```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);
        }

        /**
         * Sets the text that is displayed at the bottom of the screen
         *
         * @param msg
         */
        public void setMessage(String message) {
            msg = message;
        }

        /**
         * Does the rendering process for the maze
         */
        @Override
        public void render(Graphics2D g) {

            /**
             * Take the borders into account if we are rendering with Swing
             * decoration
             */
            int border_top = this.mainFrame.getInsets().top;
            int border_left = this.mainFrame.getInsets().left;

            int xs = border_left + strokeSize / 2, ys = border_top + strokeSize / 2;

            // Set the pen size using the stroke
            g.setStroke(new BasicStroke(strokeSize));

            /**
             * Grid drawing
             */
            if (drawGrid) {
                g.setColor(new Color(220, 220, 220));

                // Horizontal grid lines
                for (int i = 0; i < nCellsY + 1; i++) {
                    g.drawLine(0, ys, frameWidth - strokeSize + border_top, ys);
                    ys += hCell + strokeSize;
                }


                // Vertical grid lines
                for (int i = 0; i < nCellsX + 1; i++) {
                    g.drawLine(xs, 0, xs, frameHeight - strokeSize + border_top);
                    xs += wCell + strokeSize;
                }
            }


            /**
             * Draw the content of the maze
             */
            g.setColor(Color.BLACK);
            xs = border_left + strokeSize / 2;
            ys = border_top + strokeSize / 2;

            // Draw the solution if required
            if (solution != null) {
                for (int i = 0; i < nCellsX; i++) {
                    for (int j = 0; j < nCellsY; j++) {
                        MazeElem e = mazeContainer.maze[i][j];

                        // Draw solution
                        if (solution != null && solution[i][j] == 1) {
                            g.setColor(new Color(200, 200, 250));
                            g.fillRect(xs, ys, wCell + strokeSize, hCell + strokeSize);
                            g.setColor(Color.black);
                        }
                        ys += hCell + strokeSize;
                    }

                    ys = border_top + strokeSize / 2;
                    xs += wCell + strokeSize;
                }
            }

            xs = border_left + strokeSize / 2;
            ys = border_top + strokeSize / 2;

            // Draw the content of the frames
            for (int i = 0; i < nCellsX; i++) {
                // draw the north edge
                for (int j = 0; j < nCellsY; j++) {
                    MazeElem e = mazeContainer.maze[i][j];

                    // Draw exit
                    if (e.isExit) {
                        g.setColor(new Color(100, 100, 200));
                        g.fillRect(
                                xs + (int) Math.round(strokeSize / 2.0), ys + (int) Math.round(strokeSize / 2.0),
                                wCell, hCell);
                        g.setColor(Color.black);
                    }

                    // Draw position for player 1
                    if (e.p1Present) {
                        g.setColor(Color.red);
                        g.fillOval(
                                xs + (int) Math.round(strokeSize / 2.0), ys + (int) Math.round(strokeSize / 2.0),
                                wCell, hCell);
                        g.setColor(Color.black);
                        g.setStroke(new BasicStroke(1.0f));
                        g.drawOval(
                                xs + (int) Math.round(strokeSize / 2.0), ys + (int) Math.round(strokeSize / 2.0),
                                wCell, hCell);
                        g.setStroke(new BasicStroke(strokeSize));
                    }

                    if (e.p2Present) {
                        g.setColor(Color.yellow);
                        g.fillOval(
                                xs + (int) Math.round(strokeSize / 2.0), ys + (int) Math.round(strokeSize / 2.0),
                                wCell, hCell);
                        g.setColor(Color.black);
                        g.setStroke(new BasicStroke(1.0f));
                        g.drawOval(
                                xs + (int) Math.round(strokeSize / 2.0), ys + (int) Math.round(strokeSize / 2.0),
                                wCell, hCell);
                        g.setStroke(new BasicStroke(strokeSize));
                    }

                    // Is there a north wall ?
                    if (e.wallNorth) {
                        g.drawLine(xs, ys, xs + wCell + strokeSize, ys);
                    }

                    // Is there a left wall ?
                    if (e.wallWest) {
                        g.drawLine(xs, ys, xs, ys + hCell + strokeSize);
                    }

                    // Draw bottom for the last line
                    if ((j == nCellsY - 1) && (e.wallSouth)) {
                        g.drawLine(xs, ys + hCell + strokeSize, xs + wCell + strokeSize, ys + hCell + strokeSize);
                    }

                    // Draw right for the last column
                    if ((i == nCellsX - 1) && (e.wallEast)) {
                        g.drawLine(xs + wCell + strokeSize, ys, xs + wCell + strokeSize, ys + hCell + strokeSize);
                    }

                    ys += hCell + strokeSize;
                }

                ys = border_top + strokeSize / 2;
                xs += wCell + strokeSize;
            }

            /**
             * Draw HES-SO logo, centered, at the bottom of the screen
             */
            g.drawImage(mBitmap, fWidth / 2 - mBitmap.getWidth() / 2, fHeight - 30, null);

            // Write some information message
            if (msg != null)
                g.drawString(msg, 5, fHeight - 40);
        }
    }

    /**
     * This method is used to overlay a solution that has been found using one
     * solver algorithm such as the one implemented in {@link AStar_BEGIN}
     *
     * @param solution The solution to overlay
     */
    public void setSolution(int[][] solution) {
        assert (solution.length == nCellsX);
        assert (solution[0].length == nCellsY);
        this.solution = solution;
    }

    /**
     * Call this method to remove the solution overlay
     */
    public void clearSolution() {
        this.solution = null;
    }

    /**
     * Loads an image into mBitmap
     *
     * @param imageName The image path to be loaded (relative to the src/bin
     *            folder), i.e. /images/...)
     */
    private void loadImage(String imageName) {
        try {
            mBitmap = ImageIO.read(SimpleGraphicsBitmap.class.getResource(imageName));

        } catch (Exception e) {
            System.out.println("Could not find image " + imageName + ", exiting !");
            e.printStackTrace();
            System.exit(-1);
        }
    }

    /**
     * @see GraphicDisplay
     */
    public GraphicDisplay(MazeContainer mc, int sizeOfSquare) {
        this(mc, sizeOfSquare, true);
    }

    /**
     * Display a window showing a {@link MazeContainer}
     *
     * @param mc The maze to show
     * @param sizeOfSquare The width of each square to show
     * @param decorations If we need the borders or not
     */
    public GraphicDisplay(MazeContainer mc, int sizeOfSquare, boolean decorations) {
        mazeContainer = mc;

        nCellsX = mc.nCellsX;
        nCellsY = mc.nCellsY;

        /**
         * Compute the sizes for the graphical display
         */
        wCell = sizeOfSquare;
        hCell = sizeOfSquare;

        /**
         * Size of the frame should have space for all the cells (nCellsX *
         * wCell) and also space for the grid (hence the nCellsX + 1 *
         * strokeWidth)
         */
        frameWidth = (nCellsX * wCell + ((nCellsX + 1) * strokeSize));
        frameHeight = (nCellsY * hCell + ((nCellsY + 1) * strokeSize));

        // Load the image
        loadImage("/images/logo_hei.png");

        // Create a display and keep some space for the picture and the text at
        // the bottom
        disp = new Display("Maze - Minilabor", frameWidth, frameHeight + 55, decorations);

        // Sets the default message
        disp.setMessage("Welcome to the Maze game !");
    }

    public static void main(String args[]) {
        // Generate a maze
        MazeContainer mc = new MazeContainer(20, 15);

        // Display the maze
        GraphicDisplay gd = new GraphicDisplay(mc, 15, true);
    }
}
```