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

My programming partner is Dylan Northcutt. I submitted our source code.

2. Evaluate your programming partner.

Dylan is an excellent partner. As I stated in the previous assignment, he takes the assignments seriously and knows his stuff (concerning data structures as well as the Java language). Moreover, he also does well at tolerating my various idiosyncrasies.

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?

The straight-line distance has no **direct** impact on the running time of our algorithm. Consider, for instance, the following mazes:

```
XXXXXXXXXXXXXXXX
X   X   X       X
X X X X  X XXX X
X X    X X X   X
X X XXXX   X X X
X S.E  XXXXX XXX
X  X X          X
XXXXXXXXXXXXXXXX
```

```
XXXXXXXXXXXXXXXX
X...X...X .....X
X.X.X.X..X.XXX.X
X.X... X.X.X...X
X.X XXXX...X.X X
X.SXE..XXXXX.XXX
X  X X.......  X
XXXXXXXXXXXXXXXX
```

These mazes are nearly identical; the only difference between them is the addition of one wall. As such, both mazes have the exact same straight-line distance (2). We would expect, though, for our algorithm to run much faster for the maze on the left because the breadth-first search would reach the end node after visiting only a few nodes around the start node.

However, we believe that straight-line distance has an **indirect** relationship with the running time of our algorithm. This makes intuitive sense: a larger maze will, on average, have a larger straight-line distance and will require our algorithm to cross, on average, more edges.

continued…

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 run-time?

The straight-line distance is the distance between the start and end nodes as the crow flies. This is the length of the shortest possible path between them if there were no walls. The straight-line distance depends only the number of rows and columns between the start and end nodes. The actual solution path length, however, is highly sensitive to the density and placement of the walls. The maze to the above right shows a case where these two values diverge widely. The straight-line distance is only 2 but the actual solution path traces a very large portion of the maze. The removal of one wall, a small change, results in both distances being 2, a huge difference, in the above left maze.

As previously established, the straight-line distance has a weak, indirect relationship with running time because of the stronger influence of variables such as wall density and placement. The actual solution path, however, shows us a substantial piece of the work our algorithm has done; the breadth-first search must have touched at least every node in this path. (It does fail to capture, though, the amount of work done exploring side-paths.) **We believe, then, that that the actual solution path length is a better predictor of run-time** because it better captures the massive impact resulting from nuances in wall placement.

5. Assuming that the input maze is square (height and width are the same), 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-Oh notation? Your analysis should take in to account the density of the maze (how many wall segments there are in the field). For example, a completely open field with no walls other than the perimeter is not dense at all, as opposed to the example maze given "bigMaze.txt", which is very dense. There is no one correct answer to this since solutions may vary, but you must provide an analysis that shows you are thinking about the problem in a meaningful way related to your solution.

Our algorithm does not add walls as vertices of the graph. Therefore, a maze is that is dense in walls is not dense in edges and vice versa. This results in the worst possible case being a completely open field (with no walls other than the perimeter), as this adds the greatest number of edges to the graph, and the start and end nodes being in opposite corners. Because we are performing our search in a breath-first manner, our algorithm visits every node spiraling out from the beginning node until the end node is found. The number of vertices (V) in this graph—every possible cell aside from the perimeter—would be (N - 2) * (N - 2) and thus $O(N^2)$. The number of edges (E) would also be $O(N^2)$ because there would be roughly 2 * V edges (each interior node has 4 edges, but we divide by two to avoid double-counting edges). Seeing as the space complexity of breadth-first search is O(V + E), the complexity of our algorithm in this case would be $O(N^2)$.

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

We spent roughly 6 hours on this assignment.