CSE1322L Assignment 9

Concept Summary:

In this lab, you'll be helping a little mouse find his cheese in a maze. The maze will be stored in a two dimensional (2D) array. When the program starts the mouse will begin in approximately the middle of the maze. Some random walls will be added, and the cheese will be randomly placed in the maze.

You'll be writing one method (find_path) which will use recursion to help the mouse get from their starting position to their cheese, or perhaps you may have to tell the mouse it's impossible (again the walls are random, it's possible there is a solid wall between the mouse and the cheese).

Setup

You'll begin by grabbing some pre-written code that will generate the maze, and call the recursive method.

Start by joining this replit team:

https://replit.com/teams/join/zswsaouexmnenxlsvoghnunjqqsxxafs-CSE1322LAssignment09

You'll see a base file for Java and C#.

You can choose to code this in repl.it, or in your IDE (intellij or Visual Studio). Regardless, you must submit your code in Gradescope as usual. No submissions in Replit will be accepted.

If you run it without making any changes, it should draw a maze on the screen. The maze is a 15x15 two dimensional array. Each empty cell of the array is drawn like this:

```
-----
|   |
-----
```

The cell with an M in it is the starting place of the mouse (should be approximately in the middle):

```
-----
| M |
-----
```

If a cell has a C in it, that is the cheese that the mouse is looking for (there is only one cheese and it'll be randomly placed in the maze)

If a cell has an X in it, that is a wall, which the mouse cannot walk through.

Understanding the code you were given:

If you look at the code you were given above you'll find a completed class called Maze. Note: You do not need to change anything in the Maze class. It's all correct, but you do need to understand it.

This Maze class has the following things:

- A public attribute called dimension which is set to 15. This is what makes the maze 15x15.
- Attributes called start_x, and start_y. These are the starting position of the Mouse, which is calculated by dividing the dimension in ½.
- A public two dimensional array called theMaze. This is the maze you'll be working with.
- A public constructor which clears the 2D array, then calls initializeMaze().
- A private method initializeMaze() which randomly generates the maze. You can see that it uses a random number generator to decide if there should be a wall in each cell. Then it places the mouse and the cheese.
- An override of toString() or ToString(). This prints out the current state of the maze.

You were also given a main class which has a completed main method. Again, you do not need to change anything in the main method. It:

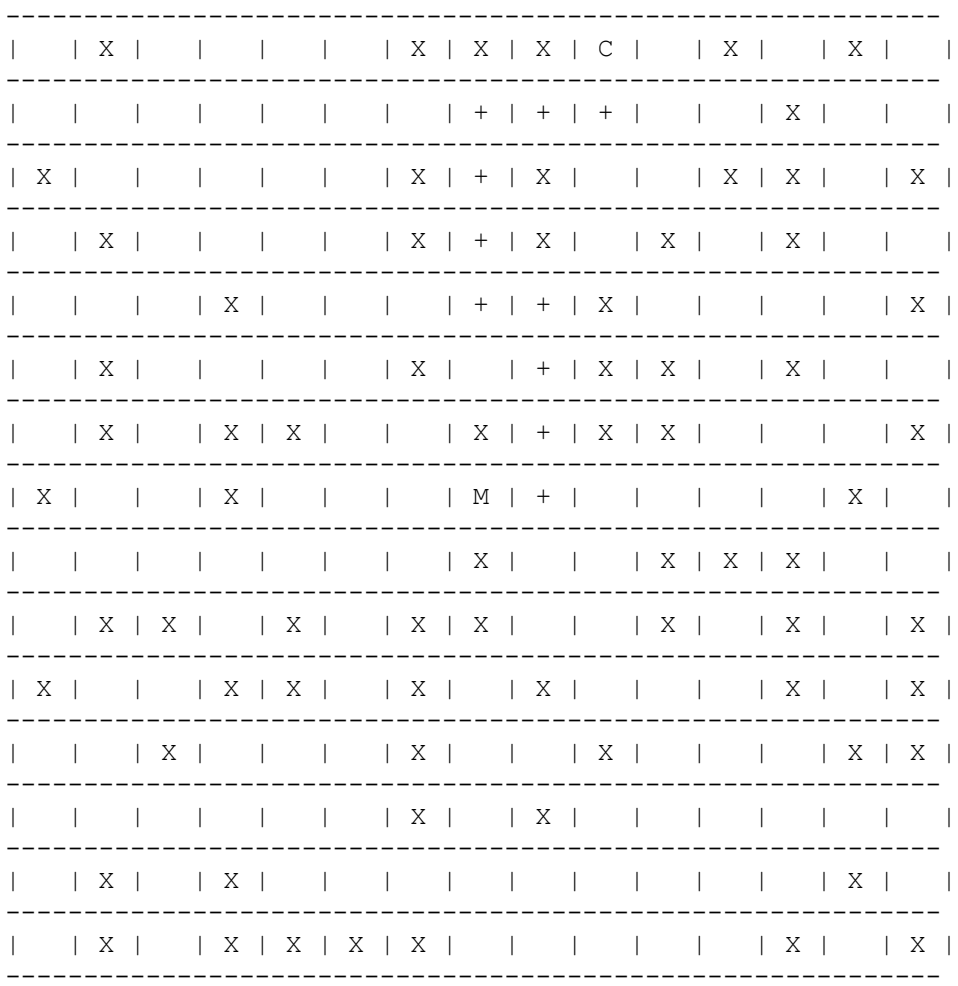
- Instantiates an object of type maze which it calls myMaze.
- Makes a call to a method find_path, passing it myMaze, and the starting position of the mouse
- Receives a boolean back which tells it whether find_path found a path or not.
- Prints Found the Cheese, or No Cheese for Me, depending on how find_path goes.

Your task:

- Look for the find_path method in the Main Class.
- You'll see it takes in a Maze which it calls my_maze, and two integers x and y position. It returns a boolean.
    - You cannot change the find_path method header or parameters.
    - The x and y positions represent the cell you are currently looking at. At the start, this method is called with x=7, y=7. On each subsequent recursion call you'll change one of them to look at other cells.
- The method is full of comments telling you what to do.
    - The first few comments are for you to write the base conditions. As we've said in lecture ALL recursion must have a base condition that stops the recursion. In this case, there are lots of conditions that will stop us:
        - We found the cheese. No reason to keep looking, return true.
        - We try to walk off the side of the maze, that's not valid, you can only look in cells 0-14 in each dimension.
        - We hit a wall, no point in going any further in that direction.
        - We try to go somewhere we've already been (e.g. backtrack on our

current path).
    - If you find the cheese, you'll return true. In the other cases, you'll return false. Note just because the current cell ISN'T a wall, that doesn't mean you've found the cheese. I.e, you should ONLY return true, when you find the cheese, in all other cases you'll just keep looking.
    - Once you've gotten the base conditions written, you'll mark the current path with a + so in the end you can see the path the mouse took.
    - Next you'll make recursive calls to try other paths.
        - The order of the calls is irrelevant, this would work the same if you change the order.
        - You'll recurse up, down, left and right from the current position. Be sure you don't use an if/else if/else if...else, you actually want to go in all 4 directions, not just one.
        - If you are starting off in cell x=7, y=7, and you want to go up a cell, you go to x=6, y=7. If you want to go down a cell you go to x=8, y=7. To go left, you'll go to x=7,y=6, and to go right, you'd go to x=7,y=8.
- After getting the code to work, when you run it, you will see the mouse try to find the cheese.
    - In repl.it, it'll clear the screen between each drawing of the board. That is what these 2 weird lines do in java:
        - `System.out.println("\033[0;0H");`
        - `System.out.flush();`
    - It's likely those lines won't work in intellij, don't worry about clearing the screen, if it works, it works, if not, it's ok. No points are assigned to clearing the screen.
    - Console.Clear() should work for C# folks in both replit, and Visual Studio, however again, no points are assigned to this, it's purely for looks.
- Remember, the board is random every time. So sometimes you run it, it'll not be able to find the cheese, other times it'll instantly find it, and sometimes it'll run for 10 minutes before either finding it or giving up. Don't be alarmed, be sure to run it a few times to ensure it's working correctly, and have patients.

Sample Output (Remember it's random, yours will look different):




```
Found the cheese!
```

Remember:

- This will likely NOT generate the most optimal path, it'll just find a path. It's absolutely fine that it may be the most bizarre path you've ever seen.
- There will be many times it will fail to find the cheese, there may even be sometimes it'll run for 7 minutes, you should be able to see the path as it's running. Besure to run it a few times.

Submission Guidelines:

Submit your final code, which should include the Maze class as well as your Main class.

DO NOT SUBMIT A URL (for replit or other). If you coded it in Repl.it, you must copy/past or download the code and submit the code.

Please follow the posted submission guidelines here:
https://ccse.kennesaw.edu/fye/submissionguidelines.php

Ensure you submit before the deadline listed on the lab schedule for CSE1322L here: https://ccse.kennesaw.edu/fye/courseschedules.php

Rubric:

- Did not change Maze class or Main method in Main class (15 points total):
- Successfully detected if the x and y were off the board and returned false (10 points)
- Successfully detected if they were looking at the cheese and return returned true(10 points)
- Successfully detected if they hit a wall and returned false (10 points)
- Successfully added the + (10 points)
- Successfully made a recursive call up, and returned true if that returned true (10 points)
- Successfully made a recursive call right, and returned true if that returned true (10 points)
- Successfully made a recursive call down, and returned true if that returned true (10 points)
- Successfully made a recursive call left, and returned true if that returned true (10 points)
- Successfully removed the + if this wasn't the valid path (5 points)