University of Nebraska – Kearney
CSIT 150: Object Oriented Programming
HW4 – GUI, Robot Maze Program Challenge

Robot Maze Program Challenge. See the HW4 program for the multiple choice questions and the other details of this assignment.

`Maze`

The previous project contained a `main()` method which had a "throws" clause that handled bad file names. Now, the main method simply creates the MazeFrame, so you need to handle bad file names in the `Maze` constructor. To do this, place all file-dependent code inside a try-block, and use your catch-block to print an error message and the contents of the Exception object parameter.

`Robot`

Add `void setMaze(Maze inMaze)` method, so that the robot's maze can be changed outside the constructor.

New GUI Object Classes

`MazePanel,` a new `JPanel` class. Once you create this `JPanel`, you can add it to your `MazeFrame` (described below). Inside:

1. Add 2 instance variables, one for a `Maze` and one for a `Robot`.
2. Add 2 "setter" methods for each of the instance variables.
3. Override the **`paintComponent()`** method to draw your game. You'll need to draw a rectangle (with your choice of color) for each location in the maze grid. I would recommend that you make the sides at least 20 pixels in each direction. You also need to add the robot to the display. Think about how to center it inside a rectangle (cell).

*I'm leaving the **`paintComponent()`** method's details a little vague … some of your grade will be determined from the clarity and creativity you use in your drawing. See chapter 14.5 for more information on drawing graphical components. Also, see the two checker games and the bouncing ball examples.*

`MazeFrame,` a new **`JFrame`** named **`MazeFrame`**.

Add instance variables: for a `File`, a `Maze`, a `Robot`, a `MazePanel`, a `Boolean` that will be true when a Robot is traveling through the maze, and other instance variables as needed for the menus and menu items.

Constructor:

1. Each of the instance variables will need to be initialized to appropriate values.
2. Add your **`MazePanel`**.
3. Add a Menu Bar. The Menu Bar will have (at least) 3 choices: File, Maze and Robot.
   Here are the menu items you MUST include (add more if you think they will be useful):
   - File (add Solve and Exit).
   - Revise the Edit menu name to say Maze instead.
   - Maze (add Load File)
   - Robot (add each type of Robot)

   The details of the actions associated with each are detailed later in this document.

4. The menu item choices need to happen in a certain sequence to make sense. For example, we can't "Solve" the maze until both a `Maze file` and `Robot` have been chosen from those menus. How to prevent this: we can "gray out" a menu choice for a `JMenuItem` by using the `setEnabled(false)` method for any of the menu items. Check it out… and think about when to run it, and which menu items should be temporarily unavailable – and also, when they should be available again (this will happen in other methods, of course…).

`public void run() //this can be named anything see the bouncing ball example`

If a robot has already finished the maze, display a `JOptionPane` telling the user to choose a new robot. Otherwise, this method starts the robot through the maze:

1. Set your `boolean` variable to indicate the robot is moving.
2. Call **`Thread.sleep(500)`**. This will delay the running of the program long enough so that you can see the display from its beginning. (The 500 is in milliseconds, so it's not long.) We need this because otherwise the robot moves too fast for the human eye. Remember to handle expectations appropriately.
3. Copy the for loop from HW3 MazeDriver project's **`main()`** method – with changes:
    - After the robot moves, instead of printing, you will need to redraw the visual. Use a special method in `JPanel` called `paintImmediately()`. Go to the documentation and read about `paintImmediately` – you will be calling it with your `MazePanel` object that is placed in the frame, and you'll need to give it the dimensions associated with your `MazePanel` object when you call it.
    - End your loop body with another call to `Thread.sleep(100)`. It will give you time to see the route your robot takes along the way to the exit location.

`public void resetMaze()`

This will be called later if you wish to re-run with a new maze after the robot has finished. This method should not do anything if the robot is currently traveling through the maze or if the maze file has already been selected, It's work will consist of making a new Maze, resetting the maze for the MazePanel (by calling the MazePanel's setMaze method), and calling repaint(). *Repaint will repaint JFrame.* Go to the documentation for more information about repaint.

Menu Item coding

Each menu item will have an `actionEvent` associated with it. It's up to you to decide if you want to make each of these have their own `ActionListener`, or if you use one `ActionListener` that checks the `eventSource`. Here are some notes about each:

`Solve:` (This menu item is not enabled until the file and robot have been selected.) If the robot is running in the maze and the maze has not been solved, ignore this event. If all is good, set your `boolean` to true and call your run() method.

`Exit:` This button only has one line of code for its action: `System.exit(0)`. This means an error-free exit from the program, and will close your maze window.

`Load File:` By choosing this item, your program will bring up a list of files from which you can choose a maze text file. As with other menu items, this one should not do anything if the robot is currently traveling through the maze. Once you have the file, construct the maze with this file, and reset the maze. Once you have the file, the user can then select a robot choice.

At this point, all seems good – except that sometimes the window that appears just isn't quite the right size for the maze that we need to draw (such as with the biggerMaze). To help this, call the `setSize`() method. Make your dimensions big enough so that you have room for all rows and columns of the maze, plus a little room around the edges. I'll let you decide how to figure this out…

Lastly, call `repaint`().

`The Robot menu choices:` These are quite similar to each other. Again, each action should not happen if the game is already running.

1. Construct a new Robot (of the appropriated type).
2. Reset the maze.
3. Associate the robot with your MazePanel (using your setRobot() method).
4. Now the user can choose to solve the maze.

Below are images that show my program with the robot (red dot) in the starting position for the testmaze.txt and biggermaze.txt, respectively.