Pacman Project - Final Report

In this project, we implemented various search algorithms for our Pacman to reach certain locations as well as methods for it to collect food pellets most efficiently. The process of expanding nodes on the frontier in each algorithm continues until either a solution is found or there are no more states to expand. Although these search algorithms share this basic structure, they vary largely on how they choose which state to expand next.

We successfully added in the implementation of depth-first search, breadth-first search, A\*, and UCS in the file.

Q1. Depth-first search always expands the deepest node in the current frontier of the search tree. (It uses LIFO logic which is implemented using Stack in our code). This is not an optimal solution and is more costly than other solutions.

Q2. Breadth-First search involves expanding the root node entirely first before proceeding to do the same with deeper nodes as well. It uses FIFO logic implemented using a queue. This gives a lower cost solution. The number of nodes expanded is very high for bigger sizes of maze.

Q3. Varying the Cost Function: In this, the node traversals were assigned different costs that were determined by the distance. The algorithm then expanded nodes with lower costs first. A priority queue was used to implement this.

Q4. A\* Search: We implemented this method and got a lower time for search implementation. The number of nodes expanded was also reduced. Manhattan distance was used to calculate heuristic value for each node

Q5. Finding All the corners: We detected the 4 food pellets on 4 corners of the maze

Q6. ​​Corner Problem:Heuristic:The corner with the longest Manhattan distance is the one which needs to be visited at last. We use this longest Manhattan distance as a heuristic one. Calculate the Manhattan distances between current position and unvisited corners to find the smallest distance to add to the heuristic value. After updating the heuristic value, we replace the current position as the closest unvisited corner and do it again, until all corners are visited.

Q7. Eating all the Dots: We first make a list of all the spots in the maze that include food pellets. The distance function is used to determine a node's Heuristic. Pacman's distance from each food position in the maze is defined as the heuristic. A list is used to keep track of these data. To find the farthest point, we find the maximum heuristic available.

Q8. Suboptimal Search: To discover the food pellets closest to Pacman, we employed the breadth-first search algorithm we used in question 2. The isGoalState function returns true if there is food in that position in the current state (as determined by the list created in the foodHeuristic function), and false otherwise.

Interesting:

1. The most interesting part of the project was when we created a working pacman which uses breadth-first search to reach the food pellets closer to it.
2. It was interesting to revisit an old game and mess around with it from the backend.

Challenges:

1. All of us as a team have experience using Python3, and not so much Python2. Although it was a smaller challenge, we found ourselves having to correct our code while debugging etc. It would have been more convenient using Python3.
2. It was harder to optimize the code for the last question to reduce the number of nodes it expands. Initially for the code we wrote, the algorithm expanded around 8000 nodes, after optimizing it the algorithm only expanded 4182 nodes.

Activities we simply hated:

1. It was a cumbersome task to have to read through all the existing files and understand the meanings of the functions and variables defined. It took up a lot of time before we could start with our solutions.