Programming Applications (PRA), Small Group Tutorial 3, "Simply a-maze-ing"

Please review the document marked 'What is a Code Dojo?' on KEATS if you were unable to attend your first Small Group Tutorial session, or if you require clarification on the structure of Small Group Tutorials in PRA.

If you have any feedback about the format of your small group tutorial sessions, please email Martin and Steffen. If there are reasons noted in your KIP (if you are unaware of what a KIP is, then this instruction does not apply to you), or hitherto undisclosed reasons, that would make participating in the session in the proposed way difficult or impossible, please also let us know.

During this Code Dojo, you will start to implement a simple maze game, similar to the one shown in Figure 1. It is not expected that you will finish this implementation, but you should be able to make a good start during the Dojo. You should finish this maze game off during your own time for practice.

1 Rules of the Game

In our maze game, the board is split into a grid of *tiles*. Tiles can either be black, which represents an obstacle or a wall of the maze; red, which represents the players current position; green which represents the exit of the maze; or empty, which represents free space in which the player can move. The objective of the game is to move the red tile (the player) to be at the same position as the green tile, thus exiting the maze, while navigating around the obstacles.

A class `PlayerGrid` is provided for you on KEATS. You should extend this class to allow you to create and modify a frame filled with a grid of coloured tiles in your application.

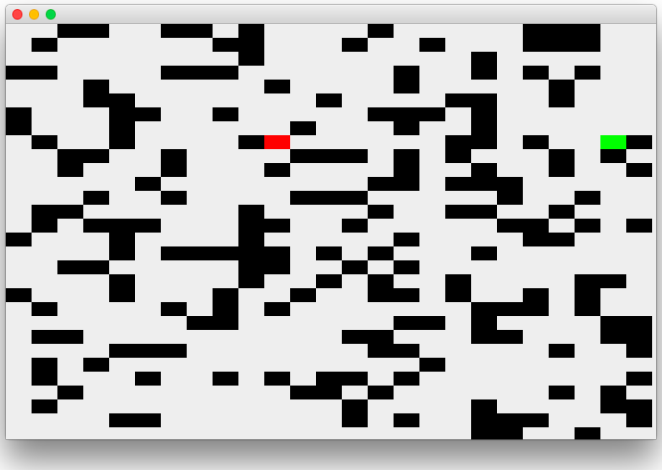

Figure 1: An overview of our maze game

Create an empty grid by invoking the `super` constructor of `PlayerGrid`, with a specified number of rows and columns, and a specified size of the frame.

Your program will then need to exhibit the following functionality:

1. When setting up the game, your program should loop through each tile on the grid, and with a certain probability fill that tile with an obstacle. Note that because this results in the random placement of obstacles, it may be the case that a player can never reach the exit of the maze (i.e. they blocked in). This is not a scenario that we will explicitly account for in our implementation.

2. Once the obstacles have been placed, your program should place the maze exit. It does not matter if this involves changing a tile that previously held an obstacle (i.e. a tile that was previously black) to the maze exit.

3. Once the exit has been placed, your program should place the player at a random position on the grid. Note that a player should not be placed on top of an obstacle or on top of the exit.

4. When the appropriate key – either up, down, left or right – is pressed on the keyboard of the computer running your application, your player should move in the direction denoted by that key. Before moving the player to the next tile in the specified

direction, your program should check if the destination tile contains an obstacle. If it does not, then the player should not move in that direction. Note that you will need to use a `KeyListener` to read keyboard input from the user, and the `keyCode`s 38, 40, 37 and 39 are up, down, left and right, respectively.

5. It should not be possible to move the player off the edge of the window at any point.

Optional:

1. Your program should display a message box congratulating the user once they reach the exit of the maze.