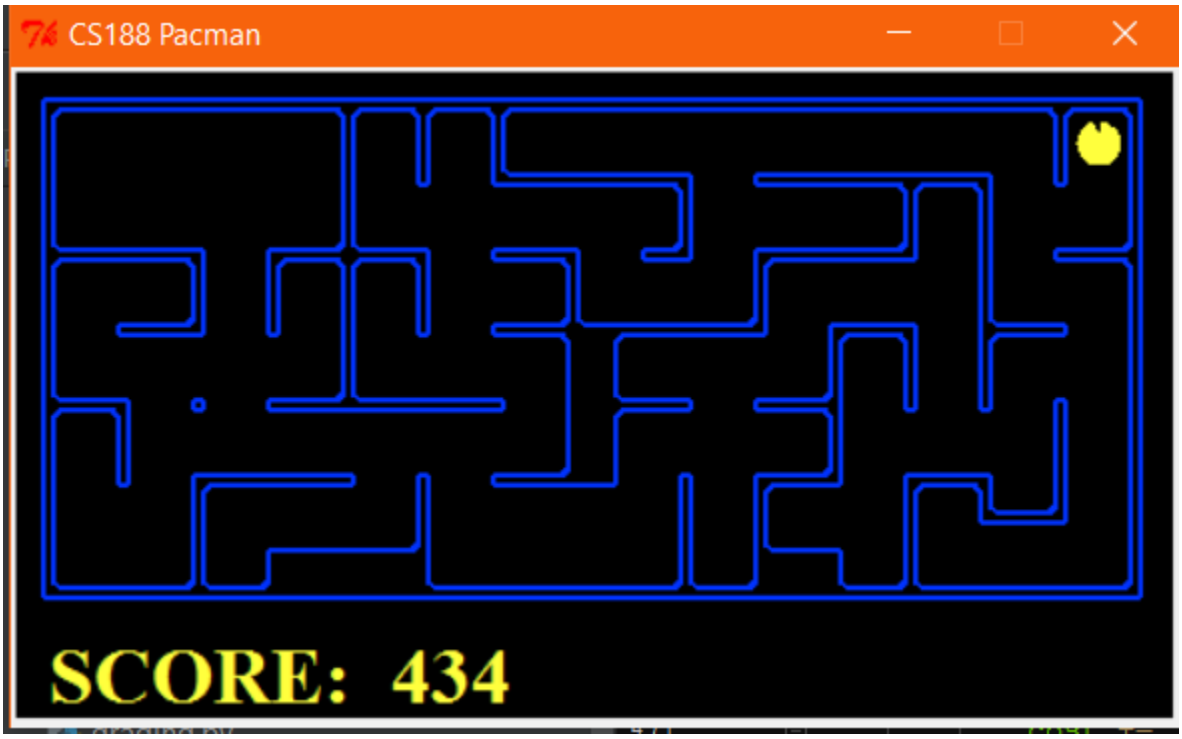
Richard Robinson
Pacman Part 2
Due: February 27,2018

PACMAN: Part 2



Overview/Abstract:

This project is about developing searching algorithms to evaluate an efficient path for the Pacman game. This path is found through expanding upon game states and weighing potential future paths in order to find a path of relatively low cost.

Problem Statements:

1) Solve pathing for accessing four corners
2) Apply a heuristic function for visiting corners
3) Implement a solution for Pacman to eat all the food

Solution Design / Implementation

FOUR CORNERS:
1) Added a list holding corners already visited as a state
2) Made a local variable in successors set to the currently visited corners
3) If the next state was a corner in the successor function and not already visited, add it
4) Call the successor function until the visited corners list contains the same values as the corners

5) Goal state is reached! Work is done

CORNERS HEURISTIC:

1) Made a heuristic function to find the Manhattan distance given two (x,y) coordinate pairs, named positionManhattanHeuristic. This simply uses the sum of the difference of the absolute values of the x,y coordinate pairs

```
def positionManhattanHeuristic(position, position2, info={}):
    "The Manhattan distance heuristic for a PositionSearchProblem"
    xy1 = position
    xy2 = position2
    return abs(xy1[0] - xy2[0]) + abs(xy1[1] - xy2[1])
```

2) Made a recursive function that uses tail recursion through a passed current coordinate and a list called recursiveManhattan. This creates a list of potential results, parses the list, and selects the closest item within that list to the current point. It then calls itself with the list missing that point.

```
def recurseManhattan(position, list):
    if len(list) == 0: return 0

    sums = []
    min = 99999
    minIndex = 0
    for xy in list:
        sums.append( (positionManhattanHeuristic(position, xy)) )

    for x in range(0, len(sums), 1):
        if sums[x] < min:
            min = sums[x]
            minIndex = x

    return min + recurseManhattan(list.pop(minIndex), list )
```

COLLECT ALL THE DOTS

1) Weighed the already made recurseManhattan using the pellet locations and added the number of pellets left to grab for the next movement. Did not have to implement anything new

```
position, foodGrid = state
return recurseManhattan(position, foodGrid.asList()) * .5 + len(foodGrid.asList())*.5
```

Solution Result/ Evaluation

Passed all tests with full marks on auto grader.

All components work as intended. There could be improvement on the heuristics, but the current ones satisfy the demands put forth for this project.

Conclusion/Reflection

This was a wonderful and mostly enjoyable project. I learned a lot about python and data types along with their usage through this project. I absolutely love how the solutions from prior sections are so easily expanded upon to supplement the next component. Through this project I have developed a much better understanding of the code base for this project.

I believe concepts learned within here can be applied to later projects. Reusability was very firmly engrained within this project. This greatly improved my understanding of searching algorithms and as of now I believe I have developed a solid understanding of how they work.