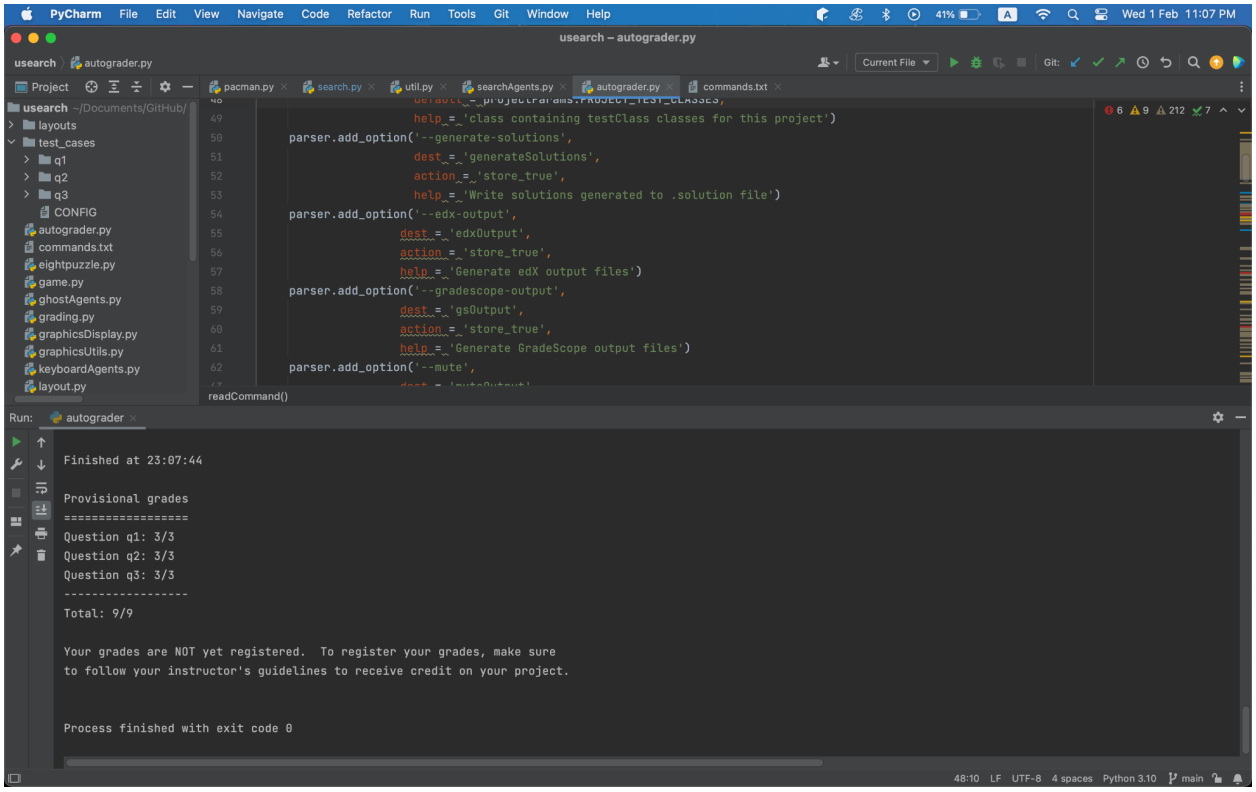
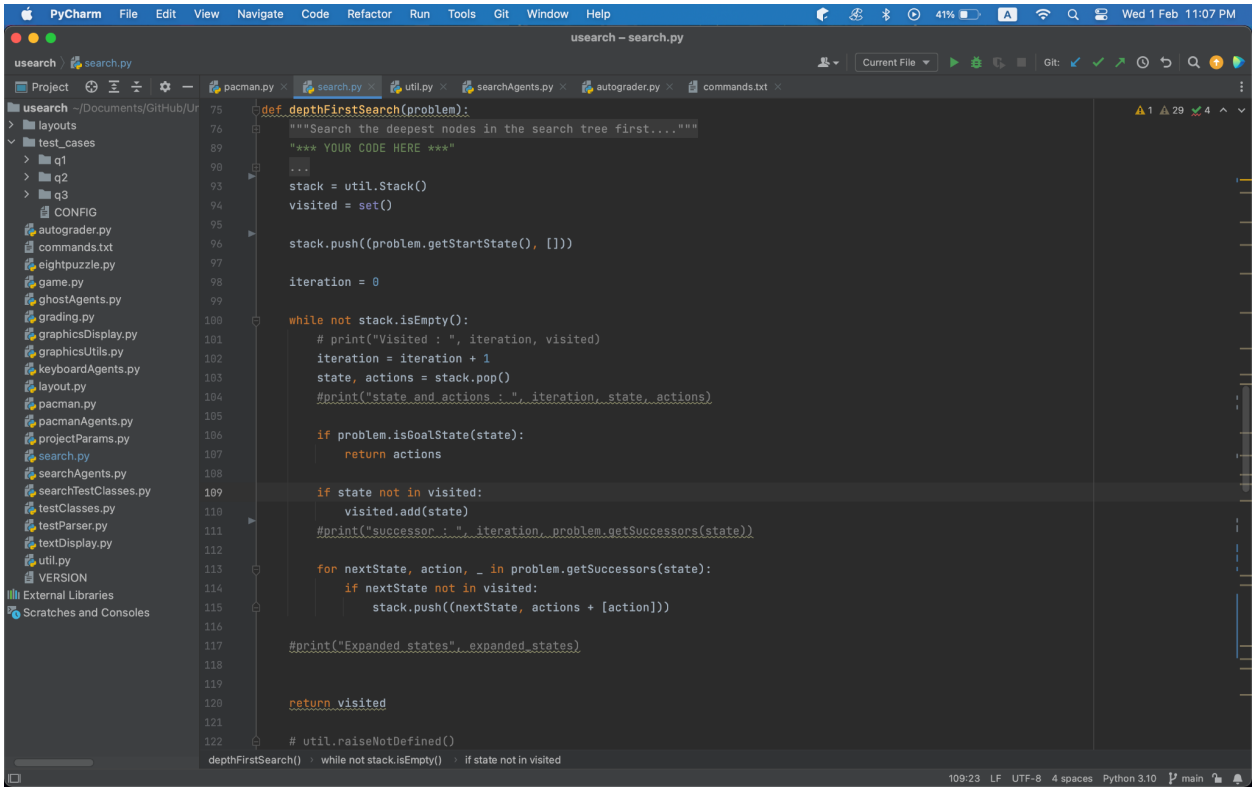
First, I want to highlight that my DFS, BFS and UCS passed the autograder test.



Now while implementing DFS;
I had to make sure of few things; like using stack, keep track of a visited array so that I don't visit a node twice, keep track of the fringe and actions to get to some state.

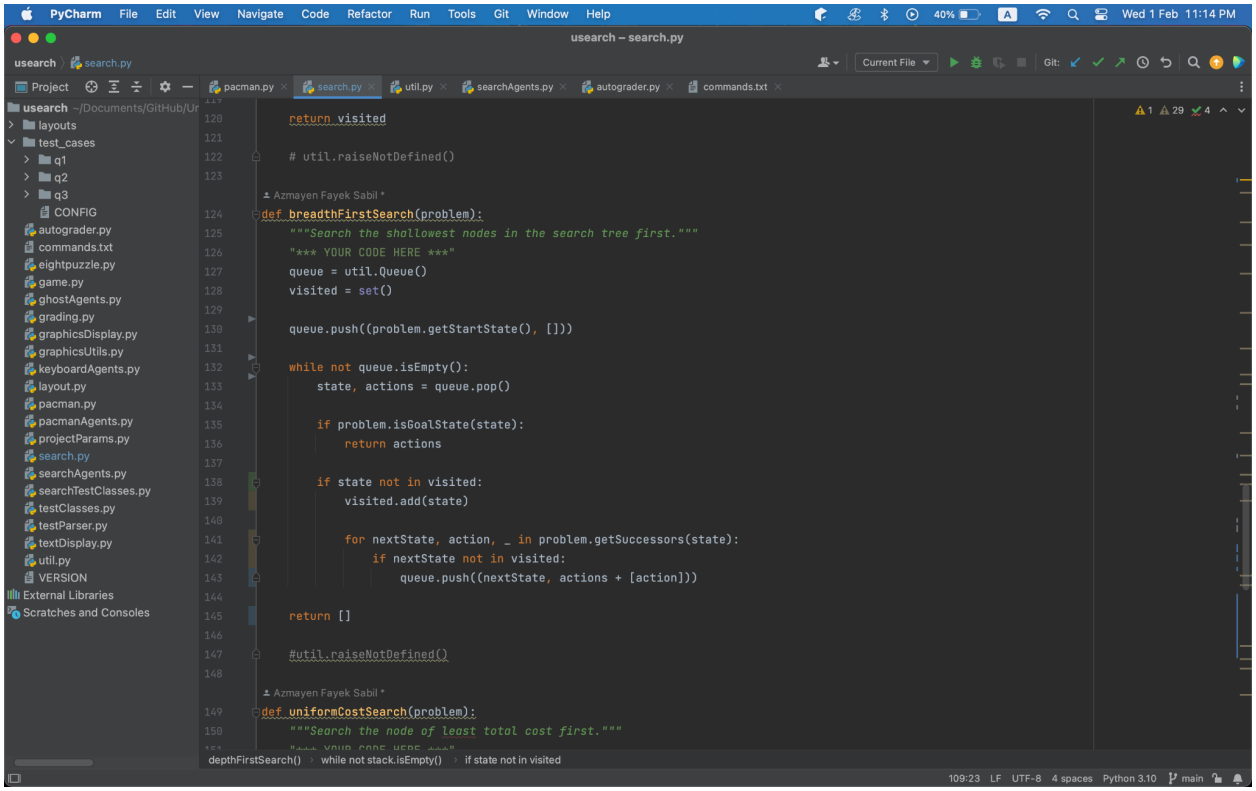


For the debugging reason I had added some extra lines of codes to track the behavior of my algorithm. Rest of the code is really straightforward. DFS basically means go as deep as possible and thats what I did here.

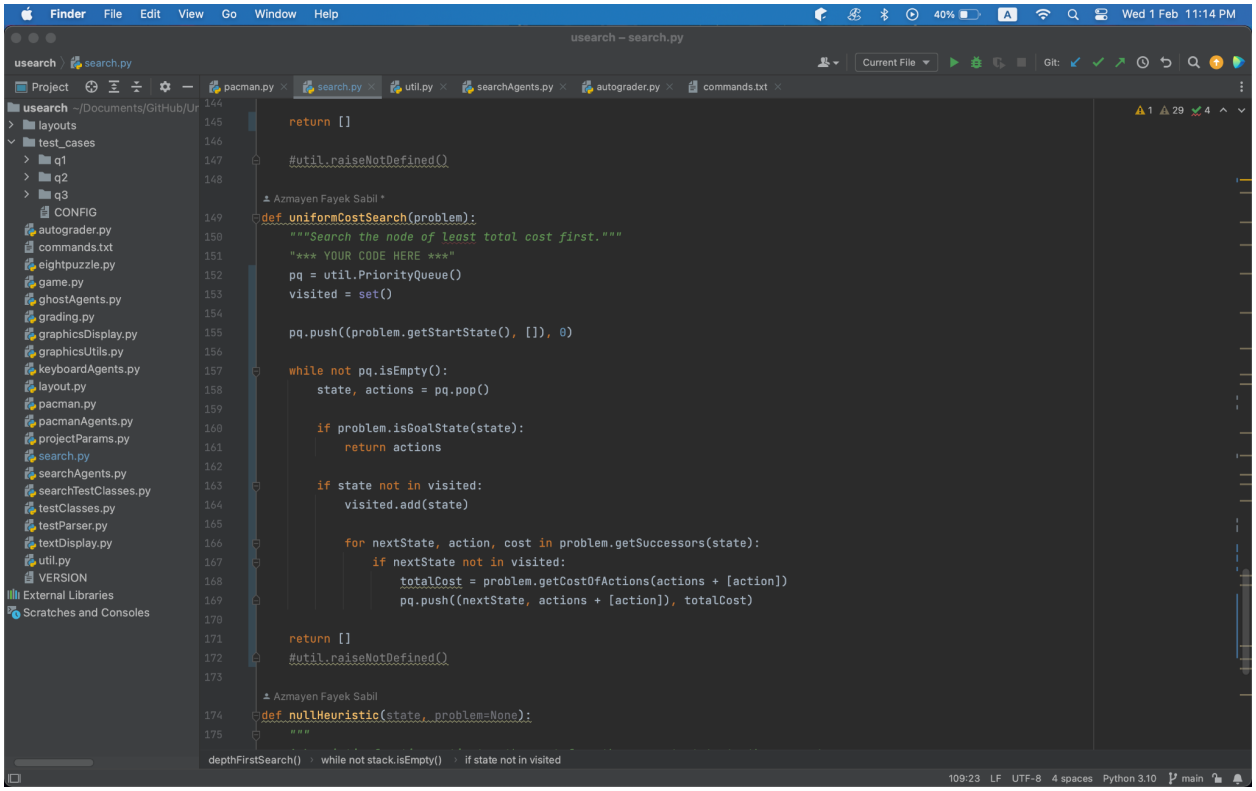
Now while implementing BFS;

I had to make sure of few things almost similar to DFS but the was is different; like using queue here, keep track of a visited array so that I don't visit a node twice, keep track of the fringe and actions to get to some state.

BFS means we visit the shallowest level first . And everything else is straight forward.



Now in case of UCS,
We also keep in mind about the cost of nodes and keep track of that also. We used priority queue here for that reason.



```
    return []

    #util.raiseNotDefined()

Azmayen Fayek Sabil *
def uniformCostSearch(problem):
    """Search the node of least total cost first."""
    "*** YOUR CODE HERE ***"
    pq = util.PriorityQueue()
    visited = set()

    pq.push((problem.getStartState(), []), 0)

    while not pq.isEmpty():
        state, actions = pq.pop()

        if problem.isGoalState(state):
            return actions

        if state not in visited:
            visited.add(state)

            for nextState, action, cost in problem.getSuccessors(state):
                if nextState not in visited:
                    totalCost = problem.getCostOfActions(actions + [action])
                    pq.push((nextState, actions + [action]), totalCost)

    return []
    #util.raiseNotDefined()

Azmayen Fayek Sabil
def nullHeuristic(state, problem=None):
    """
```

And everytime we visit a node we update the node with a new cost and compare it with other costs to get to that node. So that we can be sure that there is no other less costly node to get to that node. This way we can reach our goal state with an optimal solution.

Resources mainly I took help from: progamiz, geeksforgeeks