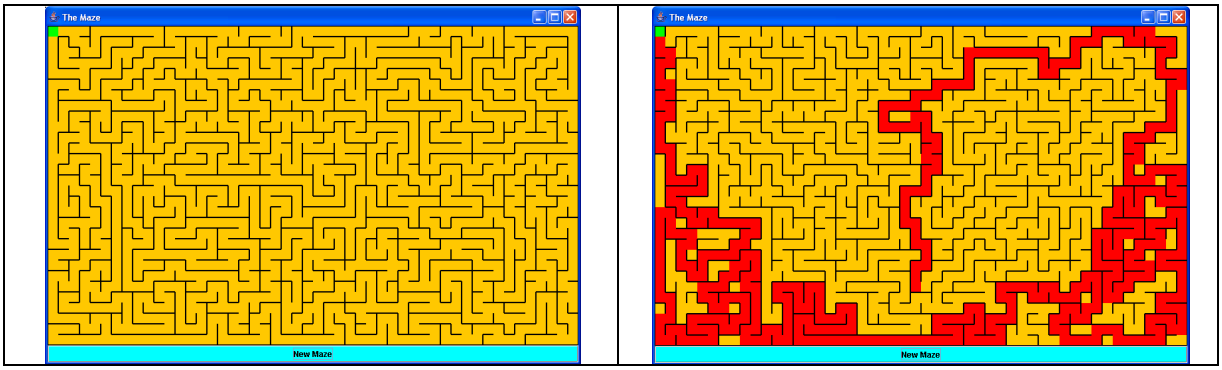
Assignment 5: The Maze

Overview

This time we deal with an everyday problem: you have to find your way out of a maze. Luckily you learned how to think recursively, so the task is relatively simple (a slightly different solution, non recursive, can be found in Greek mythology). This assignment involves relatively little coding, yet more thinking than the previous assignments. The reason is, that you are provided with all the necessary input/output methods to generate, access and display a maze. Your only task is to program a method which provides a path to the maze's exit from any starting point selected by the user.
The program to generate the maze is given to you as a JAVA class. It randomly creates a maze (figure, left). Your task is to find the path to the upper left corner, the green grid (figure, right).



My JAVA class also can display a path in the maze, if provided (that's what you have to do!). You can entirely concentrate on the path generation. Once again, OOP shows its advantages.

How to use the Maze class

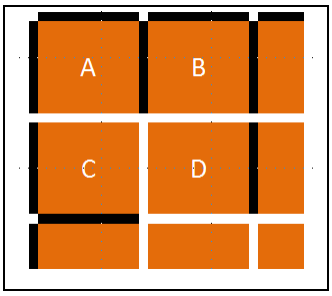
The maze consists of single cells, stored in a 2D array (in the figure above: 60 rows, 80 columns). You don't need to care about the technical details, all you need to know is that each cell can be bounded by a wall in the north, south, east, west. Openings ("non-walls") allow you to pass from one cell to the next.

Important:

- each maze is created with a single, unique way out, which implies that there are no loops in any path.
- There exists a way out to the exit from any starting point.

These two facts makes your programming life way easier (think, why?).

The following figure shows a magnified part of the maze, to illustrate the cells and walls:



- cell A has walls: N, E, W
- cell B has walls: N, E, W
- cell C has walls: W, S
- cell D has walls: E

You can walk *directly* from A to C, yet not from A to B, since there is a wall between A and B (however, there is a path between A and B: A->C->D->B)

You have access to the every single cell of the maze, querying for the wall-configuration:

- boolean hasNorthWall(int row, int column)
- boolean hasSouthWall(int row, int column)
- boolean hasEastWall(int row, int column)
- boolean hasWestWall(int row, int column)

please be aware that these methods take rows and columns, not x and y!

The methods:

Constructor: Maze(int rows, int columns)
Generates a maze of size rows x columns. A typical size would be 60x80. Bigger values tend to be critical due to stack overflow. The constructor also DISPLAYS the maze in its own window. A button is implemented to generate a new maze. A maze has exactly one path out.

public boolean hasNorthWall(int row, int col): returns true if cell(row,col) is bounded by a wall at its north side.**(equivalent: South, East, West)**

public boolean isExit(int row, int col): returns true if the cell row,col is the exit.

public void showPath(LinkedList<Coordinate> path)
used to display the computed path, shown as red rectangles, as in the example figure above. A path is passed to the method by a LinkedList<Coordinate> (the coordinate class is a small class that just has two integer fields, row and column. It is provided.) If you want to display your solution, put all cells that are part of your path as Coordinate objects into a JAVA LinkedList (named e.g. "myPath"), and pass it to this method. The maze will show all participating cells in red.

Remark:
The maze window is clickable, i.e. if you click on a cell in the maze, you can get its coordinates (as row, column, NOT x,y). The framework for this is an "event-based" program. Since this is a bit advanced, I provide the entire framework, you only need to fill in the path-search algorithm into the project.

Task Details

In the netbeans-project "TheMaze", complete the class "PathFinder" to recursively find a path. The project already contains all classes and declarations needed, you just need to work on the method "findPath", which exists as a skeleton in the class PathFinder. Currently "findPath" contains a little example bogus code that shows you how to read a cell, add a cell to a path and check the walls. REPLACE THAT CODE.

More details in class and lab.