Programming Assignment #7: Maze Solver

COP 3503, Fall 2013

Due: Wednesday, November 27, 11:59 PM via Webcourses@UCF

Abstract

In this assignment, you will solve the ASCII mazes you generated in Program #6. In the process, you will solidify your understanding of some of the algorithms and problem solving techniques we've covered in class. (I'm leaving it up to you to figure out which ones, though.)

If you use any code that I have given you so far in class, you should probably include a comment to give me credit. The intellectually curious student will, of course, try to write the whole program from scratch.

Deliverables

Maze.java

1. Problem Statement

Find the shortest path through a maze from start ('s') to exit ('e'). The maze will be passed to you as a 2D char array using the format from the previous assignment, and you must mark the solution path with periods ('.'). The mazes I use during testing will be generated by my solution to Program #6. That means each maze is guaranteed to have the following properties:

1. There won't be any cycles in the maze.
2. There won't be any unreachable cells in the maze.
3. You're guaranteed there will be a path from the start position to the exit position.
4. If it exists, the start character ('s') will be in the top-left cell (non-wall) position. (Recall that if there is one cell, it will be marked 'e', and there will not be an 's' anywhere in the field.)
5. The exit character ('e') will be in the bottom-right cell (non-wall) position.
6. The wall surrounding the maze will be unbroken.

You must mark the path your program finds with periods ('.'), but you must avoid overwriting the 's', 'e', and '#' characters. You can only move up, down, left, and right through the maze. No diagonal moves are allowed, and you cannot go through walls. For example:

Input:

```
#############
#s    # # # #
### ### # # #
# #         #
# ### # # # #
#     # # #e#
#############
```

Solution:

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

Input:

```
#############
#s          #
########### #
#          e#
#############
```

Solution:

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

Input:

```
###
#e#
###
```

Solution:

```
###
#e#
###
```

(See included files for more test cases.)

2. Method and Class Requirements

Implement the following methods in a class named `Maze`. Please note that they are all **`static`**. You may implement helper methods as you see fit. Before submitting, please be sure to test your code with the `TestCases.java` file I've included with this write-up. `TestCases.java` will show you exactly how I intend to test your program.

`public static void` `solve(char [][] maze)`

Given a 2D char array that contains a maze, find the shortest path from start to exit and mark it with periods ('.') as described above. **Do not print anything to the screen.** Printing stray characters to the screen (including newline characters) is a leading cause of test case failure.

`public static double` `difficultyRating()`

Return a double on the range 1.0 (ridiculously easy) to 5.0 (insanely difficult).

`public static double` `hoursSpent()`

Return an estimate (greater than zero) of the number of hours you spent on this assignment.

3. Grading Criteria and Miscellaneous Requirements

The *tentative* scoring breakdown (not set in stone) for this programming assignment is:

80%	program passes test within a reasonable amount of time
10%	`difficultyRating()` and `hoursSpent()` return doubles in the specified ranges
10%	adequate comments and whitespace

Programs that do not compile will receive zero credit. Please be sure to submit your `.java` file, not a `.class` file (and certainly not a `.doc` or `.pdf` file). Your best bet is to submit your program in advance of the deadline, then download the source code from Webcourses, re-compile, and re-test your code in order to ensure that you uploaded the correct version of your source code.

NEW! Please remove `main()` before submitting. A lot of `main()` methods are causing compilation issues because they include references to home-brewed classes that are not submitted with the assignment. Please remove.

Your program should not print anything to the screen. Extraneous output is disruptive to the TAs' grading process and will result in severe point deductions. Please do not print to the screen.

Please do not create a java `package`. Articulating a `package` in your source code could be disruptive to the grading process and will result in severe point deductions.

Name your source file, class(es), and method(s) correctly. Minor errors in spelling and/or capitalization could be hugely disruptive to the grading process and may result in severe point deductions. Please double check your work!

Test your code thoroughly. Please be sure to create your own test cases and thoroughly test your code.

Start early! Work hard! Ask questions! Good luck!