Homework 4 Writeup

Solving Mazes with A* Pathfinding

The assignment I wanted to tackle was going to be something practical. I have always wanted to write a maze generator and then find some way of solving that maze. It's easily a good example of using pathfinding to solve an intellectual problem in face of intellectual obstacles.

The maze was the first hurdle. I wanted to generate interesting mazes rather than something completely random. There are a few methods to go about this: depth-first, Kruskal's and Prim's algorithms. I chose to use Prim's algorithm to create a minimal spanning tree of nodes, with random passages so as to make mostly one-way streets that are sufficiently 'messy'. The algorithm looks at neighboring walls and removes a random wall only if the floor on the opposite side is not already in the maze.

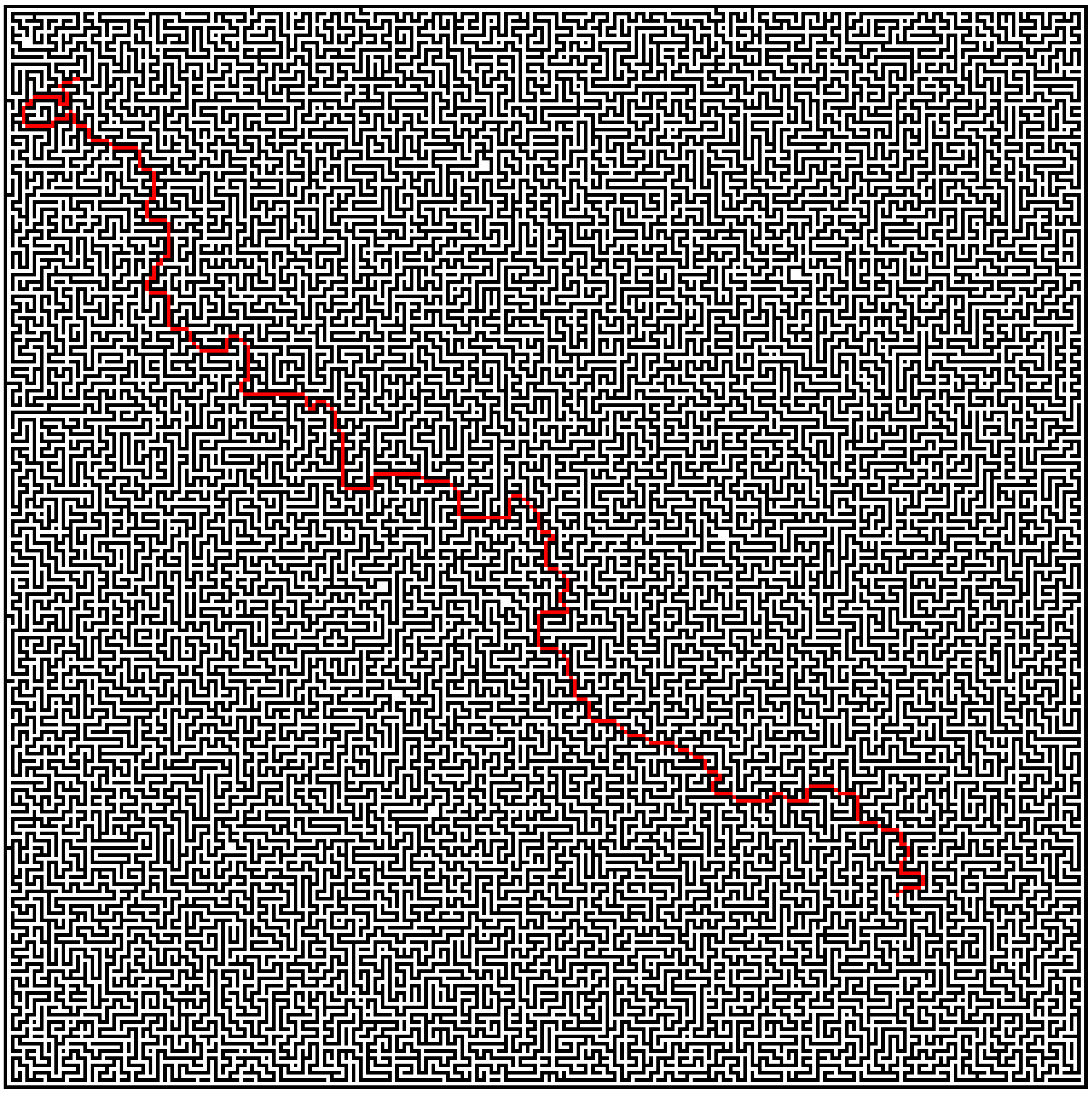
Once the algorithm for the maze was written, I had to "solve" the maze. Solving the maze for me was trying to find a path from a start to an end point. This is not necessarily the far left or far right of the map, but any two random points. Of course it can't be some trivial distance away. I also intend for this path to be the shortest path between those points.

For the pathfinding algorithm I used A*. The goal was to try and in some way find a path that always got you closer to the goal, provided the path travelled thus far is kept shorter than some alternate path. It sounds like depth-first would be a good approach, but it would be too slow. Generating the maze took easily 10 or more times longer than solving it. The A* algorithm keeps track of where you are (the 'current node') and looks at the costs of its neighbors. It chooses the neighbor that is closest to the goal and is going closer. If the path starts going too far it is simply ignored. Only the path that has the shortest route makes it to the end. It is a greedy algorithm.

For the heuristic I used *Manhattan Distance*, what it does is give you the distance two points are away from each other in terms of up and down movement only (like travelling city blocks in Manhattan, where the namesake comes from). That means I can weigh the heuristic so that orthogonal movement is biased compared to diagonal movement. A diagonal is roughly 1.4 times longer than orthogonal movement, but despite that it is most of the time faster to take.

Finally, I wrote a runner that draws the maze and path to screen. I figured a text output would have been useless for this kind of assignment, and so writing the graphics was an extra necessary step. Black squares are blockades and white squares are the open walkways. A red path is the path found via A* pathfinding. Figure 1 shows what a graphical output looks like ran with the parameters 20, 20, 246, and 245 (the two start and end locations respectively).

Figure 1:



To build the code, run "make"

To run the code, run "java Paver x1 y1 x2 y2" (the arguments are your start and end points).