```java
package maze.display;

import maze.data.MazeContainer;

/**
 * A class that displays a textual version of the maze given in the form of a
 * {@link MazeContainer}
 *
 * @author Pierre-Andre Mudry
 * @date February 29th, 2012
 * @version 1.0
 */
public class TextDisplay {

    /**
     * Displays the maze given in parameter on the default console
     *
     * @param mazeC The {@link MazeContainer} to display
     */
    public static void displayMaze(MazeContainer mazeC) {

        // Get the real labyrinth
        MazeElem[][] maze = mazeC.maze;

        // Size of the labyrinth
        int nCellsX = mazeC.nCellsX;
        int nCellsY = mazeC.nCellsY;

        /**
         * Draw the labyrinth
         */
        for (int i = 0; i < nCellsY; i++) {
            // Draws the north edge
            for (int j = 0; j < nCellsX; j++) {
                MazeElem e = maze[j][i];

                // TODO Task 1

                System.out.print("*   ");
            }

            System.out.println("*");

            // Draws the west edge
            for (int j = 0; j < nCellsX; j++) {
                MazeElem e = maze[j][i];

                // TODO Task 1

                System.out.print("    ");
            }
            System.out.println("|");
        }
        // Draws the bottom line
        for (int j = 0; j < nCellsX; j++) {
            System.out.print("*---");
        }
        System.out.println("*");
    }

    public static void main(String[] args) {
        MazeContainer mg = new MazeContainer(6, 6);
        TextDisplay.displayMaze(mg);
    }
}
```

```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
```

```java
package maze.solvers;

import maze.data.MazeContainer;

/**
 * A-Star (Lee) algorithm for maze solving
 *
 * @author Pierre-Andre Mudry, Romain Cherix
 * @date February 2012
 * @version 1.2
 *
 */
public class AStar {

    private MazeElem[][] maze;
    private int width, height;
    private int[][] solution;

    // Debug information
    public final boolean VERBOSE = true;

    private AStar(MazeContainer mazeContainer) {
        maze = mazeContainer.maze;
        width = mazeContainer.nCellsX;
        height = mazeContainer.nCellsY;
    }

    /**
     * Solves the maze
     *
     * @param x
     *            The x-coordinate of the start point
     * @param y
     *            The y-coordinate of the start point
     */
    private void solve(int x, int y) {
        /**
         * The solution at the beginning is an array full of zeroes
         */
        solution = new int[width][height];

        // We indicate the starting position
        solution[x][y] = 1;

        // This is the step counter
        int m = 1;

        /**
         * Do the expansion until we have reached the exit.
         */
        while (expansion(m) == false) {
            m++;
        }

        /**
         * m contains the total number of steps to find the solution
         */
        if (VERBOSE)
            System.out.println("\n[AStar solver] Took " + m + " steps for the solution\n");

        /**
         * As the forward propagation is over, we can now do the back-prop
         * phase.
         */

        // TODO When you are confident your algorithm is working, you
        // can uncomment this line in order to have the backtrace done for you
        // backtrace(m);
    }

    /**
     * Lee forward propagation algorithm
     *
     * @param m
     *            The current step of the algorithm
     * @return A boolean value that indicates if wave has hit exit
     */
    private boolean expansion(int m) {

        // TODO Implement your algorithm here

        return true;
    }

    /**
     * Grants uniform access for the whole maze and makes sure that we do not
```

```
89      * cross the borders of the maze
90      *
91      * @param i
92      *            x position
93      * @param j
94      *            y position
95      * @return distance to the origin point, -1 if outside the graph
96      */
97     private int access_solution(int i, int j) {
98         if (i >= width || i < 0 || j >= height || j < 0)
99             return -1;
100        else
101            return solution[i][j];
102    }
103
104    /**
105     * Lee algorithm back-trace phase when the array has been annotated with the
106     * distances
107     *
108     * @param m
109     *            The highest distance from origin point
110     */
111    private void backtrace(int m) {
112        int[][] ret = new int[width][height];
113
114        int x = 0, y = 0;
115
116        // Get the coordinates of exit in original maze
117        for (int i = 0; i < width; i++) {
118            for (int j = 0; j < height; j++) {
119                if (maze[i][j].isExit) {
120                    x = i;
121                    y = j;
122                    break;
123                }
124            }
125        }
126
127        // The exit is part of the solution
128        ret[x][y] = 1;
129
130        /**
131         * While we haven't reached the beginning, annotate the solution with
132         * the correct path
133         */
134        while (m > 0) {
135            if (access_solution(x - 1, y) == m && !maze[x][y].wallWest)
136                ret[--x][y] = 1;
137
138            if (access_solution(x, y - 1) == m && !maze[x][y].wallNorth)
139                ret[x][--y] = 1;
140
141            if (access_solution(x + 1, y) == m && !maze[x][y].wallEast)
142                ret[++x][y] = 1;
143
144            if (access_solution(x, y + 1) == m && !maze[x][y].wallSouth)
145                ret[x][++y] = 1;
146
147            m--;
148        }
149
150        // Update the solution with the backprop version
151        solution = ret;
152    }
153
154    /**
155     * Displays the solution on the console for control
156     */
157    public static void displaySolution(int[][] mazeSolution) {
158        String solutionText = "";
159
160        int width = mazeSolution[0].length;
161        int height = mazeSolution.length;
162
163        if (mazeSolution != null)
164            for (int j = 0; j < width; j++) {
165
166                for (int i = 0; i < height; i++) {
167
168
169                    if (i != height - 1)
170                        solutionText += mazeSolution[i][j] + " - ";
171                    else
172                        solutionText += mazeSolution[i][j];
173                }
174                solutionText += "\n";
```

```
175             }
176
177         System.out.println solutionText ;
178     }
179
180     /**
181      * This class is thought to be used statically using only this method
182      *
183      * @param mc
184      *            The {@link MazeContainer} that we want to solve
185      * @param x
186      *            The x-coordinate of the start point
187      * @param y
188      *            The y-coordinate of the start point
189      * @return An array containing 1's along the solution path
190      */
191     public static int[][] solve MazeContainer mc, int x, int y  {
192         AStar alg = new AStar mc ;
193         alg.solve x, y ;
194         return alg.solution;
195     }
196
197     public static void main String args[]  {
198         /**
199          * Create a maze and display its textual representation
200          */
201         MazeContainer mc = new MazeContainer 4, 4 ;
202         TextDisplay.displayMaze mc ;
203
204         /**
205          * Compute a solution and display it
206          */
207         int[][] solution = AStar.solve mc, 0, 0 ;
208         AStar.displaySolution solution ;
209     }
210
211 }
```

```java
package maze;

import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.Timer;

import maze.data.MazeContainer;
import maze.data.MazeElem;
import maze.data.MazeUtils.Direction;
import maze.data.MazeUtils.Player;
import maze.display.GraphicDisplay;
import maze.display.KeyboardListener;
import maze.solvers.AStar;

/**
 * Game logic for moving the players and selectively showing the solution
 *
 * @author Pierre-Andre Mudry, Romain Cherix
 * @date February 2012
 * @version 1.4
 */
public class MazeGame {

    public MazeElem[][] maze;
    int width, height;

    GraphicDisplay gd;
    MazeContainer mc;
    KeyboardListener kl;

    /**
     * By default, you are the first player but this can change..
     */
    Player player = Player.PLAYER1;

    MazeGame MazeContainer mc) {
        gd = new GraphicDisplay(mc, 12);

        // Link key presses with the actions
        kl = new KeyboardListener(mc, this);
        gd.registerKeyListener(kl);

        setNewMaze(mc);
    }

    /**
     * Dynamically changes the maze that is displayed
     *
     * @param mc
     */
    public void setNewMaze MazeContainer mc) {
        maze = mc.maze;
        width = mc.nCellsX;
        height = mc.nCellsY;
        this.mc = mc;

        /**
         * Update graphical maze
         * FIXME : this should allow mazes of different sizes,
         * which is not the case now
         */
        gd.setNewMaze(mc);
    }

    /**
     * Displays the solution on the screen for a player during a whole second
     *
     * @param p Which player's solution do you want
     */
    public void displaySolution() {
        Point p1 = findPlayer(player);

        int[][] solution = AStar.solve(mc, p1.x, p1.y);
        gd.setSolution(solution);

        Timer timer = new Timer(1000, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                gd.clearSolution();
            }
        });

        timer.setRepeats(false);
        timer.start();
    }
```

```java

    /**
     * Call this when you want a new game
     * @param mazeID
     */
    public void generateNewMaze(int mazeID){
        this.setNewMaze(new MazeContainer(width, height, mazeID));
    }

    /**
     * Gives us the location of player inside the maze
     *
     * @param p The player we want
     * @return The location of the player
     */
    private Point findPlayer(Player p) {
        /**
         * We go through all the elements
         */
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {

                // TODO Complete this
                Point point = new Point(0,0);
                return point;

            }
        }

        // This means that the player hasn't been found
        // which can happen for instance in single player games
        return null;
    }

    /**
     * Check if some player has reached the exit of the maze
     */
    public boolean checkWinner() {

        // TODO Complete this

        return false;
    }

    /**
     * Method used to move a player inside the maze
     *
     * @param d Which direction do you want to go to ?
     */
    public void movePlayer(Direction d) {

        // TODO Complete this

    }

    public static void main(String[] args) {
        MazeContainer mc = new MazeContainer(10, 10);
        MazeGame mg = new MazeGame(mc);

        // TODO Students should implement next line
        // mg.movePlayer(Direction.DOWN);

        // This shows a nice message window
        // JOptionPane.showMessageDialog(null, "Title of the window", "Text of the window !",
JOptionPane.INFORMATION_MESSAGE);
    }
}

```

```java
package maze.display;

import java.awt.event.KeyEvent;

/**
 * Links key presses and actions
 *
 * @author Pierre-Andre Mudry
 * @date February 2012
 * @version 1.0
 */
public class KeyboardListener implements KeyListener {

    MazeGame mg;
    MazeContainer mc;

    /**
     * To link keys to actions from the game in the maze, we need references to
     * both
     *
     * @param mc The maze
     * @param mg The game
     */
    public KeyboardListener MazeContainer mc, MazeGame mg) {
        this.mc = mc;
        this.mg = mg;
    }

    /**
     * What happens when a key has been pressed
     */
    @Override
    public void keyPressed KeyEvent arg0) {

        /**
         * Keys for player 1
         */
        switch (arg0.getKeyCode()) {
        case KeyEvent.VK_A:
            System.out.println "You pressed the 'A' key";
            break;
        case KeyEvent.VK_F12:
            System.out.println "You pressed the F12 key";
            break;
        case KeyEvent.VK_EURO_SIGN:
            System.out.println "You pressed the € key" ;
            break;
        }

    }

    /**
     * This method is called when a key has been released (i.e. no more pressed)
     */
    @Override
    public void keyReleased KeyEvent arg0) {

    }

    /**
     * This method is called when a key has been pressed and released (complete
     * cycle)
     */
    @Override
    public void keyTyped KeyEvent arg0) {

    }

}

```