Kelly Romer

1. What is the key difference between depth first search and breadth first search in your agent? How did you design your functions to be generalizable?

Depth first search searches all the way down a branch of the graph before returning to the root node, whereas breadth first search searches a level of neighbors in the graph at a time, starting with the root node's neighbors. For depth first search, I used a stack because it can track the path in a LIFO manner. This means that if, for example, the traversal has reached a dead-end, or maximum depth, we can pop items from the stack accordingly and get back on the correct path. I designed by function to be generalizable by using the SearchProblem class methods such as getStartState() and getSuccessors(). I also figured out that by pushing a tuple to the stack that I could keep track of the current node as well as the cumulative path, which I ended up doing for the other two search problems as well. For breadth first search, I used a queue. When I was reading about BFS, I learned that it is important to visit child nodes in the same order that their parents were discovered. A queue is a great way to implement this because it operates on a FIFO basis. Much like the DFS function, I generalized this function by using the SearchProblem class and staying close to the general algorithm for BFS.

2. What is the difference in cost for UCS with the StayWest and StayEast search agents you tested with in Question 3? What behavior is encouraged by changing the cost function? Why could this be useful?

	StayWest	StayEast
mediumDotted	Cost: 17183894840	Cost: 1
mediumScary	Cost: 8719479864	Cost: DIED

As the table shows, the cost for StayWest is typically high, while the cost of StayEast was very low (1). Changing the cost function encourages PacMan to take a different route depending on what will have the 'least expensive' cost. In our StayWest function, Pacman is penalized for moving to the West side of the board, and in StayEast, the opposite is true. It would be helpful to change the cost function depending on the layout of the maze. In our medium mazes, the West side of the board is a lot more complicated, so PacMan took a lot longer of a path to the goal when the StayWest function was applied. Another thing to consider when implementing a cost function, however, is the position of the ghosts. When the mediumScary maze was tested using the StayEast cost function, the PacMan died almost right away because of the ghosts. So, the flexibility of changing the cost function allows PacMan to adapt to its game environment accordingly. This will be useful when we are trying to maximize both efficiency and frequency of PacMan's wins.