Maze Walk

Due: Wed. Mar. 19th, @12:00pm

Objective

In this assignment you will be exploring a recursive solution to the maze walk problem.

Maze

You are given a maze as an ASCIIDatafile. This maze has one exit marked with 'E', all walls of the maze are represented as '#'. The size of the maze (rows, col) is given as an integer pair i.e. (8,11). Below is a sample maze.

```
8    11
###########
#         #
#### ######
#    #    E
#    #    #
# ###### ##
#         #
###########
```

Once you run your maze walk application, a path is traced from the starting location (in this case (1,1)) to the exit 'E', using ASCII characters. The user is prompted to enter the starting location at application startup.

```
###########
#>>>v     #
####v######
# v<v#  >>E
#v<^<#  ^.#
#v######^##
#>>>>>>>^ #
###########
```

The Maze Walk Algorithm

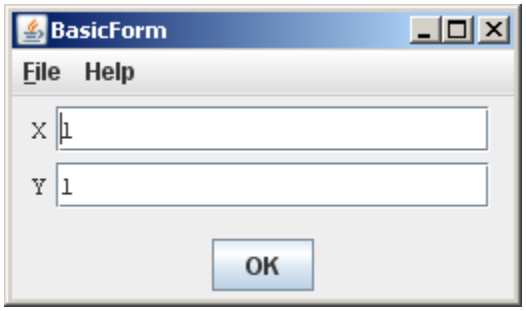
The recursive maze walk is similar to the image scan we discussed in class. Both deal with a boundary which constrains the recursive calls. The premise is to try a direction and if successful keep trying that direction until the exit condition is met or a wall is encountered. This is repeated for all four directions. Note: only 4 directions are considered. If a location has been exhausted, i.e. all four directions, then it is marked with a "." so that no further attempts are made at that location. This prevents infinite recursive calls. Consider the case where a location can be entered from multiple directions. The

solution exemplified above shows one such case, and is dependent on order of the recursive calls.

Write a method findPath(int x, int y) which is recursively called in order to find a path out of the maze. You may modify findPath as appropriate to include additional parameters or a return value to help facilitate the path tracing as in the example.

Little Hints

1) When reading characters from the ASCIIDataFile use the `readC()` method. This method reads exactly one ASCII character.
2) At times you may have extra spaces at the end of some lines of text or simply wish to ignore the rest of the characters on the current line since hidden characters like line feeds are still characters but not part of the matrix. The method `readLine()` will read all characters from the current location to the end of line including the line feed. Allowing the next read to start at the beginning of the next line.
3) Included are 2 maze files mz1.txt and mz2.txt. These can be used as input maze maps.
4) A form to prompt users for the initial starting location could look as follows, where the default starting location is (1,1). Very simple but effective.



5) Use an ASCIIDisplayer to output the maze once a solution has been found. Completed solutions should look similar to the sample given in this text.
6) Depending on the order in which you explore unvisited locations in the map, the final solution may vary slightly indicating a different path or tried locations.
7) The algorithm will find a solution, not necessarily the optimal solution.
8) As with most recursive algorithms the solution will be very few lines of code. If findPath seems to be getting very complicated and big, then chances are you are doing it wrong.

Submission

Refer to the course website for details on submission.
Ensure that you include sample output with *both* maze files.