## Autonomous Systems Lab - Session 1 - Search Jordan Harris

Q5. Finding All the Corners

Methods

def breadthFirstSearch(problem):

*"""Search the shallowest nodes in the search tree first."""*

def shortestBFSPath(graph, start, goal):

State Representation:   
**(((node,[empty array for visited goals]), ‘Action’, Cost)**  
*Note: the cost is not used in this algorithm.*  
  
Description:  
For this question a shortest path BFS algorithm to recursively traverse a graph, which is generated from the pacman grid world *minus* the walls, to find the shortest path to a goal node. The most recent path that is found is saved, and upon each iteration it is checked to see if it is still the shortest to the goal node. Then the next goal node is sought out.   
  
The method then converts the shortest path found to an action representation for the agent to follow.

*Note: This method is segmented at ‘Corners’ to handle both Q2 and Q5, which have a different state representations*

for questions 6 and 7 describe your heuristics and

justify why they are admissible / consistent.

Question 6. Corners Problem: Heuristic

Methods

def cornersHeuristic(state, problem):

def manhattan\_dist(xy1, xy2):

State Representation:   
**(((node,[empty array for visited goals]), ‘Action’, Cost)**  
**open.push((startState, actionList), heuristic(startState, problem))**

Description:  
  
This heuristic for an A\* implementation considers a start stating state or a new start at recursion. The state space keeps track of if it may have already visited any of the four corners/goals. Then lists out the unvisited corner/goals to compute the Manhattan distance to each of the goals. It then selects the corner with minimum manhattan distance and keeps down each distance to assess the minimum path length required. The heuristic provides a value that is no more than the best possible and optimal cost and is never negative which proves *admissibility*. It also expands only ~901 nodes and implies *consistency*.

Question 7. Eating All The Dots

Methods

def foodHeuristic(state, problem):

def euclidieanDistance(pointA, pointB):

def closestPoint(fromPoint, candidatesList):

State Representation:   
**(((node,[empty array for visited goals]), ‘Action’, Cost)**  
**open.push((startState, actionList), heuristic(startState, problem))**

Description:

This heuristic also uses an A\* implementation, and is very similar to the previous questions A\* in its set up but all there are many more goal states to consider. This heuristic iterates through all food on the map by calculating the euclidieanDistance from the current state to the to each food/goal choosing the in the end the shortest distance. The heuristic also provides a value that is optimal, non-trivial & non-negative which proves admissibility. It also expands only ~10908 nodes and inspires consistency.