1 Project Description

For the project in this class it is your job to write a maze generator and a maze solver. A maze can be viewed as a graph on a 2D grid, where each vertex’s neighbors can be those vertices who are adjacent to it.

The language for this project is Java, so we will give you two templates to get started, MazeGenerator.java and MazeSolver.java. MazeGenerator simply takes in a width and height, and returns a Maze, which is a grid-shaped graph with the locations of two special nodes, the Start and End nodes, given. MazeSolver takes in a Maze and gives a path to the solution. Your MazeSolver will have access to the Maze through a function called Observe\((x,y)\), which will tell you the valid moves from location \((x,y)\) (thus it is feasible to use bidirectional search, since you begin with the start and end nodes already observed). MazeSolver must find a solution, and MazeGenerator must generate mazes that are solvable).

A competition will be held between all teams which pits each team’s Solver against every other team’s Generator. Each match will be a run in which \(K\) mazes are generated by the Generator, and passed to the competing Solver. Expect \(K\) to be around an order of 100 (the exact value of \(K\) will be given to your MazeSolver through the constructor). The performance of your MazeSolver and MazeGenerator will be evaluated based on how many calls to the function Observe are made, minus the number of nodes in the optimal path. Thus, a solver which finds the end node with no missteps will have a net loss of zero, and a maze that requires you to visit every node will have a net gain of zero.

2 Code

The code in the “maze” package is (which you should *not* modify for your submission):

- **Pair.java**: A simple class for storing a pair of integers

- **Vertex.java**: The Vertex class stores the location and neighbor information of a node in the maze. Pay particular attention to the “legalNeighbors” field.
Maze.java: The Maze object, by its default constructor is not modifiable, and whose structure can only be accessed through the “Observe” function. The objects of the type of the nested class, WriteableMaze, however, can be modified through the various provided functions (this is what you should use in your MazeGenerator). The WriteableMaze can give you the standard Maze object by the function “getFixedMaze()”, which would be passed to a MazeSolver for finding a solution. Also, there is a provided “visualize” function which you may find useful for debugging.

We will be using our versions of these classes for evaluating your code, so please only modify them for debugging purposes, and be sure to check that your submitted code works with the original versions before submitting. The code in the toplevel folder is:

- MazeGenerator.java: Template file for your code that generates mazes. Your MazeGenerator should generate mazes that are iid (independent and identically distributed). This means that your generator cannot have memory of how many mazes it has generated, or on what types of mazes it has already generated, or anything of the sort. You are encouraged to generate different types of mazes, but which type is chosen can only depend on your random number generator, and it must be from the same distribution every time.

- MazeSolver.java: Template file for your code that solves mazes. Unlike the MazeGenerator, your MazeSolver is allowed to have memory between matches, so that it may learn better strategies against the types of mazes it is seeing.

- Test.java: Sample file for how a maze would be generated and passed to the MazeSolver, and then visualizing the generated maze. This code won’t work until you fill in the above two files.

- CPair.java: A comparable pair class, which you may use for any PriorityQueue in your algorithms.

In your submission, only include the MazeGenerator.java and MazeSolver.java, along with any other files that you use other than those in the “maze” package, in a single folder. Please do not compress your submission.

3 Grading

Your project will be graded based on completion of the basic task, ranking and cleverness of your algorithms and techniques, and the quality of your writeup:

- (5 points) Create a MazeSolver.java which always solves a maze, if it is solvable.

- (5 points) Create a MazeGenerator.java which always generates solvable mazes.

- (20 points) Make a good, clear writeup that explains your ideas and algorithms, and why you made these choices for your submission. Also explain the things that you tried that didn’t work out or didn’t make it into your final submission (it’s better to
have tried and failed than to have not tried at all). Finally, your ranking will heavily
influence this score as well. You only need to submit one writeup per team, but please
list all members of your team in your writeup, and give a rough indication of the
contributions of each member.

These grading criteria are subject to change.