PROJECT 01: THE SEARCHIN' PAC-MAN

ARTIFICIAL INTELLIGENCE

CSE 537

INTRODUCTION

The report is based on the The Searchin' Pac-Man – a game enjoyed by all! We have handled multifarious Pac-Man scenarios by applying general search algorithms like Depth-First Search (DFS), Breadth-First Search (BFS), Uniform-Cost Search (UCS) and A* Search & implemented CornersProblem with/without heuristics which enabled the Pac-Man agent to effectively and efficiently reach a particular location. We also employed these search functions to resolve FoodSearchProblem with consistent heuristic to collect all the food dots.

Files utilized –

- ✓ Search.py
- ✓ searchAgents.py

Files provided –

- pacman.py
- game.py
- util.py

Authors/Project Owners – Rishabh Khot (SBU ID: 112682983) & Shubhi Nigam (SBU ID: 112672816)
Stony Brook University

1. **Depth-First Search Algorithm**

   Search Agent traversing through the graph may encounter repetitive states, i.e. the ones that were visited before. This may lead to redundant work, memory and time wastage. To overcome this problem, we followed the Depth-First Search Algorithm (uninformed search) and implemented the solution by maintaining a separate 'visited' list. The fringe used is a stack data structure.

   Implementation Details:

   1.1

   - Number of nodes expanded: 16
   - Total Cost: 10
   - Running Time: 0.0s

   1.2

   - Number of nodes expanded: 146
   - Total Cost: 130
   - Running Time: 0.0s

   1.3

   - Number of nodes expanded: 391
   - Total Cost: 212
   - Running Time: 0.0s

2. **Breadth-First Search Algorithm**

   By implementing Breadth-First Search Algorithm, we restricted Search Agent to traverse the shallowest node first. The 'visited' list is maintained to keep track of all the nodes touched by the agent during its journey to find the optimal solution. The fringe used is a Stack list.

   Implementation Details:

2.1

- Number of nodes explored: 269
- Total Cost: 68
- Running Time: 0.0s

2.2

- Number of nodes expanded: 620
- Total Cost: 210
- Running Time: 0.0s

3. Uniform Cost Search

UCS is a variation of BFS search algorithm to accommodate the cost of each path taken. In this algorithm, the fringe list is maintained as a Priority Queue. This queue holds onto the node information and the cost associated with it which helps Search Algorithm to follow a path of lowest and optimum cost to its destination.

Implementation Details:

3.1

- Number of nodes expanded: 270
- Total Cost: 68
- Running Time: 0.0s

3.2

- Number of nodes expanded: 187
- Total Cost: 1
- Running Time: 0.0s

3.3

- Number of nodes explored: 108
- Total Cost: 68719479864
- Running Time: 0.0s

4. **A* Search**

A* Search is a very effective type of informed search algorithm. It takes into account a heuristic function which is the estimate of cost from node 'N' to the goal. The search works on the strategy of summation of path cost and heuristic cost function and therefore, with an admissible heuristic, search agent will reach the goal optimally. Apart from the data structures mentioned in the above problem-search statements, this search requires to maintain a heuristic function.

Implementation Details:

4.1

- Number of nodes expanded: 549
- Total Cost: 210
- Running Time: 0.0s

5. **CornersProblem – Finding All the Corners**

A* is used to guide the search agent to find a path through the maze that incorporates all 4 corners in the solution. The function accommodates the starting location and coordinates of all the 4 corners.

Implementation Details:

5.1

- Number of nodes expanded: 252
- Total Cost: 28
- Running Time: 0.0s

5.2

- Number of nodes expanded: 1966
- Total Cost: 106
- Running Time: 0.0s

6. CornersProblem with Heuristic

The optimal solution to CornersProblem is found which contains the heuristic that is both admissible and consistent.

Implementation Details:

6.1

- Number of nodes expanded: 801
- Total Cost: 106
- Running Time: 0.8s

6.2

- Number of nodes expanded: 10
- Total Cost: 7
- Running Time: 0.0s

7. FoodSearchProblem with foodHeuristic

The search problem requires Pac-Man search agent to eat all the food, i.e. traverse all the dots in the least possible steps. For the solution, we implemented a function called foodHeuristic which keeps track of all the distances and stores them in problem.heuristicInfo dictionary and the function returns the maximum food distance.

Implementation Details:

- Number of nodes expanded: 4137
- Total Cost: 60
- Running Time: 2.9s

CONCLUSION

A* is slightly better than BFS as it expanded 549 nodes in contrast to 620 for the total cost of 210. UCS and BFS performed approx. the same for medium maze. Performance of DFS was much better than BFS for tiny and medium mazes.