TCSS 342 - Data Structures
Assignment 5 - Maze Generator

Due Date: Friday, June 5th

Guidelines

This assignment consists of programming work. Solutions should be a complete working Java program including both your work and any files provided by the instructor used in your solution. These files should be compressed in a zip file for submission through the Canvas link.

This assignment is to be completed on your own or in a group of two. If you choose to work in a group of two this must be clear in your submission. Please see the course syllabus or the course instructor for clarification on what is acceptable and unacceptable academic behavior regarding collaboration outside a group of two.

Assignment

Many gaming applications require 2-dimensional n by m mazes with no cycles (the no cycles requirement creates a class of tough mazes with a single unique solution). These mazes are then the target of maze solving games or used by maze solving algorithms. One problem is generating a large library of mazes to be used by the games or to test the algorithms. Of course algorithms exist to generate random mazes. It will be our goal to implement one of these algorithms.

To generate a 2-dimensional n by m maze with no cycles it is helpful to think of a graph with n*m nodes arranged in n rows of m nodes. Each node is then connected to the nodes above, below, to the left and to the right, if they exist, and no other nodes. Let's call this special type of graph G(n,m). A 2-dimensional n by m maze with no cycles is a spanning tree on this graph. To generate a spanning tree of this graph we have several options.

We can use a depth-first search, a breadth-first search, Prim's algorithm, Kruskal's algorithm, or come up with our own method. You can find these methods discussed here. All graph representation and implementation of your maze generation algorithm are left up to the student but you will be expected to generate a maze somewhat uniformly at random. So you cannot just generate one maze over and over or miss out on possible mazes. Keep this in mind when selecting your algorithm. Some are better than others.

In this assignment you will implement a Maze Generator by:

- representing the graph G(n,m)
- using a randomized method to create a spanning tree of G(n,m)
- displaying the final maze to the console with the solution path highlighted with special characters

You will also create a Main class to control the Maze Generator by:

- generating a 5x5 maze with debugging on to show the steps of your algorithm.
- generating a larger maze with debugging off.
- testing and debugging components.
- **(Optional)** create a graphical display for the maze.

Formal Specifications

You are responsible for implementing the Maze class that must function according to the following interface:

- Maze(int width, int depth, boolean debug) - creates a 2d maze of size n by m and with the debug flag set to true will show the steps of maze creation.
- void display() - displays the maze using 'X' and ' ' characters matching the standard presented in the attached trace.txt. Use a special character of your choice to mark the solution path.

The following files are provided for you:

- trace.txt - The console output of my sample solution.

You will submit a .zip file containing:

- Main.java - your test controller.
- Maze.java - the completed and functional Maze Generator.

Grading Rubric

Each of the following will be awarded **one or more** points toward your assignment grade. Not all points need to be achieved to receive a perfect grade. Excess points contribute to the total points gathered for the quarter. Excess points at the end of the quarter will convert into a bonus assignment grade.

Maze

- Selection of maze generation algorithm.
- Selection of maze solving algorithm.
- Selection of a graph representation.
- Proper selection and use of data structures.
- Efficient implementation of maze generation.
- Efficient implementation of maze display.
- Correct maze.
- Correct solution path.
- **(Optional)** Create a graphical display for the maze.

Main

- Test with the debugging flag on.
- Test different sizes of maze.

- Test methods for component testing.

Extra Points

- Compiles first try.
- All and only the files requested are submitted.
- First graded submission. (That is, not a resubmission after a grade awarded.)
- On time.
- No debugging help.

Tips for maximizing your grade:

- Make sure your classes match the interface structure exactly. I will use my own controller (Main.java) and test files on your code and it should work without changes. Do not change the method name (even capitalization), return type, argument number, type, and order. Make sure all parts of the interface are implemented.
- Only zip up the .java files. If you use eclipse these are found in the “src” directory, not the “bin” directory. I can do nothing with the .class files.
- All program components should be in the default package. If you use a different package it increases the difficulty of grading and thus affects your grade.
- Place your name in the comments at the top of every file. If you are a group of two make sure both names appear clearly in these comments.