Recursion Program

Create a grid of #’s and dots(.) using a two dimensional array representation of a maze that looks like this:

############

#...#......#

..#.#.####.#

###.#....#.#

#....###.#..

####.#.#.#.#

#..#.#.#.#.#

##.#.#.#.#.#

#........#.#

######.###.#

#......#...#

#’s represent the walls of the maze, and the dots represent locations in the possible paths through the maze. A move can be made only to a location in the array that contains a dot.

You can use any maze formation you wish, but there must be a starting point and exit, with a path to each with some dead paths along the way.

Write a program that contains a recursive method (mazeTraversal) to walk through the maze. The method should receive as arguments a 12 by 12 character array representing the maze and the current location in the maze. (The first time this method is called, it should receive the entry point in the maze. In this example, it would be 2,0). As mazeTraversal attempts to locate the exit, it should place the character x in each square in the path, which is used as a breadcrumb to indicate where it has been.

The algorithm: From the current location in the maze, try to move one space in any of the possible directions(down, right, up or left). If it’s possible to move in at least one direction, call mazeTraversal recursively, passing the new spot on the maze as the current spot. If it’s not possible to go in any direction, “back up” to a previous location in the maze and try a new direction for that location (this is an example of recursive backtracking). Program the method to display the maze after each move so the user can watch as the maze is solved. The final output of the maze should display only the path needed to solve the maze---If going in a particular direction results in a dead end, the x’s going in that direction should not be displayed. [Hint: To display only the final path, it may be helpful to mark off spots that result in a dead end with another character.

Grading Criteria:

10 points- Creates a valid 2D array with appropriate path

30 points- Uses recursion to solve the problem

10 points- Correctly displays the correct path.

10 points- Displays the maze after each move, inserting X’s along the path

10 points- Correctly backtracks, removing X’s along the way.

10 points- Javadoc and internal comments

10 points- Correctly finds the exit

10 points- Correctly validates input