CECS 277 – Project 2 – Maze Solver –

Space Class – Make a class that represents a single location in the maze. A Space has an x and y location in the maze, a character of what type of space it is, a list of neighboring spaces, variables to track if it has been visited or added before, and a Space for its parent.

Maze Class – Read in the maze from a text file and store it into an array of Spaces. In the file the first line is the size of the maze in rows and columns, and then the maze itself. The maze consists of the following characters: '*' – represents a wall. ' ' – (a space) represents an empty space. 's' – the starting position. 'f' – the ending position.

The Maze class loads the specified maze from the file. Make methods to find the starting Space, to print out the maze, and to take in a particular space and then return an array of its four neighboring spaces (up, right, down, left).

MazeSolver Class– Make an abstract MazeSolover class that has a list that holds the set of Spaces that have been explored so far. Create methods to add and remove Spaces to and from the list, to print out the solution, and an abstract method to solve a maze.

Sub-MazeSolver Classes – Make several different classes that extend from MazeSolver, and create a Maze object in each:

1. Random Search – randomly moves to an open space.
2. Right-Hand Search – follows the right hand wall.
3. Depth-First Search – visits spaces using DFS.
4. Breadth-First Search – visits spaces using BFS.
5. A* Search – heuristic based search.

Basic Algorithm:

1. Add the starting space to the list.
2. If the list is empty, then end algorithm, there is no solution.
3. Remove the space from the list and update it as visited, if it is the finish, end the algorithm and print out the resulting path.
4. Else, find all of the valid neighboring spaces and add them to the list. If necessary, update these spaces as added, and update each of their parents as the removed space.
5. Repeat from 2.

Main – Create several text files that contain mazes so that the user may choose what level maze to solve. I have supplied a few, but you can create your own. Use that file to pass to the Maze class. Present the maze to the user, and then allow them to choose to solve the maze, ask for a hint, or to show the full solution. If they want to solve it, allow them to use the keyboard to move, if they want a hint, move them one step closer to the finish from wherever they are using any of your solving methods, if they chose to show the full solution, have them choose from your solution methods and display the solution path on the maze. Determine which of your methods solves the maze the best.

Notes: Add in constructors for each of the classes, plus any necessary get/set methods. Avoid modifying the structure stated here. Check all user input for validity.