```java
package maze.display;

import hevs.graphics.ImageGraphicsMultiBuffer;

/**
 * A graphic view of a {@link MazeContainer}
 *
 * @author Pierre-Andre Mudry
 * @version 1.5
 * @date February 2012
 */
public class GraphicDisplay {

    // The number of cells
    public final int nCellsX, nCellsY;

    /**
     * Window and drawing related
     */
    // Dimensions (in pixels) of each cell
    public final int wCell;
    public final int hCell;

    // Size of the whole screen
    public final int frameWidth, frameHeight;

    // Shall we draw the grid ?
    boolean drawGrid = false;

    // Size of the stroke (grid and maze)
    private int strokeSize = 7;

    /**
     * UI related
     */
    // The logo
    private BufferedImage mBitmap;

    // The message at the bottom of the screen
    private String msg;

    // Contains the maze that we will display
    private MazeContainer mazeContainer;

    // Contains the Display that is used to show the maze
    public Display disp;

    int[][] solution;

    /**
     * FIXME
     * @param kl
     */
    public void registerKeyListener(KeyboardListener kl) {
        disp.registerKeyListener(kl);
    }

    /**
     * Sets the message that will be displayed at the bottom of the screen
     *
     * @param msg
     */
    public void setMessage(String msg) {
        disp.setMessage(msg);
    }

    /**
     * Sets a new maze for display
     *
     * @param mc
     */
    public void setNewMaze(MazeContainer mc) {
        this.mazeContainer = mc;
    }

    public class Display extends ImageGraphicsMultiBuffer {
        private static final long serialVersionUID = 1L;

        public Display(String title, int width, int height, boolean hasDecoration) {
            super(title, width, height, hasDecoration);
        }

        public void registerKeyListener(KeyListener kl) {
            super.mainFrame.addKeyListener(kl);
        }

```

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

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

```
272         this.solution = null;
273     }
274
275     /**
276      * Loads an image into mBitmap
277      *
278      * @param imageName The image path to be loaded (relative to the src/bin
279      *            folder), i.e. /images/...)
280      */
281     private void loadImage String imageName) {
282         try {
283             mBitmap = ImageIO.read SimpleGraphicsBitmap.class.getResource imageName));
284
285         } catch (Exception e) {
286             System.out.println("Could not find image " + imageName + ", exiting !");
287             e.printStackTrace();
288             System.exit(-1);
289         }
290     }
291
292     /**
293      * @see GraphicDisplay
294      */
295     public GraphicDisplay MazeContainer mc, int sizeOfSquare) {
296         this(mc, sizeOfSquare, true);
297     }
298
299     /**
300      * Display a window showing a {@link MazeContainer}
301      *
302      * @param mc The maze to show
303      * @param sizeOfSquare The width of each square to show
304      * @param decorations If we need the borders or not
305      */
306     public GraphicDisplay MazeContainer mc, int sizeOfSquare, boolean decorations) {
307         mazeContainer = mc;
308
309         nCellsX = mc.nCellsX;
310         nCellsY = mc.nCellsY;
311
312         /**
313          * Compute the sizes for the graphical display
314          */
315         wCell = sizeOfSquare;
316         hCell = sizeOfSquare;
317
318         /**
319          * Size of the frame should have space for all the cells (nCellsX *
320          * wCell) and also space for the grid (hence the nCellsX + 1 *
321          * strokeWidth)
322          */
323         frameWidth = (nCellsX * wCell + ((nCellsX + 1) * strokeSize));
324         frameHeight = (nCellsY * hCell + ((nCellsY + 1) * strokeSize));
325
326         // Load the image
327         loadImage("/images/logo_hei.png");
328
329         // Create a display and keep some space for the picture and the text at
330         // the bottom
331         disp = new Display("Maze - Minilabor", frameWidth, frameHeight + 55, decorations);
332
333         // Sets the default message
334         disp.setMessage("Welcome to the Maze game !");
335     }
336
337     public static void main String args[]) {
338         // Generate a maze
339         MazeContainer mc = new MazeContainer(10, 10);
340
341         // Display the maze
342         GraphicDisplay gd = new GraphicDisplay(mc, 10, false);
343     }
344 }
345
```