Project #2
Expanded Maze Solver

Develop the algorithm/pseudocode and the python code required to solve the following problem.

Problem

You are tasked with creating a program capable of solving a maze. In this case your program should be able to solve a maze that is any number of rows in height and any number of columns in width. The maze itself will be brought in as input from a file and should be validated to ensure it contains only correct symbols and that it contains a starting and ending point. Once verified the maze should then be solved iteratively – showing each step.

Directions

Develop a program capable of solving any maze of any size that is brought in via file input. Your maze solver will begin at the tile marked S (start) and attempt to find its way to the tile marked E (end) by making its way through the corridors created by spaces. Your program should mark the path it is taking using directional arrows (caret “^”, greater than “>”, lowercase letter V “v”, and less than “<”) and also mark the paths it has taken but decided were incorrect using periods, “.”.

Requirements

1) Filename input should come from the keyboard.
2) Maze input should come from a user defined file.
3) All output should print to the screen.
4) File input must only contain spaces, pound signs (#), and the letters S and E.
5) All input should be read in string format.
6) Make no assumptions about validity of data.
7) Solver must print out each step of the solution.
8) Solution must make use of functions. A “main()” function along with at least **4** other functions.
9) Solution may not use global variables, functions defined within functions, or any other concept not covered in lecture.

Keep in mind

1) Mazes used as input will follow these rules:
   a. Mazes may split any number of times.
   b. Mazes may contain any number of direct or indirect loops.
   c. Mazes will never contain hallways more than 1 space in width.
   d. Mazes might not have explicitly defined outer walls, meaning the edge of the maze is defined by the edges of the list. In other words, the outer edge of the maze might not have pound signs around it – allowing your solver to walk along the edge of the maze.
   e. The S and E tiles can be located anywhere in the maze, not just at the edges.

f. The S or E tiles might not be present at all.
g. The maze will make use of the following symbols:
   i. Pound signs ‘#’ to signify a wall you cannot pass through.
   ii. Whitespaces “ “ to signify open hallways/rooms you may pass through.
   iii. The letter ‘S’ to signify the start position of the maze.
   iv. The letter ‘E’ to signify the end position of the maze.

2) Your program’s final output should consist of either:
   a. The final maze with the path from S to E clearly marked containing only the following symbols:
      i. Pound signs ‘#’ to signify a wall you cannot pass through.
      ii. Whitespaces ‘ ‘ to signify open hallways/rooms you may pass through.
      iii. The letter ‘S’ to signify the start position of the maze.
      iv. The letter ‘E’ to signify the end position of the maze.
      v. The characters ‘^’, ‘>’, ‘v’, ‘<’ to signify a position where your solver moved up, right, down or left respectively.
      vi. Periods ‘**.**’ to signify positions your solver traversed but determined were not the correct path. These are optional as your solver might solve the maze without taking any wrong turns.
   b. One of the following error messages:
      i. No Start Point
         1. “Error: No start position found.”
      ii. No End Point
         1. “Error: No end position found.”
      iii. Maze Unsolvable (No path between S and E exists)
         1. “Error: No route could be found from start to end. Maze unsolvable.”
      iv. Maze file does not exist.
         1. “Error: Specified file does not exist.”
      v. File is empty
         1. “Error: Specified file contains no maze.”
      vi. File contains an invalid character
         1. “Error: Maze contains invalid characters. Line <line number> contains invalid character ‘<invalid character>’”
            a. Where <line number> is the invalid line number (start counting at 0)
            b. Where <invalid character> is the invalid character that was found surrounded by single quotes.

3) Your solution must solve the maze in incremental steps. For each step, your program must print out the entire maze and mark where your solver moved this turn by placing a symbol into the correct position in the maze. The last step of your solution should show either:
   a. A solved maze with the path clearly marking the route from the S to the E using the directional characters mentioned above (<, ^, >, v). **Do not replace the S or E.**
   b. One of the error messages specified above.

4) Your program must print at least 1 blank line between each iteration of the maze it prints to the screen. This is to make it possible for the auto-grader to find your final maze as well as to increase readability.
5) Your program must make use of functions and thus must contain a main() function and a minimum of 4 additional function. Your solution must also make use of lists to store the maze.
6) Your program may not contain global variables, functions defined inside of other functions, or anything else I have stated in class you may not use - including anything not covered in lecture. If you are unsure about a solution, please ask before submitting the program!

Solution Layout

Your python source code should have the following layout in this order:

1) Your name, the date, and the assignment commented along the top.
2) A comment explaining the problem statement.
3) Comments that clearly state the algorithm/pseudocode you developed and used.
4) Your python source code with comments.

Your code may have the algorithm commented into the code, but also put the entire algorithm in the space explained above so I can see each step of the algorithm before I see any code. Failure to keep a copy of the algorithm/requirements separate of the actual source code will be deducted as if you didn't write the algorithm or requirements at all. **Pay attention to the solution layout!**

What to turn in

A 'zip' file of the name "project_2.zip" that contains only a single correctly formatted python source file named "project_2.py".

Extra Credit information on next page.

Extra Credit

Extra Credit points are only possible if your solver can complete **all** the project 2 mazes correctly. To qualify for extra credit, you can also implement the following features. Each is worth the number of points listed at the end of the feature description.

1) Handle hallways that are wider than a single space. (10 Points)
    a. This feature requires your solver handle mazes that contain hallways that are wider than 1 space as well as large rooms within the maze. Your maze solver must be able to not only search the entire room (the E may be in the center of a room) but it must also be able to enter and exit the various rooms without trouble.
2) Handle mazes that have the S located in the middle of a hallway. (10 Points)
    a. This feature requires your solver handle mazes that have the S located away from a wall or in the middle of a hallways. This means that a 4-way intersection may contain the S in the middle of it, requiring your solver to possibly search in 4 different directions before determining if a maze is unsolvable.
3) Multiple Start Positions (10 Points)
    a. This feature requires your solver to handle mazes that contain more than 1 Start position (S). This means that the input maze may contain multiple Start positions and only a subset (or none) of them are capable of solving the maze. The simplest maze to illustrate this would be:
        i.
        ```
        #####
        #S#S#
        # # #
        # #E#
        #####
        ```
        ii. This creates a maze with two start positions, but only one of them is capable of solving the maze.
    b. Your solve **must** remove all extraneous Start positions (S) from the maze, leaving only the one you used to solve the maze. Thus a solution to the one above would require your maze solve to only leave the S on the right, removing the S on the left as “extraneous”.
4) Handle non-rectangular mazes / Handle ‘jagged’ mazes. (20 Points)
    a. This feature requires your solver handle mazes that contain variable length rows.