COEN 166 Artificial Intelligence

Lab Assignment #4: Search II

Gagan Gupta **SCU#00001478479**

Problem 1 A* Search:

Function 1:

```
def aStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""

    from util import PriorityQueue
    x = PriorityQueue()
    path = []
    visited = []
    start = problem.getStartState()
    x.push((start,[]),0)

    while not (x.isEmpty()):
        popped = x.pop()
        pState = popped[0]
        pPath = popped[1]
        visited.append(pState)
        if problem.goalTest(pState):
            return pPath
        futureActions = problem.getActions(pState)
        if len(futureActions)!=0:
            for action in futureActions:
                tempState = problem.getResult(pState, action)
                tempPath = pPath + [action]
                tempCost = problem.getCostOfActions(tempPath) + heuristic(tempState,problem)
                if tempState not in visited and tempState not in (state[0] for state in x.heap):
                    x.push((tempState,tempPath),tempCost)
```

Comment: A* Search for a Pacman Maze in which we use the PriorityQueue from util. The item we push onto the PriorityQueue contains the current state and the actions taken to get there as well as an initial priority of 0. We first push the starting state and go into the while loop where we loop until the PriorityQueue is empty or the goal state is reached. The first action is to pop the most prioritized element from the PriorityQueue and add that state to already visited. From that state we also get a list of the possible actions to take. If there are actions to take, we go through each action, find the state that results from that action, add that action to the path popped from

the queue, and generate a new cost based on the actions taken to get to the state and the heuristic value of the that state. If that state hasn't been visited and isn't already in the PriorityQueue, we push the resulting state and concatenated path along with the new cost onto the PriorityQueue. When we hit the goal state in the search, we return action list that was popped from the goal state. Pacman then follows this list of actions to reach the goal state.