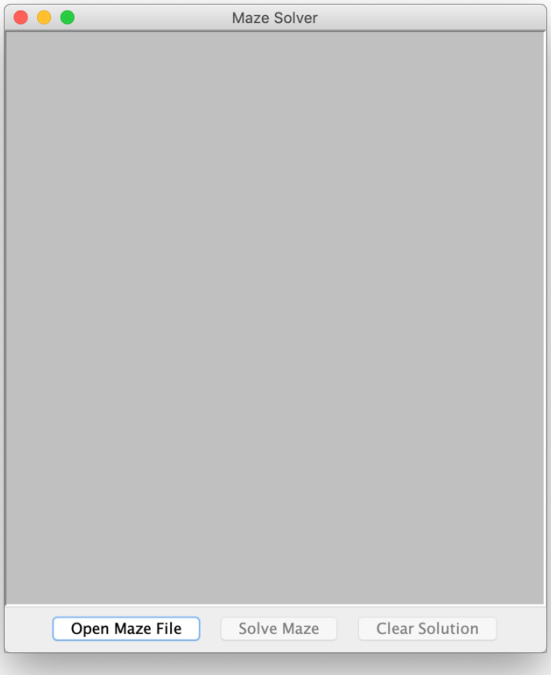
Assignment 3 – 100 points

In this assignment, you will write a Java program to read, draw, and solve simple mazes. The application will consist of a custom panel on which mazes may be drawn and a panel of controls.

Initial screenshot.



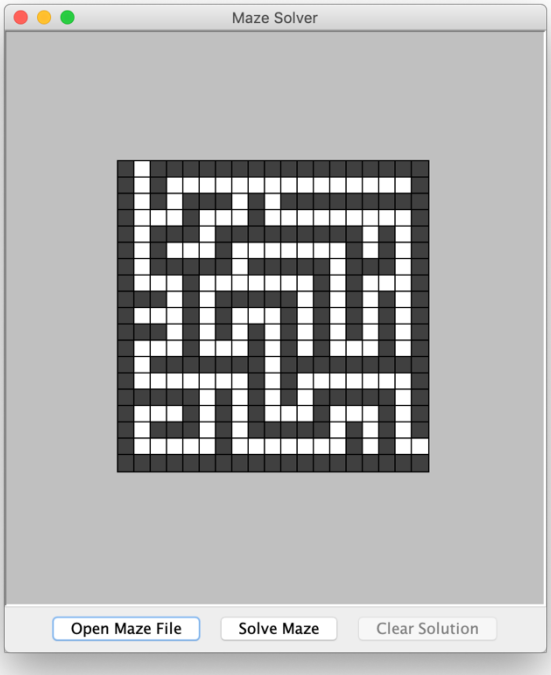
Pressing the “Open Maze File” button should allow the user to select an input file containing maze data. The program should then read the maze data from the file, draw the maze, enable the “Solve Maze” button, and disable the “Clear Solution” button. If the maze data file cannot be opened or is in the wrong format, an appropriate error message dialog should be displayed.

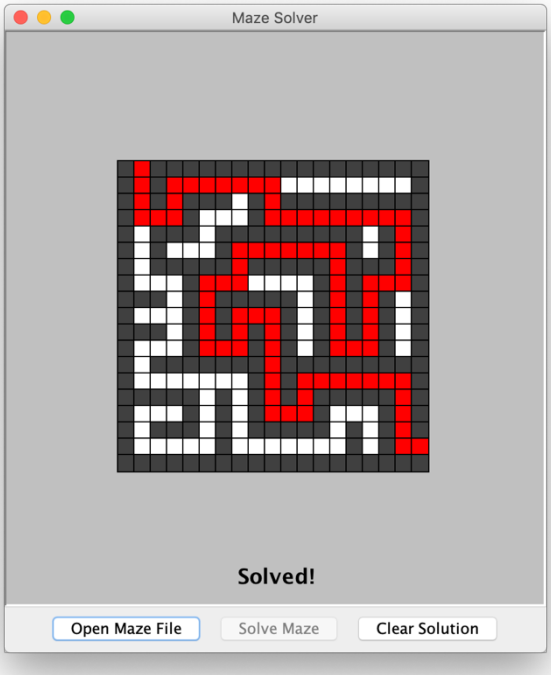
Pressing the “Solve Maze” button should attempt to solve the maze. The maze data will designate one maze square as the start of the maze and another maze square as the end of the maze. A solution to the maze exists if a path can be traced from the start of the maze to the end of the maze. There will be at most one such valid path, but it is possible that a maze may have no solution.

If the maze has a solution, the squares of the path should be redrawn in a different color (such as red) and a “Solved!” message should be displayed. If the maze does not have a solution, a “No solution exists for this maze.” message should be displayed. Solving the maze should also disable the “Solve Maze” button and enable the “Clear Solution” button.

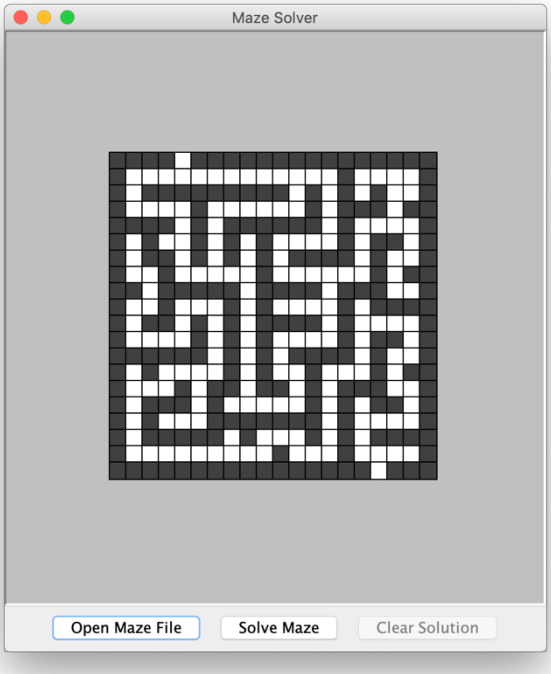
Pressing “Clear Solution” should reset the maze to its unsolved state. It should also enable the “Solve Maze” button and disable the “Clear Solution” button.

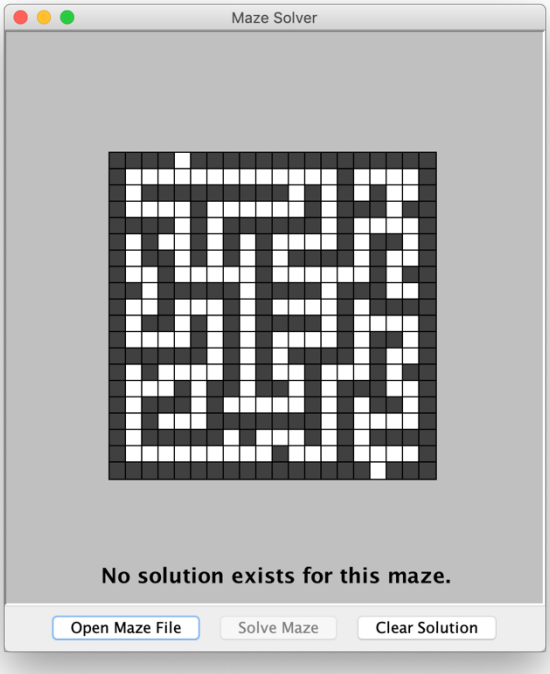
A solvable maze.





An unsolvable maze.





Input File Format

The contents of a sample input file are shown below:

```
19
19
#s################
#.#..............#
#.#.###.#.########
#...#...#........#
#.###.########.#.#
#.#...#.......#.#.#
#.#####.#####.###.#
#...#.......#.#...#
###.#.#####.#.#.#.#
#...#.#...#.#.#.#.#
###.#.#.#.#.#.#.#.#
#...#...#.#.#...#.#
#.#######.########
#.......#.#.......#
#####.#.#.#.#####.#
#...#.#.#...#...#.#
#.###.#.#####.#.#.#
#.....#.......#.#.e
###################
```

The number on the first line of the file specifies the number of rows in the maze data that follows. The number on the second line of the file specifies the number of columns in each row in the maze data that follows.

Those two numbers are then followed by the data for the squares that make up the maze. There are four characters used in the file to represent maze squares:

Character	Meaning
`#`	Square is a wall. Drawn in `DARK_GRAY` in the screenshots above.
`.`	Square is a space. Drawn in `WHITE` in the screenshots above.
`s`	Square is a space that is also the start of the maze.
`e`	Square is a space that is also the end of the maze.

Notes

- See the API documentation for `JFileChooser` and *How to Use File Choosers* in the Java Tutorials for details of how to use one in your program. It's really quite easy.

- Sample maze data files are available for download from Blackboard.

- The layout for this application is straightforward. You do not need to reproduce the sample screenshots as long as your program has similar functionality.

  For what it's worth, I used a `BorderLayout` for the application's content pane, with the `MazePanel` in the `CENTER` region and a `JPanel` with a `FlowLayout` to hold the buttons in the `SOUTH` or `PAGE_END` region. The `MazePanel` has a lowered bevel border. I also drew the squares of the maze so that they overlapped by one pixel so that the lines between the squares ended up one pixel wide rather than two.

- To make grading by the TAs easier, don't put a `package` statement in the `.java` files that you submit.

- Submit your `.java` files using Blackboard's Assignment Manager. If you prefer, you may submit a single zip file containing the files instead.