A Star

Casey Weed

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

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

```
103                 }
104             }
105         });
106     }
107
108     static class AStarSimple extends Pane {
109         // objs
110         public SNode start;
111         public SNode goal;
112         public Robot robot;
113
114         // misc
115         public double robotSize = 1;
116         public int shapeChoice = 0;
117         public int stepSize = 5;
118
119         // obstacles
120         public ArrayList<Polygon> obstacles = new ArrayList<>();
121
122         // Directions for delta x,y when checking neighbors
123         public enum Direction {
124
125             TOPLEFT(-1, -1),
126             TOPRIGHT(1, -1),
127             BOTTOMLEFT(-1, 1),
128             BOTTOMRIGHT(1, 1),
129             LEFT(-1, 0),
130             RIGHT(1, 0),
131             UP(0, -1),
132             DOWN(0, 1);
133
134             public final int dx;
135             public final int dy;
136
137             Direction(int dx, int dy) {
138                 this.dx = dx;
139                 this.dy = dy;
140             }
141         }
142
143         // TODO: When changing size check if you collide, if you do DO NOT increment
144         public void incRobotSize(double step) {
145             double before = robotSize;
146             robotSize += step > 0 ? step : 1;
147             robot.setScale(robotSize);
148
149             // depracated; screen is cleared for any change whatsoever, better to
                    be safe than sorry though
150             for (Polygon o : obstacles) {
151                 if (intersecting(robot.getShape(), o)) {
152                     robotSize = before;
153                     robot.setScale(before);
154                 }
155             }
156         }
157
158         public void decRobotSize(double step) {
```

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

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

```
273                 // create coordinates and scale value for polygon
274                 int x = cxl + rng.nextInt((cxr - cxl) - 1);
275                 int y = cyt + rng.nextInt((cyb - cyt) - 1);
276                 int scale = 5 + rng.nextInt(7);
277                 int deg = rng.nextInt(360);
278                 Color c = new Color(
279                         (Math.random() * 255) / 255.0,
280                         (Math.random() * 255) / 255.0,
281                         (Math.random() * 255) / 255.0,
282                         1.0
283                 );
284
285                 // until I replace it with reflection, this'll work
286                 Polygon[] pool = new Polygon[] {
287                         new Obstacles.Octagon(x, y, scale),
288                         new Obstacles.Pentagon(x, y, scale)
289                 };
290
291                 // create polygon, check if touching start/end
292                 Polygon o = pool[rng.nextInt(pool.length)];
293                 o.setFill(c);
294                 o.setRotate(deg);
295
296                 // if the obstacle touches anything important we will try this
                        iteration again
297                 if (intersecting(o, start.getShape())
298                         || intersecting(o, goal.getShape())
299                         || intersecting(o, robot.getShape())) {
300                     i--;
301                     continue;
302                 }
303
304                 obstacles.add(o);
305             }
306
307             getChildren().addAll(obstacles);
308         }
309
310         public void spawnSGSNodes() {
311             // constants in integers for nice neat movement
312             int w = (int) (getWidth() / 12);
313             int h = (int) (getHeight() / 10);
314
315             // start in TL, goal in BR
316             start = new SNode(w, h);
317             goal = new SNode((int) getWidth() - w, (int) getHeight() - h);
318             goal.setColor(Color.DARKGOLDENROD);
319
320             // add node shapes to scene for visualization
321             getChildren().addAll(start.getShape(), goal.getShape());
322         }
323
324         public void newScenario() {
325             // cleanup all shapes on the scene
326             getChildren().clear();
327
328             // spawn all required entities for a new scenario
```

```
329             spawnSGSNodes();
330             spawnRobot();
331             spawnObstacles(5);
332         }
333 
334         public void cleanup() {
335             getChildren().clear();
336             obstacles.clear();
337             spawnSGSNodes();
338             spawnRobot();
339         }
340 
341         public void solve() {
342             if (obstacles.size() == 0 || robot == null) {
343                 return;
344             }
345 
346             // our open & closed lists
347             PriorityQueue<SNode> open = new PriorityQueue<>(new SNodeComparator());
348             ArrayList<SNode> closed = new ArrayList<>();
349 
350             start.setG(0);
351             start.setF(SNode.distanceTo(start, goal));
352 
353             open.add(start);
354 
355             while (!open.isEmpty()) {
356                 SNode current = open.poll();
357                 closed.add(current);
358                 robot.setXY(current.getX(), current.getY());
359 
360                 // TODO: toggle so you can switch on 'generous' detection of goal
                        (using robot shape)
361                 //  if (current.equals(goal) ||
                        robot.getShape().contains(goal.getPoint2D()) ||
                        robot.hit(goal.getShape())) {
362                 if (current.equals(goal) ||
                        robot.getShape().contains(goal.getPoint2D())
363                         || intersecting(current.getShape(), goal.getShape())
364                         || robot.hit(goal.getShape())) {
365                     goal.setParent(current);
366                     regurgitate(goal);
367                     return;
368                 }
369 
370                 for (Direction d : Direction.values()) {
371                     SNode n = new SNode(robot.getX() + (stepSize * d.dx),
                            robot.getY() + (stepSize * d.dy));
372                     if (closed.contains(n) || robot.collides(d, obstacles))
                            continue;
373 
374                     // create tentative G
375                     double tempG = current.getG() + SNode.distanceTo(current, n);
376 
377                     // if not in open list then add
378                     if (!open.contains(n)) {
379                         open.add(n);
```

```
380                     } else if (tempG >= n.getG()) { // not a better path, forget it
381                         continue;
382                     }
383
384                     // set parent, F, G, and H score with supposed tie breaking
385                     double tempH = SNode.distanceTo(n, goal) * (1.0 + (1.0 /
                            1000.0)); // tie breaking
386                     n.setParent(current);
387                     n.setG(tempG);
388                     n.setF(n.getG() + tempH);
389                     Shape nShape = n.getShape();
390
391                     getChildren().add(nShape);
392                 }
393             }
394         }
395
396         public void regurgitate(SNode n) {
397             // Polyline for visualization, path for transition
398             Polyline line = new Polyline();
399             Path path = new Path();
400
401             // path, move to initial goal point and work backwards
402             path.getElements().add(new MoveTo(goal.getX(), goal.getY()));
403             path.getElements().add(new LineTo(n.getX(), n.getY()));
404
405             // line, same with path, start with end node and work backwards
406             line.getPoints().addAll(
407                     Double.valueOf(n.getX()), Double.valueOf(n.getY())
408             );
409
410             // continue adding pieces of the path until we run into null parent
                    (starting point)
411             while (n.getParent() != null) {
412                 n = n.getParent();
413                 line.getPoints().addAll(
414                         Double.valueOf(n.getX()), Double.valueOf(n.getY())
415                 );
416                 path.getElements().add(new LineTo(n.getX(), n.getY()));
417             }
418
419             // spiffy up our line
420             line.setStrokeWidth(4);
421             line.setStroke(Color.RED);
422             line.toFront();
423             line.setStrokeLineCap(StrokeLineCap.ROUND);
424             getChildren().add(line);
425
426             // create a basic looping transition of the path
427             final PathTransition pathtransition = new PathTransition();
428             pathtransition.setDuration(Duration.seconds(10));
429             pathtransition.setDelay(Duration.seconds(0.5));
430             pathtransition.setPath(path);
431             pathtransition.setNode(robot.getShape());
432             robot.getShape().toFront();
433             pathtransition.setCycleCount(Timeline.INDEFINITE);
434             pathtransition.setAutoReverse(true);
```

```
435                 pathtransition.play();
436             }
437         }
438 }
```

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  }
```