Laboratory 3: Maze Recursion

Java API

First, we'll show you a demo of what you are trying to accomplish.

Download Lab03.zip from ilearn and extract the contents into your working directory.

Starting Files:

- Drawable.java
- DrawPanel.java
- CenterFrame.java
- MazeGUI.java
- SimpleDialogs.java
- MazeDriver.java
- Maze.java //all of your work is in this file

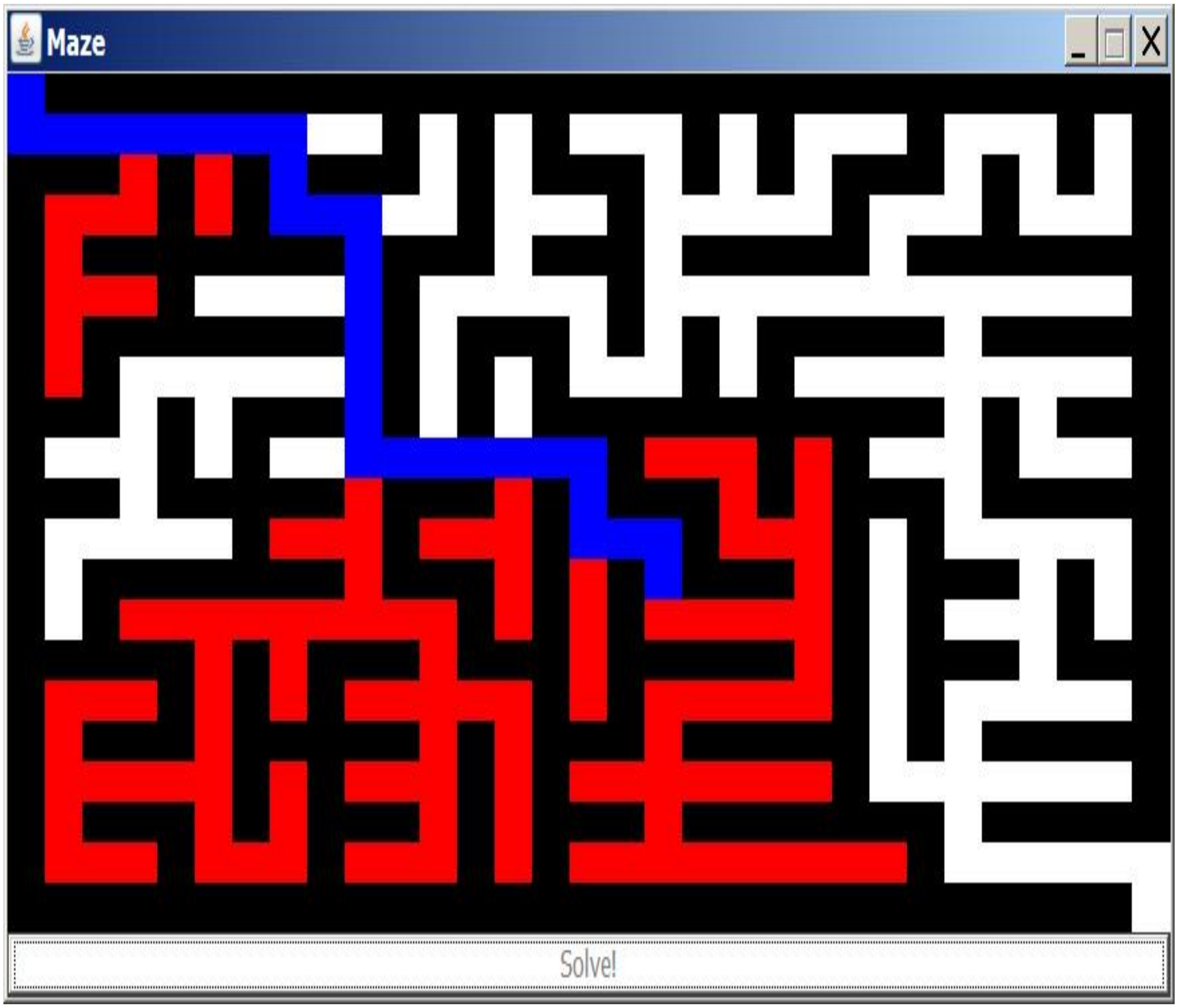
Lab:

- Recursion
- Return Values

Part I: Maze Recursion

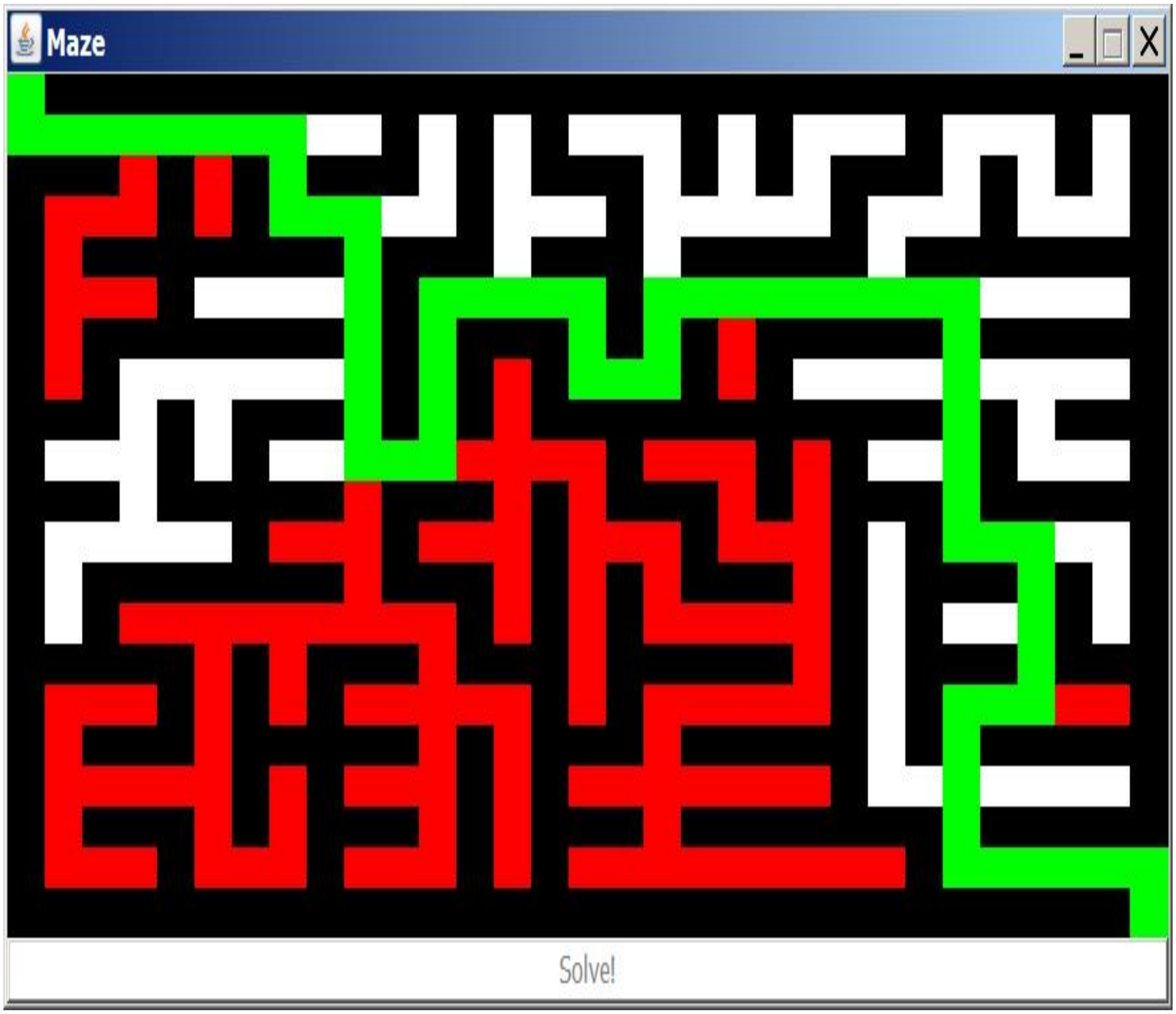
A maze is an excellent application for recursion. From a specific location in the maze (a 2D array of 0s for walls and 1s for paths), make recursive calls for all four possible directions (right, left, up, down) by adding or subtracting to the current row or column. Remember that the call stack maintains the order that methods are called (LIFO), the values of the local variables for each method call, and where the method call occurred in the calling method. When a direction that has not been tried is explored, mark that location as tried by placing the appropriate integer in the current location in the 2D array (blue in the figure). Marking a direction as tried will prevent the algorithm from exploring that location again later. If there is no solution, every location will eventually be marked as tried, and then the user is informed that there is no solution.

A recursive call in a direction already tried or a direction that is a wall can simply terminate without doing anything (removing themselves from the call stack as there is no solution in that direction). When a particular location has had all four directions leading from that location explored (either walls or directions that have been tried), the algorithm must **back track** or move to previous locations that have not had all four directions explored. This is easily done by letting the current location's method terminate and pick up where the previous method call left off. When this occurs, mark the location in red (see figure below). This previous method may have directions that have not been tried, or may now also be completed, in which case the back tracking continues.



The maze with tried locations in blue and back tracking in red.

If there is a solution to the maze, eventually the recursive calls will find the solution. The important idea is that once the solution is found, the methods that are still on the method call stack are those methods that led to the solution. All other method calls corresponding to directions that have been tried and led to a dead end **have completed and were removed from the call stack**. Thus, marking the solution path is trivial. As the methods that led to the solution complete and come off the stack, mark those locations as part of the solution (green in the figure).



The maze with solution in green.

Note that you will need to make use of a **return value** in your recursive calls to the traverse method. Although it is just a boolean, it is an essential component to getting your maze to work. Don't simply make recursive calls, **store** what the recursive calls return to you in a variable to help in decision making. In particular, you need to determine whether the current location led to a solution. If so, mark as path and terminate, returning true. Otherwise, try other directions. If all directions have been tried, mark as back track and return false.

Make sure that you understand this lab. Recursion is important in this class and will be coming up again and again.

Complete Maze.java. Your red region may look different than mine (why?) but your green region should look the same.

To compile: **javac *.java**

To run: **java MazeDriver**

Or you can run the provided batch file: make.bat