

```

1 package maze.solvers;
2
3 import java.text.DecimalFormat;
4
5
6
7
8
9
10 /**
11  * A-Star (Lee) algorithm for maze solving
12  *
13  * @author Pierre-Andre Mudry, Romain Cherix
14  * @date February 2012
15  * @version 1.2
16  *
17  */
18 public class AStar {
19
20     private MazeElem[][] maze;
21     private int width, height;
22     private int[][] solution;
23
24     // Debug information
25     public final boolean VERBOSE = true;
26
27     private AStar(MazeContainer mazeContainer) {
28         maze = mazeContainer.maze;
29         width = mazeContainer.nCellsX;
30         height = mazeContainer.nCellsY;
31     }
32
33     /**
34     * Solves the maze
35     *
36     * @param x The x-coordinate of the start point
37     * @param y The y-coordinate of the start point
38     */
39     private void solve(int x, int y) {
40         /**
41         * The solution at the beginning is an array full of zeroes
42         */
43         solution = new int[width][height];
44
45         // We indicate the starting position
46         solution[x][y] = 1;
47
48         // This is the step counter
49         int m = 1;
50
51         /**
52         * Do the expansion until we have reached the exit.
53         */
54         while (expansion(m) == false) {
55             m++;
56         }
57
58         /**
59         * m contains the total number of steps to find the solution
60         */
61         if (VERBOSE)
62             System.out.println("\n[AStar solver] Took " + m + " steps for the solution\n");
63
64         /**
65         * As the forward propagation is over, we can now do the back-prop
66         * phase.
67         */
68         backtrace(m);
69     }
70
71     /**
72     * Lee forward propagation algorithm
73     *
74     * @param m The current step of the algorithm
75     * @return A boolean value that indicates if wave has hit exit
76     */
77     private boolean expansion(int m) {
78
79         for (int j = 0; j < height; j++) {
80             for (int i = 0; i < width; i++) {
81
82                 /**
83                 * At each step m, we propagate the wave for each cell of the
84                 * solution that has the index m.
85                 */
86                 if (solution[i][j] == m) {
87                     if (!maze[i][j].wallWest)
88                         if (maze[i][j].isExit) {
89                             return true;
90                         } else if (solution[i - 1][j] == 0)
91                             solution[i - 1][j] = m + 1;

```

```

92
93         if (!maze[i][j].wallNorth)
94             if (maze[i][j].isExit) {
95                 return true;
96             } else if (solution[i][j - 1] == 0)
97                 solution[i][j - 1] = m + 1;
98
99         if (!maze[i][j].wallEast)
100             if (maze[i][j].isExit) {
101                 return true;
102             } else if (solution[i + 1][j] == 0)
103                 solution[i + 1][j] = m + 1;
104
105         if (!maze[i][j].wallSouth)
106             if (maze[i][j].isExit) {
107                 return true;
108             } else if (solution[i][j + 1] == 0)
109                 solution[i][j + 1] = m + 1;
110     }
111 }
112
113     return false;
114 }
115 }
116
117 /**
118  * Grants uniform access for the whole maze and makes sure that we do not
119  * cross the borders of the maze
120  *
121  * @param i x position
122  * @param j y position
123  * @return distance to the origin point, -1 if outside the graph
124  */
125 private int access_solution(int i, int j) {
126     if (i >= width || i < 0 || j >= height || j < 0)
127         return -1;
128     else
129         return solution[i][j];
130 }
131
132 /**
133  * Lee algorithm back-trace phase when the array has been annotated with the
134  * distances
135  *
136  * @param m The highest distance from origin point
137  */
138 private void backtrace(int m) {
139     int[][] ret = new int[width][height];
140
141     int x = 0, y = 0;
142
143     // Get the coordinates of exit in original maze
144     for (int j = 0; j < height; j++) {
145         for (int i = 0; i < width; i++) {
146             if (maze[i][j].isExit) {
147                 x = i;
148                 y = j;
149                 break;
150             }
151         }
152     }
153
154     // The exit is part of the solution
155     ret[x][y] = 1;
156
157     /**
158      * While we haven't reached the beginning, annotate the solution with
159      * the correct path
160      */
161     while (m > 0) {
162         if (access_solution(x - 1, y) == m && !maze[x][y].wallWest)
163             ret[--x][y] = 1;
164
165         if (access_solution(x, y - 1) == m && !maze[x][y].wallNorth)
166             ret[x][--y] = 1;
167
168         if (access_solution(x + 1, y) == m && !maze[x][y].wallEast)
169             ret[++x][y] = 1;
170
171         if (access_solution(x, y + 1) == m && !maze[x][y].wallSouth)
172             ret[x][++y] = 1;
173
174         m--;
175     }
176
177     // Update the solution with the backprop version

```

```

178     solution = ret;
179 }
180
181 /**
182  * Displays the solution on the console for control
183  */
184 public static void displaySolution(int[][] mazeSolution) {
185     String solutionText = "";
186
187     int width = mazeSolution[0].length;
188     int height = mazeSolution.length;
189
190     if (mazeSolution != null)
191         for (int j = 0; j < width; j++) {
192             for (int i = 0; i < height; i++) {
193
194                 DecimalFormat myFormatter = new DecimalFormat("00");
195                 String s = myFormatter.format(mazeSolution[i][j]);
196
197                 if (i != height - 1)
198                     solutionText += s + " - ";
199                 else
200                     solutionText += s;
201             }
202             solutionText += "\n";
203         }
204
205     System.out.println(solutionText);
206 }
207
208 /**
209  * This class is thought to be used statically using only this method
210  *
211  * @param mc The {@link MazeContainer} that we want to solve
212  * @param x The x-coordinate of the start point
213  * @param y The y-coordinate of the start point
214  * @return An array containing 1's along the solution path
215  */
216 public static int[][] solve(MazeContainer mc, int x, int y) {
217     AStar alg = new AStar(mc);
218     alg.solve(x, y);
219     return alg.solution;
220 }
221
222 public static void main(String args[]) {
223     /**
224      * Create a maze and display its textual representation
225      */
226     MazeContainer mc = new MazeContainer(50, 80);
227     TextDisplay.displayMaze(mc);
228
229     /**
230      * Compute a solution and display it
231      */
232     int[][] solution = AStar.solve(mc, 0, 0);
233     AStar.displaySolution(solution);
234
235     GraphicDisplay gd = new GraphicDisplay(mc, 2, false);
236     gd.setSolution(solution);
237
238 }
239
240

```