CS 102: Data Structures

Project Four - Maze Master

Student Name	Mohammad El-Abid
Student ID	8905652
Professor	Subrina Thompson
Date	Friday, April 13, 2012

Table of Contents

Abstract

This application generates a random mazes and attempts to solve them. Once it generates a maze that it can solve, it outputs the maze and its solution to the standard output.

Introduction

Before receiving the full instructions I implemented a Maze class that used a stack to process the steps. There was another two-dimensional array that would mark if a position had been explored before. Upon receiving the instructions to use recursion I rebuilt the Maze class, `Maze2`.

Media

Application output to (movie where available) `stdout`:

```
Generating a solvable 50 by 10 maze (500 tiles).
Spawning 5 threads. (Warning: IDEs 'stop' functon will orphan the threads)
Please wait...
Solved by thread #2
Generated 536,446 mazes in 7.67 seconds.

||||||||||||||||||||||||||||||||||||||||||||||||||
SXXXX||XXXXX|||| ||| XXXXXX  |||  ||||   |  ||| ||
||||XXXX| |XX|| |||XXX| |XX|| |   | |||||  |||  ||
||   || ||  XXX|XXXX|||  X|XXXXX|| ||||   |  | | |
|| ||||||   ||XXX||||    X|X|  XX| ||   ||| |  |||
|| |  |     |||  ||  ||||XXX| | XXXXXXXXX |   | ||
|||| ||    |  |  | | |  ||   || ||  ||  X| | | |||
||||   | ||  |      | || | || ||    |  |XXXXXXX| |
| || | |  || |||  | |  | |||  | |||  | |    ||XXXE
||||||||||||||||||||||||||||||||||||||||||||||||||
```

Code

File: Application.java

```
package edu.bridgeport.melabid;

import java.text.DecimalFormat;
import java.util.Scanner;

public class Application {

    public static void main(String[] args) {
           int width = 50;
           int height = 10;
           int threadCount = 5;

           System.out.println("Generating a solvable " + width + " by
" + height + " maze (" + (width * height) + " tiles).");
           System.out.println("Spawning " + threadCount + " threads.
(Warning: IDEs 'stop' functon will orphan the threads)");

           SolveThread[] threads = new SolveThread[threadCount];

           for(int i = 0; threadCount > i; i++) {
                threads[i] = new SolveThread(width, height);
                threads[i].start();
           }

           System.out.println("Please wait...");

           int solved = -1;

           while(solved == -1) {
                for(int i = 0; threadCount > i; i++) {
                     if(threads[i].isFinished()) {
                          solved = i;
                          break;
                     }
                }
           }

           System.out.println("Solved by thread #" + (solved + 1));

           long iterations = 0;

           for(SolveThread thread : threads) {
                thread.stop(); // close threads (use stop instead of
interrupt because it orphans)
```

```
                iterations += thread.getIterations();
            }


            long elapse = threads[solved].getElapse();
            Maze2 maze = threads[solved].getMaze();

            String timeFormatted = (new
DecimalFormat("###,##0.00")).format(elapse/1000.0);
            String mazes = (new
DecimalFormat("###,###")).format(iterations);

            System.out.println("Generated " + mazes + " mazes in " +
timeFormatted + " seconds.");
            System.out.println();

            String mazeSolution = maze.toString();
            mazeSolution =
mazeSolution.replaceAll(String.valueOf(Maze2.CHECKED),
String.valueOf(Maze2.OPEN));
            System.out.println(mazeSolution);

            System.out.println("Display animated solution? (y/n)");
            Scanner in = new Scanner(System.in);
            String res = in.next();

            if(res.equalsIgnoreCase("y") ||
res.equalsIgnoreCase("yes")) {
                maze.mazeTraversal(true);
                System.out.println(maze);
            }
    }
}
```

File: SolveThread.java

```
package edu.bridgeport.melabid;

public class SolveThread extends Thread {
    private Maze2 maze;
    private long elapse;
    private long iterations;
    private boolean finished = false;

    public SolveThread(int width, int height) {
            maze = new Maze2(width, height);
    }

    public void run() {
```

```
            long start = System.currentTimeMillis();
            iterations = maze.generateSolvable();
            long end = System.currentTimeMillis();
            elapse = end-start;
            finished = true;
    }

    public boolean isFinished() {
            return finished;
    }

    public Maze2 getMaze() {
            return maze;
    }

    public long getIterations() {
            return iterations;
    }

    public long getElapse() {
            return elapse;
    }
}
```

File: Maze2.java

```
package edu.bridgeport.melabid;

import java.util.Random;

public class Maze2 {
    private char[][] maze;
    private int row_length, column_length;
    private int start, end;
    public final static char WALL = '|';
    public final static char OPEN = ' ';
    public final static char CHECKED = ',';
    public final static char START = 'S';
    public final static char END = 'E';
    public final static char WALKED = 'X';

    public Maze2() {
            this(10,10);
    }

    public Maze2(int row_length, int column_length) {
            this.row_length = row_length;
            this.column_length = column_length;
```

```
            maze = new char[row_length][column_length];
            randomizeMaze();
      }

      // Testing purposes
      public Maze2(char[][] map) {
            this.maze = map;
            this.row_length = map.length;
            this.column_length = row_length > 0 ? map[0].length : 0;
      }

      /**
       * @return true if the maze can be solved
       * @deprecated
       */
      public boolean solvable() {
            return mazeTraversal(false);
      }

      public boolean mazeTraversal(boolean verbal) {
            // Reset the maze
            for(int i = 0; column_length > i; i++) {
                  for(int j = 0; row_length > j; j++) {
                        if(maze[j][i] == WALKED) {
                              maze[j][i] = OPEN;
                        }
                  }
            }
            // Solve
            boolean result = mazeTraversal(0, start, verbal);
            // side effect is that START becomes CHECKED, so reset it
here
            maze[0][start] = START;
            return result;
      }

      private boolean mazeTraversal(int x, int y, boolean verbal) {

            if(maze[x][y] == END) {
                  return true;
            } else if(maze[x][y] == WALL) {
                  return false;
            } else if(maze[x][y] == CHECKED) {
                  return false;
            // This breaks because we are starting at start, thus
returning false and BANG!
            //} else if(maze[x][y] == START) {
            //    return false;
            } else {
```

```
                maze[x][y] = CHECKED;
                if(verbal) {
                        System.out.println(toString());
                        try {
                                Thread.sleep(500);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }

        // END is to the south-east, so check down and right first

        if(x + 1 < row_length) {
                // right
                if(mazeTraversal( x + 1, y, verbal)) {
                        maze[x][y] = WALKED;
                        return true;
                }
        }

        if(y + 1 < column_length) {
                // down
                if(mazeTraversal(x, y + 1, verbal)) {
                        maze[x][y] = WALKED;
                        return true;
                }
        }

        if(x - 1 >= 0) {
                // left
                if(mazeTraversal(x - 1, y, verbal)) {
                        maze[x][y] = WALKED;
                        return true;
                }
        }

        if(y - 1 >= 0) {
                // up
                if(mazeTraversal(x, y - 1, verbal)) {
                        maze[x][y] = WALKED;
                        return true;
                }
        }

        return false;
}

public void randomizeMaze() {
```

```
            Random rand = new Random();

            for(int i = 0; maze.length > i; i++) {
                  if(i == 0 || i == maze.length - 1) { // left or right
                        for(int j = 0; maze[i].length > j; j++) {
                              maze[i][j] = WALL;
                        }
                  } else {
                        maze[i][0] = WALL; // top
                        for(int j = 1; maze[i].length > j + 1; j++)
maze[i][j] = rand.nextBoolean() ? WALL : OPEN;
                        maze[i][column_length - 1] = WALL; // bottom
                  }
            }

            // pick start point in the first quarter of the maze
            start = rand.nextInt((column_length - 2)/4) + 1;

            // pick an end point in the last quarter of the maze
            end = rand.nextInt((column_length - 2)/4) + 1 +
(column_length * 3/4);

            maze[0][start] = START;
            maze[row_length - 1][end] = END;
      }

      public long generateSolvable() {
            long iterations = 0;
            while(!mazeTraversal(false)) {
                  randomizeMaze();
                  iterations++;
            }
            return iterations;
      }

      public String toString() {
            StringBuilder build = new StringBuilder();

            for(int i = 0; column_length > i; i++) {
                  for(int j = 0; row_length > j; j++) {
                        build.append(maze[j][i]);
                  }
                  build.append("\n");
            }

            return build.toString();
      }
}
```

Conclusion

I learnt how to use Java threads and using recursion as a stack. It was fun to create something that felt advanced, even though the solution is simple, arriving to it was a lot of fun.