNode(int x, int y) {
19         this.x = x;
20         this.y = y;
21         this.pt = new Point2D(x, y);
22         this.f = this.g = this.h = Double.MAX_VALUE;
23     }
24
25     public SNode(Point2D pt) {
26         this.x = (int) pt.getX();
27         this.y = (int) pt.getY();
28         this.pt = new Point2D(x, y);
29     }
30
31     public SNode(int x, int y, SNode goal) {
32         this.x = x;
33         this.y = y;
34         this.pt = new Point2D(x, y);
35         setF(goal);
36     }
37
38     public int getX() {
39         return x;
40     }
41
42     public int getY() {
43         return y;
44     }
45
46     public Point2D getPoint2D() {
47         // return new Point2D(x, y);
48         return pt;
49     }
50
```

```
51      static public double distanceTo(SNode f, SNode t) {
52          double dx = Math.abs(f.x - t.x);
53          double dy = Math.abs(f.y-  t.y);
54          return d * (dx + dy) + (d2 - 2 * d) * Math.min(dx, dy);
55      }
56
57      public double distanceTo(SNode snode) {
58          return Math.abs(x - snode.getX())  + Math.abs(y - snode.getY());
59      }
60
61      public double distanceTo(double x, double y) {
62          return Math.abs(this.x - x)  + Math.abs(this.y - y);
63      }
64
65      /**
66       * Sets parent to previous node, calculates heuristic based on goal node. Adds
             default cost to previous node cost.
67       * @param previous node
68       * @param goal node
69       */
70      public void setF(SNode previous, SNode goal) {
71          this.parent = previous;
72          h = distanceTo(goal);
73          g = previous.getG() + d;
74          f = g + h;
75      }
76
77      public void setF(SNode goal) {
78          h = distanceTo(goal.getX(), goal.getY());
79          f = g + h;
80      }
81
82      public void setF(double x, double y) {
83          h = distanceTo(x, y);
84          f = g + h;
85      }
86
87      public void setF(double f) {
88          this.f = f;
89      }
90
91      public double getF() {
92          return f;
93      }
94
95      public void setH(SNode goal) {
96          this.h = distanceTo(goal.getX(), goal.getY());
97      }
98
99      public double getH() {
100         return h;
101     }
102
103     public void setG(double g) {
104         this.g = g;
105     }
106
```

```
107     public double getG() {
108         return g;
109     }
110
111     public void setColor(Color color) {
112         this.color = color;
113     }
114
115     public void setObstacle(boolean obstacle) {
116         this.obstacle = obstacle;
117     }
118
119     public boolean isObstacle() {
120         return obstacle;
121     }
122
123     public Circle getShape() {
124         Circle c = new Circle(x, y, defaultSize, color);
125         c.toBack();
126         return c;
127     }
128
129     public Circle getShape(double size) {
130         Circle c = new Circle(x, y, size, color);
131         c.toBack();
132         return c;
133     }
134
135     public boolean containedBy(Shape p) {
136         return p.contains(getPoint2D());
137     }
138
139     public SNode getParent() {
140         return parent;
141     }
142
143     public void setParent(SNode parent) {
144         this.parent = parent;
145     }
146
147     public boolean sameAs(SNode s) {
148         return getX() == s.getX() && getY() == s.getY();
149     }
150
151     @Override
152     public boolean equals(Object o) {
153         if (this == o) return true;
154         if (!(o instanceof SNode)) return false;
155         SNode s = (SNode) o;
156         return (this.x == s.x) && (this.y == s.y);
157     }
158
159     @Override
160     public String toString() {
161         return new String("SN " + getX() + ":" + getY());
162     }
163
```

```
164     public void setH(double h) {
165         this.h = h;
166     }
167 
168     @Override
169     public int compareTo(SNode o) {
170         if (f < o.getF()) {
171             return -1;
172         } else if (f == o.getF()) {
173             return 0;
174         } else {
175             return 1;
176         }
177     }
178 }
```

3 Robot

Robot is much like the **SNode** class. It stores a shape (*shape*) and the coordinates of the current location (x & y). Movement and scaling is also handled by the class. Collision is also implemented; if the shape the robot is represented as collides with any specified polygons in a direction, or a single polygon without a movement direction.

```
1  import javafx.scene.shape.Polygon;
2  import javafx.scene.shape.Shape;
3  import javafx.scene.transform.Scale;
4
5  import java.util.ArrayList;
6
7  public class Robot {
8      private int x, y;
9      private Shape shape;
10
11     public Robot(Polygon shape) {
12         this.shape = shape;
13     }
14
15     public Robot(int x, int y, Shape shape) {
16         this.x = x;
17         this.y = y;
18         this.shape = shape;
19         resetScale();
20     }
21
22     public void setShape(Shape shape) {
23         this.shape = shape;
24         resetScale();
25     }
26
27     public int getX() {
28         return this.x;
29     }
30
31     public int getY() {
32         return this.y;
33     }
34
35     public void moveX(double dx) {
36         this.x += dx;
37         shape.setTranslateX(x);
38     }
39
40     public void moveY(double dy) {
41         this.y += dy;
42         shape.setTranslateY(y);
43     }
44
45     public void setXY(int x, int y) {
46         this.x = x;
47         this.y = y;
48         shape.setTranslateX(x);
49         shape.setTranslateY(y);
```

```
50      }
51
52      // TODO: transforms need to be on origin (center), or in case of triangle
            possibly corner
53      public void setScale(double size) {
54          if (size >= 1) {
55              Scale scale = new Scale(size, size);
56              shape.getTransforms().clear();
57              shape.getTransforms().add(scale);
58              // TODO: need to adjust for origin EVENTUALLY using middle point of
                    bounding box
59          }
60      }
61
62      public void resetScale() {
63          shape.getTransforms().clear();
64      }
65
66      public Shape getShape() {
67          return shape;
68      }
69
70      public boolean collides(AStar.AStarSimple.Direction direction,
            ArrayList<Polygon> polygons) {
71          // get before coordinates
72          int bx = x;
73          int by = y;
74
75          // do temporary move in direction
76          moveX(direction.dx);
77          moveY(direction.dy);
78
79          // check shapes for intersection
80          boolean collision = false;
81          for (Polygon p : polygons) {
82              if (hit(p)) {
83                  collision = true;
84                  break;
85              }
86          }
87
88          // move back
89          setXY(bx, by);
90
91          return collision;
92      }
93
94      public boolean hit(Shape shape) {
95          Shape xs = Shape.intersect(this.shape, shape);
96          if (xs.getBoundsInLocal().getWidth() != -1)
97              return true; // width is > 0, therefore an intersection has occurred
98          else
99              return false;
100     }
101 }
```

4 Obstacles

Obstacles is simply a collection of static polygons that can be scaled by an initial value. Since obstacles are typically not renewed, they feature no ability to rescale them after initialization.

```
1  import javafx.scene.paint.Color;
2  import javafx.scene.shape.Polygon;
3
4  // TODO: Move obstacles to abstract class to take advantage of scaling vars and
       whatnot?
5  public class Obstacles {
6      static class Rectangle extends Polygon {
7          public Rectangle(double x, double y, double scale) {
8              getPoints().addAll(
9                      0d, 0d,
10                     10d * scale, 0d,
11                     10d * scale, 10d * scale,
12                     0d, 10d * scale
13             );
14
15             // stroke & fill
16             setStrokeWidth(1);
17             setStroke(Color.BLACK);
18             setFill(Color.LIGHTGRAY);
19
20             // translate
21             this.setTranslateX(x);
22             this.setTranslateY(y);
23         }
24     }
25
26     static class Pentagon extends Polygon {
27         public Pentagon(double x, double y, double scale) {
28             getPoints().addAll(
29                     0d, -5d * scale,
30                     10d * scale, 5d * scale,
31                     7d * scale, 15d * scale,
32                     -7d * scale, 15d * scale,
33                     -10d * scale, 5d * scale
34             );
35
36             // stroke & fill
37             setStrokeWidth(1);
38             setStroke(Color.BLACK);
39             setFill(Color.LIGHTGRAY);
40
41             // set translated location based on x,y coordinates
42             this.setTranslateX(x);
43             this.setTranslateY(y);
44         }
45     }
46
47     static class Octagon extends Polygon {
48         public Octagon(double x, double y, double scale) {
49             getPoints().addAll(
50                     2.5d, -7d,
```

```
51                      7d, -2.5d,
52                      7d, 2.5d,
53                      2.5, 7d,
54                      -2.5d, 7d,
55                      -7d, 2.5d,
56                      -7d, -2.5d,
57                      -2.5d, -7d
58              );
59              for (int i = 0; i < getPoints().size(); i++) {
60                  getPoints().set(i, getPoints().get(i) * scale);
61              }
62
63              // stroke & fill
64              setStrokeWidth(1);
65              setStroke(Color.BLACK);
66              setFill(Color.LIGHTGRAY);
67
68              // set translated location based on x,y coordinates
69              this.setTranslateX(x);
70              this.setTranslateY(y);
71          }
72      }
73  }
```