Princeton University
Department of Operations Research
and Financial Engineering

ORF 201
Computer Methods in Problem Solving

Lab 7: Maze Solver

Due Sunday, Apr 2, 11:59 pm

---

1. Problem Description

Is there a good way to solve a given (possibly quite complicated) maze? When we solved mazes as kids, we usually used the following procedure:

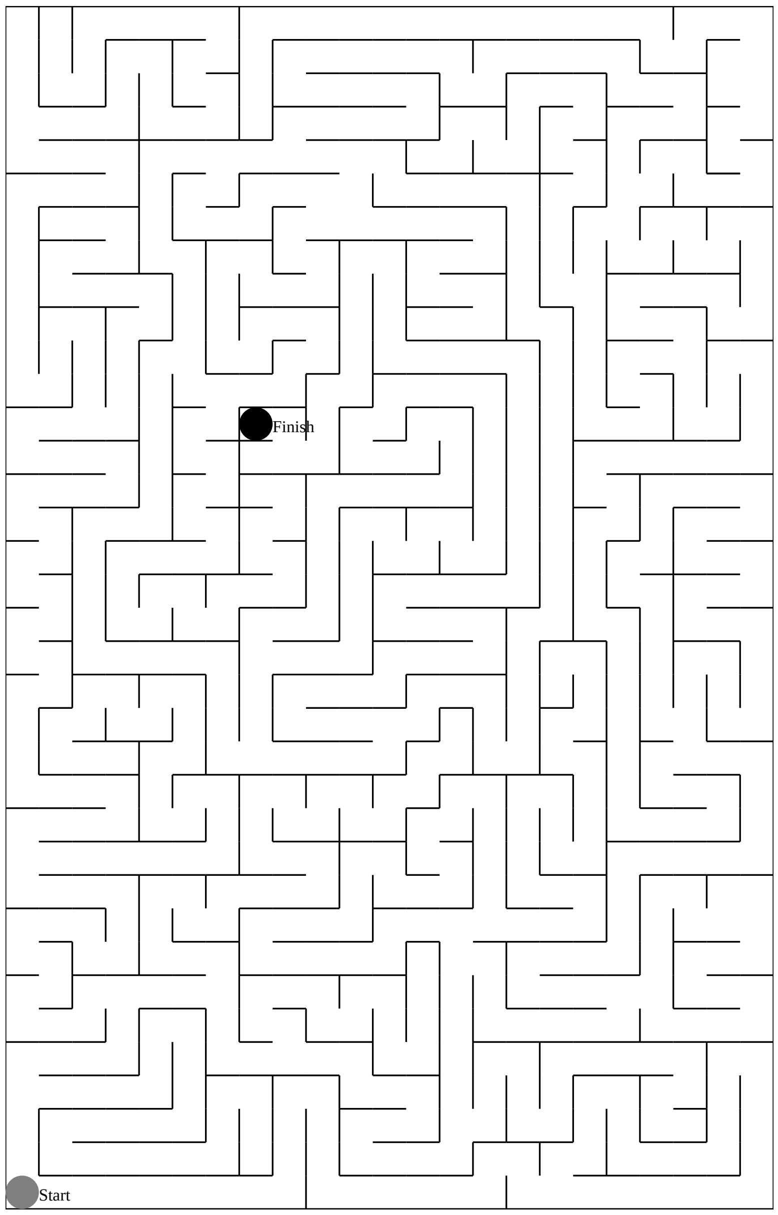
- Using a pencil, we would draw our path down the corridors.
- Whenever we had a choice about which direction to go next, we'd try each possible choice (selecting only those corridors that do not loop back onto a part of the maze we've been over before).
- When we'd reach a dead end, we'd back up along our current path, erasing it as we go, until we'd return to a position from which further choices still remain.
- We would continue doing this until either we solved the maze or we'd exhausted all possible paths without reaching the destination.

This method seems quite reasonable and so we shall program the computer to implement it. To do this, we first need to store the maze in a form that the computer can understand – that is, we shall need an appropriate data structure. Then we shall use the above ideas to design a recursive algorithm to solve the maze.

2. Getting Started

As usual, you need to create some directories and copy some files. Begin like this:

```
cd public_html/JAVA/ORF201
mkdir maze
cd maze
```



Then copy the following files:

```
    cp /u/orf201/public_html/JAVA/ORF201/maze/index.html .
    cp /u/orf201/public_html/JAVA/ORF201/maze/maze.html .
    cp /u/orf201/public_html/JAVA/ORF201/maze/Maze.java .
```

Compile the java code that you just copied over:

```
    javac Maze.java
```

Then use appletviewer to see what the code currently does:

```
    appletviewer maze.html
```

An applet window should pop up with an editable text field for entering the size of the maze, two buttons (generate and solve), and an empty drawing canvas below. At the moment, if you push the `generate` or `solve` button nothing happens. That is because the graphics part and the solver part of the code have been stripped out. It is your job to fill them in according to the instructions given below.

3. Programming Assignment

The class `MazeCanvas` in the file `Maze.java` contains basically three distinct methods:

- `genmaze()`—to generate a random maze.
- `solve()`—to solve the current maze.
- `paint()`—for most of the graphics.

The maze itself is represented by a two-dimensional array `maze[][]`. Each individual element of `maze[][]` is an instance of the class `MazeElem`. This class consists of five booleans, one to indicate the presense/absense of each of the four possible walls and one to indicate whether a crumb has been dropped.

The programming assignment for Lab 7 is do the following:

(1) Write the `paint()` method to draw the maze.
(2) Write a recursively called `solve()` method that will solve the maze (if possible) showing the trial paths as it proceeds, and

Your program should use the graphics window both to show the maze and to animate the recursive solution process.

A sample implementation can be found in `/u/orf201/public_html/JAVA/ORF201/maze/Maze.clas`
To run it as an application, type

```
    cd /u/orf201/public_html/JAVA/ORF201/maze
    java Maze
```

3.1. **Data Structure.**

For each site of the maze, we shall use the following data structure to store the relevant information for that site:

```
    class MazeElem {
        boolean left;
        boolean right;
        boolean top;
        boolean bottom;
        boolean crumb;
    }
```

A value of `true` for `left` means that the current site has a wall on the left, whereas a `false` indicates that there is no wall. A value of `true` for `crumb` indicates that the site has been visited before.

3.2. **Recursion.** Recall that a recursive method is simply one that (under certain conditions) calls itself. Your job is to create a function that will solve a maze from any starting position. The logic and structure of such a function will look like this:

```
    move pencil to current location;
    mark current location as visited;
    if (current location is destination) {
            maze is now solved;
            stop to smell the roses;
    }
    if (can go left and location on left is unvisited) {
            move left and solve maze from there;
    }
    if (can go right and location on right is unvisited) {
            move right and solve maze from there;
    }
    if (can go up and location on top is unvisited) {
            move up and solve maze from there;
    }
    if (can go down and location on bottom is unvisited) {
            move down and solve maze from there;
    }
    erase current location from pencil trace;
```

For the recursive calls, we move from the current location to adjoining squares which are not blocked by walls and then solve the maze from those locations. As mentioned at the start, this is exactly how we solved mazes as kids. If we'd come to a place where could go say left or right, we'd try out both possibilities and see we're the two paths led.

You will need to convert the above "pseudocode" into a legitimate Java program. You will also need to add the graphics portion to animate the solution process.

3.3. **Graphics.** We've grown accustomed to putting all of our graphics calls in the `paint()` method. We required to adhere to this. In fact, graphics calls can appear anywhere in a class that extends `Canvas` (or `Frame`). Of course, a call to `GL.ginit()` still must precede any other call to a `GL` method. The distinguishing role played by `paint()` is that it is the method that gets called whenever the canvas needs to be redrawn for example when brought from the background to the foreground or when changed from iconified to not iconified. For this assignment it is simplest just to put the static drawing in `paint()` and put the dynamic animation of the maze solver directly in `solve()`.

4. Maze.html

Don't forget to edit `maze.html` to make the following changes:

- *Change* `codebase` *from* `"../.."` *to* `http://www.princeton.edu/~yourname/JAVA`.
- *Add the honor code pledge:*
  This program represents my own work in accordance with University regulations.

5. Going Public

If your umask is set for restricted access (i.e., 077), don't forget to make your files public:

```
chmod a+rx public_html/JAVA/ORF201/maze
chmod a+r public_html/JAVA/ORF201/maze/index.html .
chmod a+r public_html/JAVA/ORF201/maze/maze.html .
chmod a+r public_html/JAVA/ORF201/maze/Maze.class .
```

After you've made your files public check to see if you can bring your applet up in a browser. Fire up netscape and go to the following address:

```
http://www.princeton.edu/˜yourname/JAVA/ORF201/maze/maze.html
```

If your code works and permissions are set correctly, you should see the maze applet in the browser.

6. Minimum Requirements

At a minimum:

(1) Your program must correctly draw randomly generated mazes in the drawing canvas.
(2) It must correctly solve the maze using a recursive implementation of `solve()`.
(3) It must animate the solution process by showing those cells that have been visited.