CSC 111 – Project 2

Maze Project

**Submission Instructions:** Submit all source code (.java files only, no .class files) **as a single ZIP file** to the appropriate section on Blackboard. For guidance on how to make a zip file, see [these instructions for Windows](https://support.microsoft.com/en-us/windows/zip-and-unzip-files-8d28fa72-f2f9-712f-67df-f80cf89fd4e5) or [these instructions for Macs](https://support.apple.com/guide/mac-help/zip-and-unzip-files-and-folders-on-mac-mchlp2528/mac#:~:text=Compress%20a%20file%20or%20folder,file%20is%20called%20Archive.zip.).

**Goal:** Use recursion to navigate and solve a maze, using backtracking to work your way out of dead ends.

**Project Requirements**

This project is based on Programming Project 5 from Chapter 6 of your textbook (pages 345-347). You will write a program to navigate a rectangular maze from a starting position along the bottom row to an exit along the top row. For this program, you will recursively test potential paths, backtracking if a path leads to a dead end.

You should complete this project in two phases: (1) Read the input file, create the maze, mark the start and exit cells, and display the initial maze. (2) Solve the maze and display the solved maze. These two phases are explained in more detail below.

**Phase 1 – Creating, Marking, and Displaying the Maze**

You will create a **Maze** class to store the rectangular maze and methods to access it. For this problem, the maze is stored as a rectangular **char** array consisting of cells in columns and rows, maze[][]. Cell maze[0][0] is the upper-left corner. Columns increase going across each row designated by the number in the first bracket, and rows increase going down each column designated by the number in the second bracket. So the cell in the fourth column and on the sixth row would be designated maze[3][5]. (Remember that arrays begin at zero, not one!) Each cell is either a wall or is open. One call along the bottom row is marked as the start cell and one cell along the top row is marked as the exit cell.

The maze is stored in a text file formatted as described in the textbook on pages 345-346. (The same maze is given to you as an input file, maze.txt.) Note that the file shows the width (number of columns) followed by the height (number of rows) on the first line. The exit cell and the start cell on the second and third lines, respectively, are shown column first, followed by row. You will not be able to create the maze until you have read the first line of the file to determine its size.

Before attempting to solve the rest of the problem, read the maze from a text file, mark the start and exit cells, and display it. Define constants such as WALL, CLEAR, START, and FINISH to designate the state of each cell.

**Phase 2 – Solving the Maze**

Find a solution by using the strategy suggested in the text. Write methods that update the state of the maze, such as goNorth(), goWest(), etc., attempting to extend the current path another step in the designated direction. A cell along the current path should be marked PATH and cells that have led to a dead end should be marked VISITED. The goNorth() method is detailed on pages 346-347 of the textbook.

Note that your program will be tested on other Maze files with different sizes and different start and finish locations.

**Extra Credit**

Generate a maze randomly instead of reading it from a text file. If you do this, your final code should provide the option to use a file or generate one randomly.

**When your program runs successfully, follow the submission instructions at the top of this file.**

**Some Notes About Code Style:**

Your code should follow typical examples of good coding style, including those principles demonstrated in this class and its prerequisites. Some examples of good style include (but are not limited to):

* Clear, descriptive variable, method, and class names.
  + If names are given as part of the assignment, match those exactly.
  + Use camelCase for variables and methods.
  + Use UpperCamelCase for classes.
* Use of named constants, if appropriate
* Appropriate indentation for blocks of code
* Appropriate spacing between methods and/or blocks of code

Comments should also be used appropriately. A file header is necessary. Additionally, each function and method (except main) should have a brief comment describing its purpose. Explain any unclear variables or blocks of code as well.

**Rubric -- How You Are Graded**

Your grade is based on how well your code performs the assigned task and how well you follow style guidelines, such as readability and commenting.

**Phase 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Full (3)** | **Emerging (2)** | **Minimal (1)** | **None (0)** |
| Maze class | Correctly implemented as char[][] with columns first | Minor design or logic errors | Major design or logic errors | Does not compile |
| Reads maze from file | Correctly implemented | Minor logic errors | Major logic errors | Does not compile |
| Displays initial maze with marked start and finish | Correctly implemented | Minor logic errors | Major logic errors | Does not compile |
| Defines and uses constants (e.g. WALL, START, etc.) correctly | Correctly implemented | Minor logic errors | Major logic errors | Does not compile |

**Phase 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Full (3)** | **Emerging (2)** | **Minimal (1)** | **None (0)** |
| Displays final maze with correct path and visited cells. | Correctly implemented | Minor logic errors | Major logic errors | Does not compile |
| Overall navigation algorithm, including goNorth(), goWest(), etc. | **6 POINTS**  Correctly implemented | **4 POINTS**  Minor logic errors | **2 POINTS**  Major logic errors | **0 POINTS**  Does not compile |
| Marks cells on the PATH | Correctly implemented | Minor logic errors | Major logic errors | Does not compile |
| Marks VISITED nodes | Correctly implemented | Minor logic errors | Major logic errors | Does not compile |

**Readability, style, and comments**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Full (3)** | **Emerging (2)** | **Minimal (1)** | **None (0)** |
| Readability | Legible and follows styling guidelines | Minor issues with styling. | Major issues with styling. | Does not meet styling guidelines at all. |
| Comments | Appropriate usage of comments. | Minor issues with missing or insufficient explanation in comments. | Major issues with comments. | Does not present any comments (or only comment is a header for name, file, etc.) |
| Submission Requirements | Submitted one zip file containing only source code | Some flaws in submission | Submitted source code without placing them in a zip file | No submission |