CS335 Program 2
"Maze"
Due Friday 11/2

1. Introduction

This project requires you to implement a Java stand-alone application that generates and solves mazes. You will provide visual display and control of the maze parameters via a graphical user interface.

2. Features

The program must run in two primary modes: maze generation and computer-automated maze solving.

1. **Maze Generation**
   Your program must generate a maze using a randomized depth-first search algorithm (there is a pointer to an explanation below). The user must be allowed to visualize the maze construction step based on user-selected parameters (resolution in rows and columns, speed). The result – the randomly generated maze – should be displayed for the computer to solve. It is not acceptable to store the solution path as the maze is generated for playback during the automated solving stage. Your solver must be given no advance knowledge of the maze solution path.

2. **Automated Depth-First Search Solver**
   Your program should solve the maze it generates using a depth-first search algorithm and a stack data structure (note that Java has built-in support for implementing the stack). The solution should be visualized as the solver runs. The speed of the visualization of the solver's path through the maze must be controllable by the user. Your visualization should show cells that are on the active path in one color, backtracked cells (visited but determined to be dead ends) in a second color, and unvisited cells in the default color. In the example below, yellow is the active path, black is unvisited, and grey is backtracked.

3. **The Graphical User Interface**
   On the GUI you must provide the following functionality for the user:

Start/stop	Start (or stop) the automated/user-guided solvers
Speed	Change speed of the visualization of maze generation and solver
Rows/columns	Set maze resolution (10 by 10 to 60 by 60; rows/cols independent)
Generate	Generate a new maze based on current parameters
Percent visited	Display a running percentage of number of cells visited

   The specifics of the design of these elements are your responsibility. The arrangement and usability of these controls will be worth 25% of the grade you receive for your solution.

3. Algorithmic Details

Maze generation: The tutorial at **http://www.algosome.com/articles/maze-generation-depth-first.html** describes the randomized depth-first maze generation algorithm. Use this algorithm to generate mazes given a grid size. Key requirement: always place the start point in the upper left cell, and the end point (exit) in the lower right cell.

Maze solving: Your automated solver should use depth-first search and the right (or left) wall rule. For example, follow the right wall until you hit a dead end (no exit, all adjacent cells visited). Mark cells as "visited" as you move. When a dead-end is reached, *backtrack* to the nearest place where there is another path to take that is available. Follow it until it dead-ends, etc. When the path reaches the end point, detect that the maze has been solved (or that all cells have been visited, and no path was found). This algorithm is not particularly efficient, but is guaranteed to find a solution if one exists.

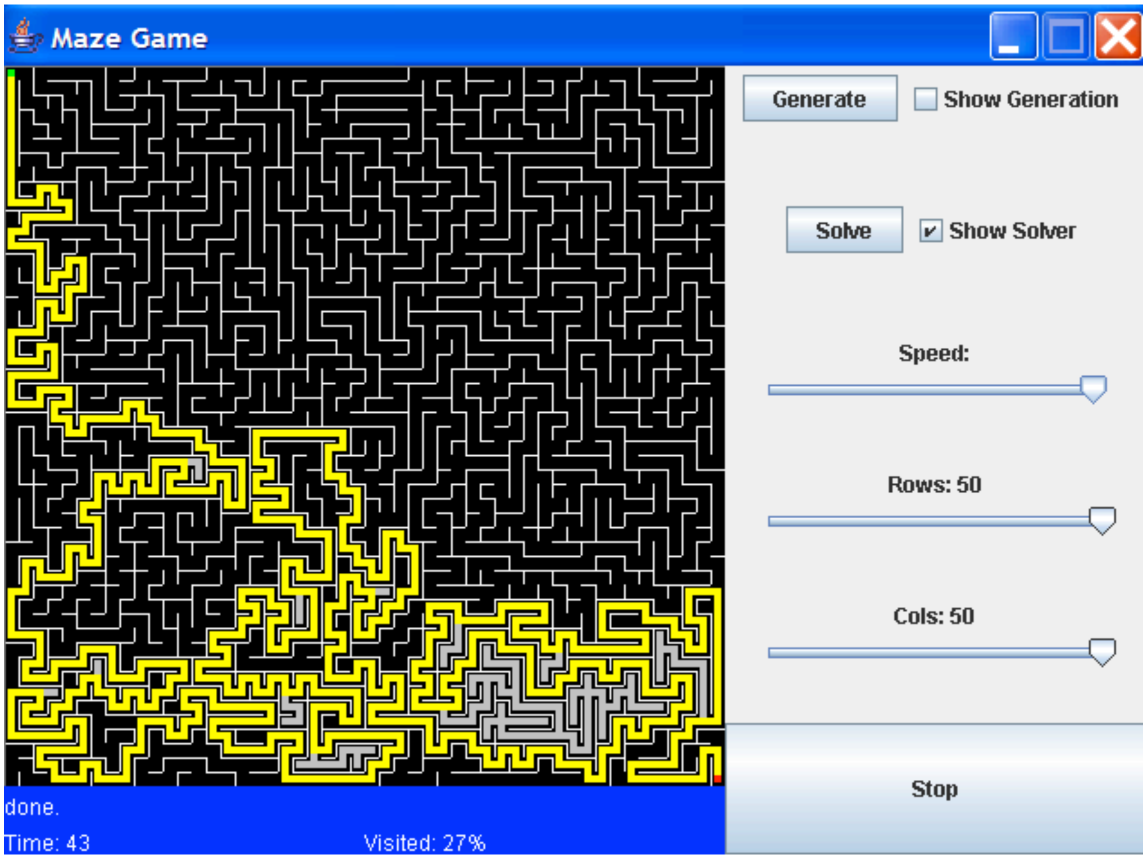
Note that because of the backtracking involved you will need to store path information in a data structure as the maze is traversed. Your visualization of the solver will need all path information in order to display the depth first search with backtracking as it evolves.

4. Example and a Few Hints

You will undoubtedly be able to find examples of solutions for parts of this problem on the Internet. You may review such examples in order to educate yourself, but you must implement a solution yourself. You may not use code you find on the Internet (or code previously submitted in this class) in your solution. Re-write any concepts you see and make your solution your own.
Use a “model-view-controller” design pattern, and consider implementing your design in phases, solving your interface requirements first before tackling the actual maze generation and solver.
As for the visualization of the rendered path, treat this as a computer graphics rendering problem. What are the building blocks for rendering the path as it follows the depth-first-search algorithm? How should each element of the maze “render itself” as part of the path in order to make it appear as if the path is being smoothly traced, erased (back-tracked), etc.? Break things down into simple cases (geometry and attributes) and figure out how to put the cases together into a rendering approach that will look nice.

5. What to Turn In

Submit your IntelliJ project files as a single zipped archive (or tar/gzip) to Canvas. Be sure to name the driver file (with main()) Maze.java. The names of the other files you may submit can be anything. If you include custom icons for this application, be sure to include them in the archive.



done.
Time: 43 Visited: 27%