Assignment 9 Analysis

Brayden Wright
u0895942

1. **Who is your programming partner? Which of you submitted the source code of your program?**

My partner for this assignment was Abdulaziz Aljanahi (u0901606). I was the one that submitted the source code.

2. **Evaluate your programming partner?**

Didn't work great this time and both of us had difficulties actually meeting up in person to discuss the assignment and implement ideas. I'm gonna try to find someone else to work with for the last pair programming assignment. Hopefully our schedules will better align.

3. **Does the straight-line distance (the absolute distance, ignoring any walls) from the start point to the goal point affect the running time of your algorithm?**

I would expect that ignoring walls would speed up runtime a decent amount. By how much, I'm not quite sure. Luckily, I get to perform a test in the next question that should provide some insight.

4. **Explain the difference between the straight-line distance and the actual solution path length. Give an example of a situation in which they differ greatly. How do each of them affect the running time of your algorithm? Which one is a more accurate indicator of runtime?**

After performing a quick analysis to determine how our program performs in this regard, we got the opposite answer of what we were expecting. Our actual solution ended up running faster, on average, than the straight-line solution. In any of the tests though, they were within 15 milliseconds of one another, so not a huge difference at this scale. The largest difference occurred in a custom made maze that forces the path to use nearly half of the present points in the maze (really zig-zaggy) with the start and goal points at opposite corners, though our actual solution still ran better than the straight-line solution.

When looking over our code again to determine why we were getting an odd result, we eventually found out what was going on. It turns out we have a somewhat unintentional optimization. When finding neighbors of each node in the graph, we were only adding non-wall points, or valid potential path points, to the list of neighbors. This caused each list of neighbors to be a size *up to* 4. If we instead treat walls as valid neighbors, nearly every list of neighbors (except border nodes) is guaranteed a size of 4. This difference in the amount of neighbors causes the neighbor-traversal part of our pathfinding to generally run quicker when walls are not considered as valid path points. Essentially, the Breadth-First Search runs quicker when passed less nodes to work through, as is what happens with our actual solution.

5. **Assuming that the maze is square, consider the problem size, N to be the length of one side of the maze. What is the worst case performance of your algorithm in Big-O notation? Take into account the density of the maze.**

The running time for Breadth-First Search can be calculated by O(V + E) where V is the number of nodes and E is the number of edges. Since we don't necessarily explore edges so much as iterate over nodes, we only care about the number of nodes. So, the runtime would end up being O(V + 1) or O(V). Since the number of nodes in a square maze with a side of length N is N^2, the worst runtime would be $O(N^2)$. The best runtime would be somewhere between O(N) and $O(N^2)$ since we can not take diagonals, but end up much closer to O(N).

6. **How many hours did you spend on this assignment?**

We spent somewhere around four to seven hours on this assignment.