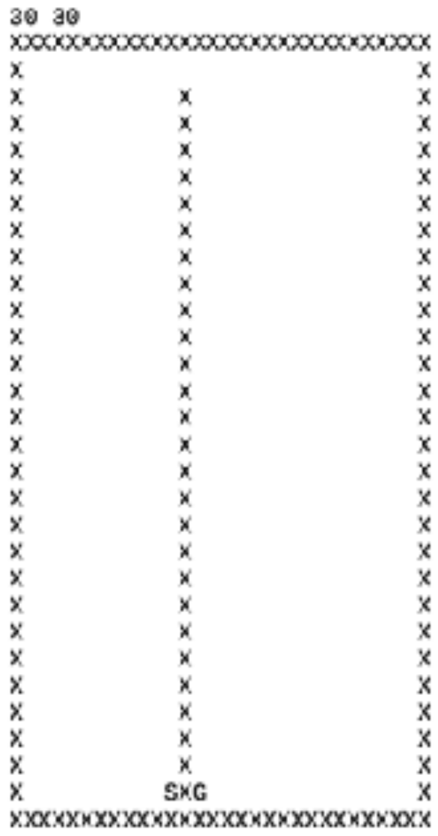
Does the straight-line distance (the absolute distance, ignoring any walls) from the start point to the goal point affect the running time of your algorithm? In other words, does how close the Start and Goal are affect how fast your algorithm finds a shortest "Manhattan" distance?

No, the time to get from Start to goal is the same every time, since the BFS finds the path back to start for each node, it will take the same amount of time no matter where the Start node is in comparison to the Goal node. This is confirmed by timing the solveMaze with no walls and moving the Start and Goal node to different distances from each-other.

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?



```
30 30
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X                            X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X          SXG               X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

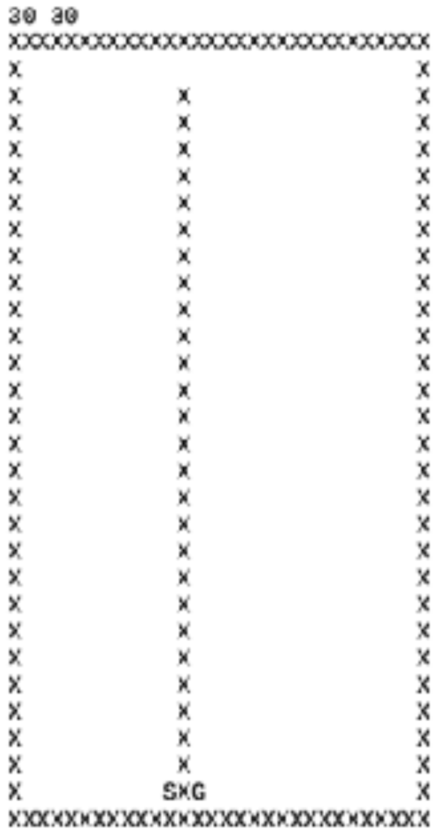
The difference between the straight line difference and the actual solution path is that the straight line path doesn't regard the walls in the maze so it can just take the shortest path from Start to Goal which is a straight line. An example of them being very different is in the figure to the right. The straight-line solution only goes one step to find the goal, while the actual solution must go up over the wall and down again to reach the goal. The straight line solution may be shorter but as the last question revealed, the run time is the same so might as well provide the accurate actual solution since it takes the same amount of time.

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 into account the density of the maze (how many wall segments there are in the field).

The worst case performance of the algorithm is O(N^2) since there are no walls in the worst case and there would be N*N (N being the N that was defined in the question) nodes in the maze meaning the BFS would have to iterate through N^2 nodes. The walls would have an effect on the algorithm if we were talking average case since but it would be a constant effect on the time making it E+N^2 with the number of edges being E. By adding a count variable into the breadth first search algorithm and incrementing the count every time the algorithm checks if the node has been visited it comes to a number that is around N^2 when the mazes is a square.

Hypothesize what effect using depth first search would have on the path planning? Provide an example maze for the purpose of this discussion. Would depth first solve the above example in a faster or slower manner?



```
30 30
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X                            X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X           X                X
X          SXG               X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Using a depth first search on the path planning has a big drawback because it depends all on what neighbors you look at first. For example in the above figure the solution would be achieved if the DFS would look at North first, then East, then South and it would be faster than the BFS and use less memory. however if the search looked at East first, then North, then West then the solution would be different and the run time would be about the same since it looks at every node. DFS is very reliant on what direction is chosen first, an example where you would want DFS is if you were traversing a family tree and you want to find who is still alive. Then you would know you would have to look towards the bottom and you could tailor the search to go South first making it more efficient than a BFS.

Say you created an entire matrix of nodes representing the maze and did not store any edges. To know if an edge exists for any node at (R,C) you would have to check (R-1,C), (R+1,C), (R,C+1), and (R,C-1). Can you think of another algorithm to find the shortest path from a given start (R1,C1) to a goal (R2,C2).

If the algorithm kept looking in one direction until reaching a null node then headed into a different direction and so on until it finds the goal or gets locked in a corner by null nodes or already visited nodes. If a side has already been visited and is otherwise stuck by null nodes, the algorithm goes to the previous node and looks in another direction. This algorithm, would not be efficient but would eventually find the goal.

How many hours did you spend on this assignment?

We spent about 10 hours on this assignment.