```java
package maze;

import java.awt.Point;

/**
 * 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, 5);

        // 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
     */
    protected 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);
    }

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

    /**
     * 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();
    }

    /**
     * Gives us the location of player inside the maze
     *
```

```
101      * @param p The player we want
102      * @return The location of the player
103      */
104     private Point findPlayer(Player p) {
105         /**
106          * We go through all the elements
107          */
108         for (int j = 0; j < height; j++) {
109             for (int i = 0; i < width; i++) {
110                 MazeElem e = maze[i][j];
111 
112                 // Once found, get it back
113                 if (p == Player.PLAYER1 && e.p1Present)
114                     return new Point(i, j);
115 
116                 if (p == Player.PLAYER2 && e.p2Present)
117                     return new Point(i, j);
118             }
119         }
120 
121         // This means that the player hasn't been found
122         // which can happen for instance in single player games
123         return null;
124     }
125 
126     /**
127      * Check if some player has reached the exit of the maze
128      */
129     public boolean checkWinner() {
130         for (int j = 0; j < height; j++) {
131             for (int i = 0; i < width; i++) {
132                 MazeElem el = maze[i][j];
133 
134                 if (el.isExit && el.p1Present) {
135                     JOptionPane.showMessageDialog(
136                             null, "You won !", "We have a winner !", JOptionPane.INFORMATION_MESSAGE);
137                     return true;
138                 }
139             }
140         }
141         return false;
142     }
143 
144     /**
145      * Method used to move a player inside the maze
146      *
147      * @param d Which direction do you want to go to ?
148      */
149     public void movePlayer(Direction d) {
150         boolean movementDone = false;
151 
152         for (int j = 0; j < height; j++){
153             for (int i = 0; i < width; i++) {
154 
155                 MazeElem e = maze[i][j];
156 
157                 if (e.p1Present && player == Player.PLAYER1) {
158                     movementDone = true;
159 
160                     switch (d) {
161                     case UP:
162                         if (!e.wallNorth) {
163                             e.p1Present = false;
164                             maze[i][j - 1].p1Present = true;
165                         }
166                         break;
167 
168                     case DOWN:
169                         if (!e.wallSouth) {
170                             e.p1Present = false;
171                             maze[i][j + 1].p1Present = true;
172                         }
173                         break;
174 
175                     case RIGHT:
176                         if (!e.wallEast) {
177                             e.p1Present = false;
178                             maze[i + 1][j].p1Present = true;
179                         }
180                         break;
181 
182                     case LEFT:
183                         if (!e.wallWest) {
184                             e.p1Present = false;
185                             maze[i - 1][j].p1Present = true;
186                         }
```

```
187                         break;
188                     }
189
190                 }
191
192                 if(movementDone) break;
193             }
194             if(movementDone) break;
195         }
196
197         /**
198          * When the move has been done, see if there is a winner
199          */
200         if(checkWinner())
201             generateNewMaze(new Random().nextInt());
202     }
203
204     /**
205      * Only for fun, generate hundreds of labyrinths per second
206      */
207     public void multiShuffle()
208     {
209         Timer timer = new Timer(100, new ActionListener() {
210             public void actionPerformed(ActionEvent e) {
211                 Random rnd = new Random();
212                 mc = new MazeContainer(15, 15, rnd.nextInt());
213                 gd.setNewMaze(mc);
214             }
215         });
216
217         timer.setRepeats(true);
218         timer.start();
219     }
220
221     public static void main(String[] args) {
222         MazeContainer mc = new MazeContainer(100, 50);
223         MazeGame mg = new MazeGame(mc);
224
225 //      mg.multiShuffle();
226
227         // TODO Students should implement next line
228         mg.movePlayer(Direction.DOWN);
229
230         // This shows a nice message window
231         // JOptionPane.showMessageDialog(null, "Title of the window", "Text of the window !",
    JOptionPane.INFORMATION_MESSAGE);
232     }
233 }
234
```