COP 3504 – HW 3

Purpose

To examine and practice the use of recursion, and to examine the problem of "search".

Description

Being the nerd that I was growing up, I held quite a fondness for creating mazes. It's always interesting keeping track of the different paths through, making sure that they're interesting and deceptive, and that only one true path would ever remain.

As it turns out, there do exist a few known strategies for algorithmically solving mazes, as well as many other "path search" problems. One of the best known such strategies is known as the **depth-first** search, which operates **recursively**.

Instructions

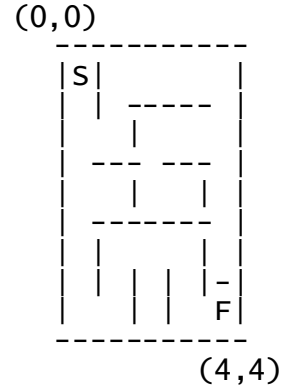
Your mission, for this assignment – implement depth-first search in the context of solving mazes. To this end, I have provided a *.jar file with some example mazes and some interfaces that will be used to evaluate your solutions. Note that the provided mazes are *examples* and will not be the actual test mazes we run your code upon; also note that we are not providing you the source! Each maze will implement the "`Maze`" interface, which in turn has a single method "`getStart()`" which will give you a "`MazeCell`" interface implementation representing the starting cell. (Both interfaces have proper Javadoc-style documentation that you should be able to access from a good IDE.) Your program will effectively be "blind," only able to see from the perspective of a single cell at a time.

`MazeCell` contains the following methods:

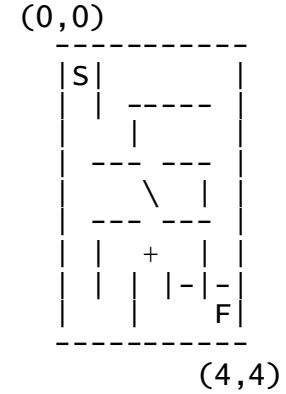
- `north()`
- `east()`
- `south()`
- `west()`
    - Gets the cell corresponding to the corresponding cardinal direction, *if* it is reachable directly.
        - That is, if there's no wall in the way.
    - Returns **`null`** otherwise.
- `isFinish()`
    - Returns **`true`** if this cell is the maze's "finish" cell, else **`false`**.

The provided example mazes are:

ExampleMaze



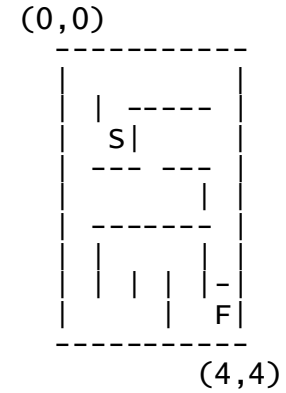
ExampleComplexMaze



The '+' indicates that N/S and E/W are separate paths – N -> E is impossible.

For 10% extra credit, your program should be able to handle additional maze-based challenges – in particular, your program should still be able to finish properly when the maze has *cycles* – that is, your solution *could* be rendered "wandering in circles" if not implemented with this possibility in mind. One such example maze implementation is provided.

ExampleMazeWithCycle



An additional 5% extra credit (additive, not multiplicative) – in order to make a proper "extra credit solution" as mentioned in the last paragraph, there must be an additional, hidden specification on the provided Maze / MazeCell implementations to allow your solution to operate properly. What is this additional specification?

Submission

Your answers for these problems should be submitted online through the Sakai website. Should you choose to include any additional documents, please put them in one of the following formats:

- *.pdf
- *.txt
- *.doc / *.docx