CECS 277 – Lab 4 – File IO

Maze Solver

Create a program that allows the user to solve a maze that is read in from a file. The user will begin at the starting point ('s') of the maze and will be able to move up, down, left, or right to move through the maze. When the user reaches the finish ('f'), they have solved the maze.

Create the following functions for your program:

1. `read_maze()` – read in the contents of the file and store the contents in a 2D list. Each character in the maze (make sure to keep all the spaces, but you can strip out the new lines if necessary) should be stored in a separate element in the 2D list (note: this function should work for any size maze, not just the 9x15 provided). Return the filled 2D list.
2. `find_start(maze)` – passes in the filled maze (of any size). Search through the elements in the maze using a set of nested for loops to find an 's'. Return the location as a two item 1D list where the first item is the row, and the second item is the column.
3. `display_maze(maze, loc)` – passes in the filled maze (of any size) and the user's location. Iterate through the contents of the maze. Display each character in the maze in a matrix format. When you reach the user's location, display an 'X' instead so that it shows where the user is in the maze (do not actually place the 'X' in the 2D list, just display it).

In the main function, you should have a loop with a menu that repeatedly prompts the user to move in a direction until the user finds the finish. The user will have a two item 1D list that stores the row and column of their location in the maze that is initialized to the start position of the map. Move the user by updating the row and column values in the location list by adding or subtracting 1 to the row or column (depending on the direction they moved). Do not allow the user to move through any walls ('*'), check the maze at the location the user is moving to see if it is a wall, if it is, display a message that they cannot move there and do not update the user's location. Display the maze each time the user moves. When the user finds the finish, display a congratulatory message and end the program.

Example Output:

```
-Maze Solver-
***************
*      *      *
*** ******* * *
* *         * *
* * **  *** * *
*   *     * * *
* ***** * * * *
*      X*  *f *
***************
1. Go North
2. Go South
3. Go East
4. Go West
```

```
Enter choice: f
Invalid input - should be an
integer.
Enter choice: 2
You cannot move there.
***************
*      *      *
*** ******* * *
* *         * *
* * **  *** * *
*   *     * * *
* ***** * * * *
*      X*  *f *
***************
```

```
1. Go North
2. Go South
3. Go East
4. Go West
Enter choice: 1
***************
*      *      *
*** ******* * *
* *         * *
* * **  *** * *
*   *     * * *
* *****X* * * *
*      s*  *f *
***************
...
***************
*      *      *
*** ******* * *
* *         * *
* * **  *** * *
*   *     * * *
* ***** * * * *
*      s*  *f *
***************
```

```
*   *     * * *
* ***** * * * *
*      s*  *fX*
***************
1. Go North
2. Go South
3. Go East
4. Go West
Enter choice: 4
***************
*      *      *
*** ******* * *
* *         * *
* * **  *** * *
*   *     * * *
* ***** * * * *
*      s*  *X *
***************
Congratulations!  You solved
the maze.
```

Notes:

1. Please place your name, date, and a brief description in a comment block at the top of your program.
2. Use the check_input module provided on Canvas to check the user's input for invalid values. Add the .py file to your project folder to use the functions. You may modify it as needed. Examples using the module is provided in a reference document on Canvas.
3. Do not create any extra functions or add any extra parameters.
4. Please do not create any global variables, instead, pass variables as arguments to the functions and return values back when needed.
5. Please read through the Coding Standards reference document on Canvas for guidelines on how to name your variables and to format your program.
6. Use docstrings to document each of your functions. Document all parameters and return values. Add brief comments in your program to describe sections of code.
7. Thoroughly test your program before submitting:
    a. Make sure that the file is read in correctly and each character is stored in a separate element in the 2D list.
    b. Make sure that you don't mix up the rows and columns of the 2D list.
    c. Make sure that your maze is displayed correctly (ie. properly oriented).
    d. Make sure that the user begins at the start position.
    e. Make sure that all user input is checked for invalid values.
    f. Make sure that the user correctly moves in the direction specified.
    g. Make sure that the user cannot move through walls in any direction.
    h. Make sure that the program ends when the user finds the finish.

Maze Solver Rubric – Time estimate: 4 hours

Maze Solver **10 points**	Correct. 2 points	A minor mistake. 1.5 points	A few mistakes. 1 point	Several mistakes. 0.5 points	No attempt. 0 points
read_maze function: 1. Creates a 2D list. 2. Opens file and reads in contents of the file. 3. Stores file contents into 2D list where each character in the maze is a separate element. 4. Returns filled 2D list. 5. Works for any size maze.					
find_start function: 1. Passes in maze. 2. Uses nested for loops to search 2D list for the 's'. 3. Returns a 1D list with the row and col [r,c] of the start location. 4. Works for any size maze.					
display_maze function: 1. Passes in maze and user loc. 2. Displays maze as a grid. 3. An 'X' is placed at the user's location (instead of the value in the maze). 4. Does not add 'X' to the list. 5. Works for any size maze.					
Main Function: 1. Initialize user's location 1D list [r,c] at start position. 2. Display maze and menu. 3. Get and verify user input. 4. Update user's location based on user's choice. 5. Repeat until finish is found. 6. User cannot move through walls					
Code Formatting: 1. Code is in functions. 2. Correct spacing. 3. Meaningful variable names. 4. No global variables. 5. Correctly documented.					