Assignment 8 Analysis:

1. I stayed with the same partner from the previous assignment, and the partnership worked really well. We made sure we contributed equally and we are good at communicating with each other to complete the assignment. We plan to continue working together for the next assignment, as well.
2. The visiting order that resulted in going to a further Goal on bigMaze_multigoal was check the left node, then the right node, then the node below, then the node above. I added a private instance variable to track how many nodes were searched while trying to find the path. Specifically, I incremented the variable every time the DFS method executed the next.cameFrom statement. I got a final answer of 151 nodes, so it searched through 151 nodes while trying to find the path to the goal.
3. Yes, it is possible. If one of the goals is in an almost straight path to the left, then depth first search will only have to search through the number of nodes it takes to get to that goal, because we begin our depth first search by looking at the left node. Breadth first search would still have to check the left path, the right path, up and down, etc. to find the closest, which means it searches every path to all those goals to ensure it has the shortest path, and thus searches more. An example of this maze type is below. Depth first search will only search the path directly to the left, while depth first search must check that path AND the path to the goal directly above it, which means it searches all possible paths to both which is clearly more than depth first search, even though the path to the above node is shorter.
   ```
   XXXXX
   XG X  X
   XX   GX
   XG    SX
   XXXXX
   ```
4. If a maze had a goal that is directly above the start node, breadth first search would catch it in constant time because it would check that neighbor and start its path before it continued on a path to other goal nodes. Depth first search, however, looks at the nodes above it LAST, which means it must go through all possible paths starting from the left neighbor node, then the right neighbor node, then the lower neighbor node, all before it even checks if the upper neighbor is a goal node. It would essentially require searching through all the nodes in the graph before it got to the actual goal node, simply because the upper node is checked last in its current order. An example of this maze is below
   ```
   XXXXX
   X X XX
   X X  GX
   X      SX
   XXXXX
   ```
5. For my experiment, I used the maze generator to create a random, large graph that was 50x50, had 30% wall density, and had five goals. After that, I had two statements that would find a path through the graph—I first did depth first search, then I had it use breadth first search on the same maze. In my graph class, I added one instance variable

that would track how many nodes depth first search looked at, and another that tracked how many nodes breadth first search looked at. I added increment statements to each iteration of the loops that checked new nodes in both the DFS and BFS methods, and printed out the value of the above instance variables once I had run the program. On average, I found that for a 50x50 maze with 5 goals, depth first search looks at 272 nodes before it finds a goal. Breadth first search, on the other hand, looks at an average of 292 nodes before finding the shortest path. Therefore, on average, breadth first search searches more nodes before finding a goal. These results show that if a program requires a path but needs to be more efficient with time, it should use depth first search. However, if the program is trying to find the shortest path to make a solution more efficient for a different part of the program, it should use breadth first search. Breadth first search can optimize other parts of the program by ensuring it uses the shortest possible path. While it isn't as efficient in the beginning as depth first search, if the path is being used in other parts of the program, then overall breadth first search will be more efficient. However, if finding a path is all that is required and it isn't used very much outside that instance, it is more efficient to use depth first search. In my experiment, they were pretty close to searching the same amount of nodes, so for smaller programs it may not matter too much. However, for larger programs, it is important to understand the purpose of the path you area getting so that you can properly choose between depth first or breadth first.