**Introduction**

This assignment lets you build a set of classes to support a program to find paths through a cave. By the end of Unit 4, you will have a program that can read a 2-dimensional cave layout from a file, search the layout to find a path to a mirror pool, then print the path to take. This unit, you will build the infrastructure for this project, including  storing data in a two dimensional structure, searching the data structure for a simple path from start to finish and reading data from a text file. *Do* ***not*** *use recursion or backtracking for this part of the assignment.*

**Preparation**

See the announcements for the link to the GitHub repository for this assignment.

**Directions**

Write a class CaveExplorer, that has the following methods:

1. Constructor with no parameters. It should create the two-dimensional structure shown below, where the characters in the layout are ‘R’ for rock, ‘.’ for a clear path, ‘M’ for mirror pool, and ‘S’ for self. This constructor hardcodes the cave without reading from a file.

RRRRRR  
R..SRR  
R.RRRR  
R.MRRR  
RRRRRR

1. toString – no parameters, returns a string (including new lines) showing the current state of the cave exploration. For the initial configuration, this string would be   
   "RRRRRR\nR..SRR\nR.RRRR\nR.MRRR\nRRRRRR\n"
2. solve – no parameters, returns a boolean true if there is a path to the mirror pool, and false if there is not. This method is where you will spend some time figuring out the logic. Even though there is only one path, you still have to search in four different directions and make sure you don't get caught in an infinite loop. How you do this is up to you.
3. getPath – no parameters, returns a String showing the path taken to get to the mirror pool. In the example, this path would be the string of directions “wwsse” for West, West, South, South, East. The method should return the empty string if there is no path.
4. Constructor with one String parameter – the name of a text file with the cave layout. The file has a line with two integers, the number of rows and columns of the cave layout, followed by the layout itself.  For example

5 6  
RRRRRR  
R..SRR  
R.RRRR  
R.MRRR  
RRRRRR

You will have to modify the text file with the cave data using your favorite editor. Your cave layout should contain a path requiring at least 4 moves in two different directions. There must be exactly one path through the cave, that is, from any location, there is only at most one location to move to next that hasn't already been visited. If you need help reading from a file, there are many resources on the web regarding using the Scanner class to read text data. A short one is the  section on Scanners at <https://math.hws.edu/javanotes/c11/s1.html>

1. [Links to an external site.](https://math.hws.edu/javanotes/c11/s1.html)
2. main – test your class by writing a main method that creates at least 2 CaveExplorer objects, solves each one, then prints the starting layout, the final layout, and the path taken, if it exists, for each one. Be sure to follow this order of operations.
3. Answer the following questions in your Word document:
   * Describe the state of your project, what works and what doesn’t.
   * Describe how you tested your program, including tests that made you rethink your code. Include the layout you used.
   * In a sentence or two, what did you learn?
   * In a sentence or two, what did you like about this project?
   * In a sentence or two, what did you find confusing or would like to see done differently regarding this project?
   * In a sentence or two, if you had another hour or two, what would you like to add to the project or how would you do things differently?

**Notes**

After your program is working for each step of the process, you should commit your changes to the repository. This allows you to go back to a previous working version in case you decide to throw out new code and try a different approach. Failure to commit after every step may result in a 0 for the assignment, regardless of the actual code.

Use cave layouts that only have a single path from the starting position to the mirror pool, as in the example, or a single path that doesn't reach a mirror pool. **Don’t include any branching paths in your layouts and don't handle any branches in your code. Doing so will result in a significant loss of points.**

If you run into issues using GitHub, let me know ASAP so we can get them ironed out.

**Submit**

Be sure to put your Word document in your repository and do one final commit. Generate a .zip file from the repository and submit that file to Canvas. Double check that what you submit to Canvas has all your .java files.

**Due Dates**

* by 11:59 p.m., Sunday, CT