

```

1 package maze.display;
2
3 import hevsgraphics.ImageGraphicsMultiBuffer;
4
5 /**
6  * A graphic view of a {@link MazeContainer}
7  *
8  * @author Pierre-Andre Mudry
9  * @version 1.5
10 * @date February 2012
11 */
12 public class GraphicDisplay {
13
14     // The number of cells
15     public final int nCellsX, nCellsY;
16
17     /**
18      * Window and drawing related
19      */
20     // Dimensions (in pixels) of each cell
21     public final int wCell;
22     public final int hCell;
23
24     // Size of the whole screen
25     public final int frameWidth, frameHeight;
26
27     // Shall we draw the grid ?
28     boolean drawGrid = false;
29
30     // Size of the stroke (grid and maze)
31     private int strokeSize = 7;
32
33     /**
34      * UI related
35      */
36     // The logo
37     private BufferedImage mBitmap;
38
39     // The message at the bottom of the screen
40     private String msg;
41
42     // Contains the maze that we will display
43     private MazeContainer mazeContainer;
44
45     // Contains the Display that is used to show the maze
46     public Display disp;
47
48     int[][] solution;
49
50     /**
51      * FIXME
52      * @param kl
53      */
54     public void registerKeyListener (KeyboardListener kl) {
55         disp.registerKeyListener (kl);
56     }
57
58     /**
59      * Sets the message that will be displayed at the bottom of the screen
60      *
61      * @param msg
62      */
63     public void setMessage (String msg) {
64         disp.setMessage (msg);
65     }
66
67     /**
68      * Sets a new maze for display
69      *
70      * @param mc
71      */
72     public void setNewMaze (MazeContainer mc) {
73         this.mazeContainer = mc;
74     }
75
76     public class Display extends ImageGraphicsMultiBuffer {
77         private static final long serialVersionUID = 1L;
78
79         public Display (String title, int width, int height, boolean hasDecoration) {
80             super (title, width, height, hasDecoration);
81         }
82
83         public void registerKeyListener (KeyListener kl) {
84             super.mainFrame.addKeyListener (kl);
85         }
86     }
87 }

```

```

102     /**
103      * Sets the text that is displayed at the bottom of the screen
104      *
105      * @param msg
106      */
107     public void setMessage(String message) {
108         msg = message;
109     }
110
111     /**
112      * Does the rendering process for the maze
113      */
114     @Override
115     public void render(Graphics2D g) {
116
117         /**
118          * Take the borders into account if we are rendering with Swing
119          * decoration
120          */
121         int border_top = this.mainFrame.getInsets().top;
122         int border_left = this.mainFrame.getInsets().left;
123
124         int xs = border_left + strokeSize / 2, ys = border_top + strokeSize / 2;
125
126         // Set the pen size using the stroke
127         g.setStroke(new BasicStroke(strokeSize));
128
129         /**
130          * Grid drawing
131          */
132         if (drawGrid) {
133             g.setColor(new Color(220, 220, 220));
134
135             // Horizontal grid lines
136             for (int i = 0; i < nCellsY + 1; i++) {
137                 g.drawLine(0, ys, frameWidth - strokeSize + border_top, ys);
138                 ys += hCell + strokeSize;
139             }
140
141             // Vertical grid lines
142             for (int i = 0; i < nCellsX + 1; i++) {
143                 g.drawLine(xs, 0, xs, frameHeight - strokeSize + border_top);
144                 xs += wCell + strokeSize;
145             }
146         }
147
148         /**
149          * Draw the content of the maze
150          */
151         g.setColor(Color.BLACK);
152         xs = border_left + strokeSize / 2;
153         ys = border_top + strokeSize / 2;
154
155         // Draw the solution if required
156         if (solution != null) {
157             for (int i = 0; i < nCellsX; i++) {
158                 for (int j = 0; j < nCellsY; j++) {
159                     MazeElem e = mazeContainer.maze[i][j];
160
161                     // Draw solution
162                     if (solution != null && solution[i][j] == 1) {
163                         g.setColor(new Color(200, 200, 250));
164                         g.fillRect(xs, ys, wCell + strokeSize, hCell + strokeSize);
165                         g.setColor(Color.black);
166                     }
167                     ys += hCell + strokeSize;
168                 }
169
170                 ys = border_top + strokeSize / 2;
171                 xs += wCell + strokeSize;
172             }
173         }
174
175         xs = border_left + strokeSize / 2;
176         ys = border_top + strokeSize / 2;
177
178         // Draw the content of the frames
179         for (int i = 0; i < nCellsX; i++) {
180             // draw the north edge
181             for (int j = 0; j < nCellsY; j++) {
182                 MazeElem e = mazeContainer.maze[i][j];
183
184                 // Draw exit
185                 if (e.isExit) {
186                     g.setColor(new Color(100, 100, 200));
187                     g.fillRect(

```

```

188         xs + int Math.round strokeSize / 2.0, ys + int Math.round strokeSize / 2.0,
189         wCell, hCell);
190     g.setColor Color.black;
191 }
192
193 // Draw position for player 1
194 if (e.p1Present) {
195     g.setColor Color.red;
196     g.fillOval(
197         xs + int Math.round strokeSize / 2.0, ys + int Math.round strokeSize / 2.0,
198         wCell, hCell);
199     g.setColor Color.black;
200     g.setStroke new BasicStroke(1.0f));
201     g.drawOval(
202         xs + int Math.round strokeSize / 2.0, ys + int Math.round strokeSize / 2.0,
203         wCell, hCell);
204     g.setStroke new BasicStroke(strokeSize);
205 }
206
207 if (e.p2Present) {
208     g.setColor Color.yellow;
209     g.fillOval(
210         xs + int Math.round strokeSize / 2.0, ys + int Math.round strokeSize / 2.0,
211         wCell, hCell);
212     g.setColor Color.black;
213     g.setStroke new BasicStroke(1.0f));
214     g.drawOval(
215         xs + int Math.round strokeSize / 2.0, ys + int Math.round strokeSize / 2.0,
216         wCell, hCell);
217     g.setStroke new BasicStroke(strokeSize);
218 }
219
220 // Is there a north wall ?
221 if (e.wallNorth) {
222     g.drawLine xs, ys, xs + wCell + strokeSize, ys;
223 }
224
225 // Is there a left wall ?
226 if (e.wallWest) {
227     g.drawLine xs, ys, xs, ys + hCell + strokeSize;
228 }
229
230 // Draw bottom for the last line
231 if ((j == nCellsY - 1) && (e.wallSouth)) {
232     g.drawLine xs, ys + hCell + strokeSize, xs + wCell + strokeSize, ys + hCell + strokeSize;
233 }
234
235 // Draw right for the last column
236 if ((i == nCellsX - 1) && (e.wallEast)) {
237     g.drawLine xs + wCell + strokeSize, ys, xs + wCell + strokeSize, ys + hCell + strokeSize;
238 }
239
240 ys += hCell + strokeSize;
241 }
242
243 ys = border_top + strokeSize / 2;
244 xs += wCell + strokeSize;
245 }
246
247 /**
248  * Draw HES-SO logo, centered, at the bottom of the screen
249  */
250 g.drawImage mBitmap, fWidth / 2 - mBitmap.getWidth() / 2, fHeight - 30, null;
251
252 // Write some information message
253 if (msg != null)
254     g.drawString msg, 5, fHeight - 40;
255 }
256
257
258 /**
259  * This method is used to overlay a solution that has been found using one
260  * solver algorithm such as the one implemented in {@link AStar_BEGIN}
261  *
262  * @param solution The solution to overlay
263  */
264 public void setSolution(int[][] solution) {
265     assert solution.length == nCellsX;
266     assert solution[0].length == nCellsY;
267     this.solution = solution;
268 }
269
270 /**
271  * Call this method to remove the solution overlay
272  */
273 public void clearSolution() {

```

```

274         this.solution = null;
275     }
276
277     /**
278     * Loads an image into mBitmap
279     *
280     * @param imageName The image path to be loaded (relative to the src/bin
281     *                  folder), i.e. /images/...)
282     */
283     private void loadImage(String imageName) {
284         try {
285             mBitmap = ImageIO.read(SimpleGraphicsBitmap.class.getResource(imageName));
286
287         } catch (Exception e) {
288             System.out.println("Could not find image " + imageName + ", exiting !");
289             e.printStackTrace();
290             System.exit(-1);
291         }
292     }
293
294     /**
295     * @see GraphicDisplay
296     */
297     public GraphicDisplay(MazeContainer mc, int sizeOfSquare) {
298         this(mc, sizeOfSquare, true);
299     }
300
301     /**
302     * Display a window showing a {@link MazeContainer}
303     *
304     * @param mc The maze to show
305     * @param sizeOfSquare The width of each square to show
306     * @param decorations If we need the borders or not
307     */
308     public GraphicDisplay(MazeContainer mc, int sizeOfSquare, boolean decorations) {
309         mazeContainer = mc;
310
311         nCellsX = mc.nCellsX;
312         nCellsY = mc.nCellsY;
313
314         /**
315         * Compute the sizes for the graphical display
316         */
317         wCell = sizeOfSquare;
318         hCell = sizeOfSquare;
319
320         /**
321         * Size of the frame should have space for all the cells (nCellsX *
322         * wCell) and also space for the grid (hence the nCellsX + 1 *
323         * strokeWidth)
324         */
325         frameWidth = (nCellsX * wCell + ((nCellsX + 1) * strokeSize));
326         frameHeight = (nCellsY * hCell + ((nCellsY + 1) * strokeSize));
327
328         // Load the image
329         loadImage("/images/logo_hei.png");
330
331         // Create a display and keep some space for the picture and the text at
332         // the bottom
333         disp = new Display("Maze - Minilabor", frameWidth, frameHeight + 55, decorations);
334
335         // Sets the default message
336         disp.setMessage("Welcome to the Maze game !");
337     }
338
339     public static void main(String args[]) {
340         // Generate a maze
341         MazeContainer mc = new MazeContainer(20, 15);
342
343         // Display the maze
344         GraphicDisplay gd = new GraphicDisplay(mc, 15, true);
345     }
346
347

```