Create a `Position` class that stores two integers reflecting the x and y coordinates of a player position in a maze. It should take a constructor accepting a pair of integers as coordinates as well as an instance of Position (this is used where you need to make a copy). You need to implement both `equals` and `hashCode` methods of `Position`.

[2]

The `Position` class should have method `move(DIR dir)` implemented which modifies the coordinates stored in it in relation to the request to move in a specific direction.

[2]

The maze has to be loaded from a .csv file. Two sample files are provided; keep in mind that there could be many other mazes used. When loading, it is suggested that you load the text file line-by-line and process cells using the `split` method of the `String` which takes a separator expression and splits a string into an array of strings between separators. Method `valueOf` of an enumeration can be helpful to turn strings into elements of `CELL`. Feel free to load in any other way you find useful but you should not use external libraries other than the Sheffield package.

[8]

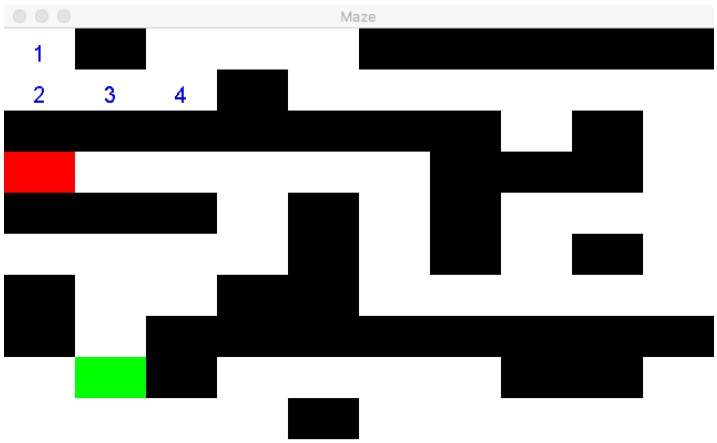
You have to visualise the loaded maze in a similar way to the figure above. Empty cell is blank, wall is solid black, teleporter red and landing cell green. Drawing a filled rectangle can be done by setting a background colour and using `clearRect`. Solutions that draw cells of a fixed size rather than scale depending on the size of the main frame will receive half the marks for this and the next part.

[8]

Given any path, such as the one generated by the expression below, visualise it:

```
Arrays.asList(new Position[]{new Position(0,0),
new Position(0,1),new Position(1,1),new Position(2,1)})
```

A drawing for the path above is shown below:



Each step in the path should displayed in blue close to the middle of a respective cell. If you implement computation of the shortest path (the 20% part below) and visualise it, you will get the credit for this part.

[8]

Make it possible to modify the maze by clicking on different cells. This can be done by implementing a `MouseListener` interface. The `mouseClicked(MouseEvent e)` method can use `e.getPoint()` in order to find coordinates of a mouse inside the component listening to mouse events. You can then find the relevant cell. A shift-click (`e.isShiftDown()`) should move the teleporter (`CELL_T`), control-click (`e.isControlDown()`) should move teleporter land cell (`CELL_L`) and otherwise each cell should be flipped between empty and wall. Your editor should not permit a teleporter or a land cell to completely disappear during edits (such by placing one over another one).

[8]

Draw a UML class diagram depicting relations between around 6 main classes of your solution. There should not be too many classes in the diagram and each of them should only include the most important methods, not all of them. Hence if you use a tool (such as Eclipse) to automatically build such a diagram, you'll have to modify it to match what is expected. Only diagrams submitted as Adobe ® .pdf files will be accepted.