Mitchell Griffith

Dr. Burns

CSC 575

22 February 2021

BFS Maze

For this assignment we were tasked with using breadth first search to solve a maze. The agent checks for walls and possible paths in the maze to move its way through and find that most optimal path every time.

The agent calls the moves() function which checks all four surrounding nodes to determine if they are a wall or not. If it is it stores them in a list of possible moves to make. This means that for each node the agent visits it is checking four other nodes. This is a pretty costly amount of movement and checking but it is a quality of breadth first search. Being complete and optimal comes with a price tag in this scenario. That is quite a bit of nodes to check on each time a move needs to be made. The amount of nodes visited in my bfs() function is 122. That means the agent checked 488 nodes throughout the whole run. This scenario is made better by using BFS in the maze environment. Just because we check four nodes each time does not mean we explore four each time. We do not explore walls which are marked with ‘X’ which limits the maze quite a bit. The number of explored nodes is also contained by the size of the maze the agent is searching its way through. If the maze was larger, I see how this can seriously impact bigO with the amount of checks that need to be made.

Another way the bread first search is limited in the maze environment is how we limit it from going backwards in the path. This is done through a list of previously visited nodes. In my code, I still create an instance of the previously visited node and add it to the frontier. I simply make all possible moves returned from moves() and then check if the node has already been explored. This is certainly a hit to bigO which could be resolved. The visited nodes and history were two of the toughest parts of the assignment for me so when I got familiar with a solution, I was hesitant to change it up. Although I still create instances of previously visited nodes, I do not create a path from that created node. This saves a lot of space and time which I was curious about when first learning about BFS and thought about the maze.

While the checked list helps stop the agent from going down pointless rabbit holes, we also have to keep the list of visited nodes loaded in all of the time. Again, referencing the amount of total nodes visited, 122 is not a crazy amount of nodes to visit but this is also a fairly small maze environment. This could get out of hand with a much larger maze with many different branching paths. The longer the list gets, the longer it takes each time to check if the node has already been visited. While this is a long list to check, it is not whole maze instances that are being saved to do this work, it is just coordinates which helps with space.

Combining those two constraints meant that many times the only option for the agent was to move in one direction. Which limits the amount of mazes that needed to be created and paths to be explored. This is an effect of the maze environment.

There is not a time where I have more than two levels of maze instances loaded into memory. Once I use a level to discover another level, I replace the upper with the lower and reuse the lower level array to discover the next level. That sentence is tough to read but it was the easiest way to describe the way I store the mazes.

I also print out the length of the string of instructions it takes to get to the end of the maze. I went through and counted the amount of steps to make sure the agent was finding the most optimal path and it always did. I also made sure that there was not a quicker way and I could not. This showed me that the agent using bfs() would always find the shortest possible path to the goal if given one.

In conclusion, I was showed how breadth first search functions and how it can become not an optimal search method as a maze gets larger. When analyzing these statistics I saw not only how the breadth first search is contained by the maze environment, but also the size of the maze. This part of the assignment also helped me understand just how much space can be taken up as the amount of nodes to explore grows. While breadth first search has been fascinating to learn and is complete and optimal, I also now understand how it can get out of hand.