CSE 537: Project 1: The Searchin' Pac-Man

Project Report

Submitted by
Riddhi Rex(SBU ID: 111464274) and Rajesh Mohan (SBU ID:111498022)

Introduction:

Pac-Man is an arcade game developed by Namco and first released in Japan in May 1980.[2][3] It was created by Japanese video game designer Toru Iwatani. It was licensed for distribution in the United States by Midway Games and released in October 1980. The player controls Pac-Man through a maze of various dots, known as Pac-Dots, as well as four multi-colored ghosts: Blinky, Pinky, Inky, and Clyde. The goal of the game is to consume all the Pac-Dots in a stage in order to proceed to the next one. Between some stages, one of three intermission animations plays. The four ghosts roam the maze, trying to kill Pac-Man. If any of the ghosts touch Pac-Man, he loses a life; when all lives have been lost, the game ends[1].

Search algorithms such as Depth first search, breadth first search, uniform cost, and A* search are implemented and their stats such as number of nodes expanded, memory usage and running time for each search strategy is discussed.

Question 1: Depth First Search

Depth first search is a graph search problem where the deepest node in the frontier is visited first. Here the fringe list is a LIFO structure. The successors of the node are added in the fringe list and removed from the list when they are expanded. Thus the search proceeds to expand the deepest unvisited nodes from the fringe list.

Execution Logs:

```
python pacman.py -l tinyMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores:        500.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
python pacman.py -l mediumMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
```

```
Average Score: 380.0
Scores:        380.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
python pacman.py -l bigMaze -z .5 -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.2 seconds
Search nodes expanded: 390
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Question 2: Breadth First Search

Breadth first search is where the nodes are visited depthwise. First the root and its immediate successors are visited and then their successors in the next level are visited. Nodes in each depth are expanded first before moving on to the next level. Nodes are pushed into a FIFO structure where each of the nodes is removed to be expanded.

Execution Logs:

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.1 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.2 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Question 3: Uniform Cost Search

Uniform cost search expands the node with the minimum path cost. This is done by using a fringe list to store all the nodes and their path cost in a priority queue. The state of the nodes along with its cost is pushed into the fringe list.

Execution Logs:

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.1 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
Path found with total cost of 1 in 0.1 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores:        646.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
Pacman emerges victorious! Score: 418
Average Score: 418.0
Scores:        418.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Question 4: A* Search

A* search is a form of best-first search where the nodes are expanded on the basis of total cost of reaching the node g(n) and the estimated cost to reach goal from that node h(n). A* search is considered to be both optimal and complete. For it to be optimal the h(n) heuristic has to be admissible which means that h(n) has to be the upper limit or the maximum cost to reach the goal. Manhattan distance is used to calculate the direct distance between the two points. A fringe list, a priority queue is used to store the state and the f(n) value which is the sum of total path cost and the heuristic value.

Execution Logs:

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a
fn=astar,heuristic=manhattanHeuristic

[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.2 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
Process finished with exit code 0
```

Question 5: Corners problem Search

The search problem is to find the food present in each corner of the maze. The shortest path has to be found to traverse all 4 corners of the maze. This uses the Breadth first search algorithm. The state contains current position and the list of corners visited. Since the list of visited corners is maintained, it is easy to keep track of the unvisited corners.

Execution Logs:

```
python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.1 seconds
Search nodes expanded: 418
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:        512.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
python pacman.py -l mediumCorners -p SearchAgent -a
fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 1.5 seconds
Search nodes expanded: 2432
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Question 6: Corners problem in corners Heuristic

A heuristic has to be developed for the Corners Problem. It has to be admissible which means the h(n) , the estimated cost to reach the goal from the node should be the lower bound for the actual cost to the nearest goal. This uses the a-star search algorithm. Manhattan distance is calculated between the current position and each of the unvisited corners to decide which corner is nearest to the current position and then that corner is first visited. This is repeated until all the corners are visited.

Execution Logs:

```
python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
Path found with total cost of 106 in 0.9 seconds
Search nodes expanded: 1736
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
```

```
Record:        Win

Process finished with exit code 0
```

Question 7: Food Heuristic

A heuristic has to be developed for the FoodSearchProblem which is both admissible and consistent. It uses the a-star search algorithm. Manhattan distance is calculated between the current position and each of the food and the closest food is visited first. This is repeated until all the food is consumed. This way the heuristic is kept admissible and consistency is maintained.

Execution Logs:

```
python pacman.py -l trickySearch -p AStarFoodSearchAgent
Path found with total cost of 60 in 27.7 seconds
Search nodes expanded: 13898
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win

Process finished with exit code 0
```

Files edited:

search.py	Contains the search algorithms
searchAgents.py	Contains search-based agents

Conclusion (Critical Analysis):

Search algorithms including Depth first search, breadth first search, uniform cost, and A* search were implemented for different maze sizes and their stats such as number of nodes expanded, memory usage and running time for each search strategy was found. It is seen that the A* star is more efficient than the BFS, DFS and the uniform cost search. The search nodes expanded was greater for BFS compared to the DFS and Uniform Cost Search. However BFS, DFS and Uniform Cost search each perform better among the rest depending on the position of the search node.

References:

1. https://en.wikipedia.org/wiki/Pac-Man