CSE 537 - Artificial Intelligence

Report: Project 2

(Multi-Agent Pac-Man)

Khan Mostafa 109365509 **Abhijit Betigeri** 109229784

{khan.mostafa, abhijit.betigeri}@stonybrook.edu
Department of Computer Science
Stony Brook University

Designing Agents namely Multi-Agent for classic Version of Pac-Man including Ghosts

Q1. Reflex Agent – Improvements in game considering food & ghost locations

Methodology Used: Reflex agent uses following strategy to evaluate action:

- Consider the location of nearest ghost; if a ghost is about to catch Pacman should run towards other direction. In this case the function should return a large negative number
- If the step eats a food, Pacman should favor this step. In this case return a very high value
- Consider distance to nearest food. Evaluation function would be reciprocal to this.

A sum of these three strategies is returned by the evaluation function. We are using Manhattan distance to compute distances.

Execution Details

The basic set1 with testClassic Layout could be cleared successfully with our defined evaluation function

Set 1:

```
python pacman.py -p ReflexAgent -l testClassic
```

Pacman Game status: Win
Total Score: 564

Set 2: We use the mediumClassic layout with one ghost – 10 games were run and results are as below.

```
python pacman.py --frameTime 0.1 -p ReflexAgent -k 1 -n 10
```

Average Score: 1052.2
Scores: 1485.0, 971.0, 1373.0, 1460.0, 675.0, 1056.0, -75.0, 1306.0, 980.0, 1291.0
Win Rate: 9/10 (0.90)
Record: Win, Win, Win, Win, Win, Win, Loss, Win, Win, Win

Set 3: Usage of mediumClassic layout with two ghost - 10 games were run and results are as below.

```
python pacman.py --frameTime 0.1 -p ReflexAgent -k 2 -n 10
```

Average Score: 855.8
Scores: 1089.0, 238.0, 273.0, 1368.0, 1256.0, 1703.0, 1679.0, 259.0, 331.0, 362.0
Win Rate: 5/10 (0.50)
Record: Win, Loss, Loss, Win, Win, Win, Win, Loss, Loss, Loss

Note: Repeated Run with 2 ghosts – Pac-Man wins with 50% signifying evaluation functions to quite good.

Set 4: Usage of openClassic layout repeatedly ie ten times

```
python pacman.py -p ReflexAgent -l openClassic -n 10 -q
```

Here are scores of each run, average score and win rate.

Average Score: 1257.5
Scores: 1260.0, 1257.0, 1257.0, 1258.0, 1244.0, 1260.0, 1264.0, 1257.0, 1259.0, 1259.0
Win Rate: 10/10 (1.00)
Record: Win, Win, Win, Win, Win, Win, Win, Win, Win, Win

Q2. Adversarial search Agent in the provided Minimax Agent Class

Methodology Used: We make the recursive Minimax decision from the current state wrt the number of agents, depth, successors, evaluation function during the particular state under consideration. Based on the agent value it acts as a minimizer or maximizer. Depth is decreased only when last agent has played its ply in this round.

Execution Set 1: for depth =4 with a sample set of 1000 games.

python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4 --numGames 1000 --frameTime 0 --fixRandomSeed --textGraphics
Win Rate: 436/1000 (0.44)
time taken: 0:28:16.454000 seconds

Execution Set 2: for depth =4 with an sample set of 100 games.

python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4 --numGames 100 --frameTime 0 --fixRandomSeed --textGraphics
Win Rate: 48/100 (0.48)
time taken: 0:04:55.984000 seconds

Execution Set 3: for depth =3 with an sample set of 100 games.

python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=3 --numGames 100 --frameTime 0 --fixRandomSeed --textGraphics
Win Rate: 35/100 (0.35)
time taken: 0:00:25.816000 seconds

Execution Set 4: for depth =4 with n=10 games with smartScorer giving number of nodes expanded at each state.

python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4, profile=True, evalFn=smartScorer –n 10
We have an evaluation function defined and with smart scorer we get the number of nodes expanded at each state.
For more details refer to report/MinimaxAgent_minimaxClassic_depth_4_smartScorer.txt

Note: Similarly smartScorer – complete details about the win rate, nodes expanded at each state are attached with depth=3,depth=4 with n=100,1000,10 combinations with file name suffix smartScorer.

Q3. Alpha-Beta pruning

Methodology Used: To decrease the number of nodes as evaluated by minimax algorithm by using two values alpha & beta representing the maximum score for maximizing player & minimum score for minimizing player. Naïve minimax expands a lot of nodes, even when it is apparent that the branch will never be played by the previous player. Alpha and Beta values act as the margin of truly possible values. If a node is evaluated to contradict this range, the branch is discarded; thus expansion of many nodes can be escaped.

Execution Set 1: for depth =4 with an sample set of 1000 games.
python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4 --numGames 1000 --frameTime 0 --fixRandomSeed --textGraphics
Win Rate: 656/1000 (0.66)
Time Taken: 0:04:53.418000 seconds

Execution Set 2: for depth =4 with an sample set of 100 games.
python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4 --numGames 1000 --frameTime 0 --fixRandomSeed --textGraphics
Win Rate: 63/1000 (0.63)
Time Taken: 0:00:32.480000 seconds

Execution Set 3: for depth =3 with an sample set of 10 games with smallClassic layout.
python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=3, evalFn=smartScorer, profile=True –l smallClassic –n 10 –framTime 0.01
Win Rate: 10/10 (1.0)

Note: Similarly as for minimax – AlphaBeta has a smartScorer – complete details about the win rate, nodes expanded at each state are attached with depth=3,depth=4 with n=100,1000,10 combinations with file name starting with AlphaBetaAgent and with suffix smartScorer.

Analysis:

We have the analysis tabulated in the alphabetaAgent_smallClassic_depth_3.txt and all the report text files - where we compare the alphabeta pruning to minimax in terms of the nodes expanded and how alphabeta is more efficient compared to minimax. % efficiency at each state is tabulated.

We can conclude that Minimax is used as strategy to find optimal moves and alphabeta pruning is a practical application of minimax algorithm. Pruning helps in improving agent's proficiency and time taken to choose next move without losing efficiency and that too in fever leaf node expansions.

Enclosure – Attached Report Files:

AlphaBetaAgent_minimaxClassic_depth_4_n_1000.2.txt
AlphaBetaAgent_minimaxClassic_depth_4_n_1000.txt
AlphaBetaAgent_minimaxClassic_depth_4_numGames_1000_fixRandomSeed.txt
AlphaBetaAgent_minimaxClassic_depth_4_numGames_1000_fixRandomSeed_displayed.txt
AlphaBetaAgent_minimaxClassic_depth_4_numGames_100_fixRandomSeed.txt
AlphaBetaAgent_smallClassic_depth_3.txt
AlphaBetaAgent_smallClassic_depth_3_numGames_100_fixRandomSeed.txt
AlphaBetaAgent_smallClassic_depth_3_smartScorer.txt
MinimaxAgent_minimaxClassic_depth_3_numGames_100_fixRandomSeed.txt
MinimaxAgent_minimaxClassic_depth_4.2.txt
MinimaxAgent_minimaxClassic_depth_4.txt
MinimaxAgent_minimaxClassic_depth_4_n_1000.txt
MinimaxAgent_minimaxClassic_depth_4_numGames_1000_fixRandomSeed.txt
MinimaxAgent_minimaxClassic_depth_4_numGames_100_fixRandomSeed.txt
MinimaxAgent_minimaxClassic_depth_4_smartScorer.2.txt
MinimaxAgent_minimaxClassic_depth_4_smartScorer.txt
MinimaxAgent_minimaxClassic_depth__numGames_100_fixRandomSeed.txt
MinimaxAgent_smallClassic_depth_3.txt
MinimaxAgent_smallClassic_depth_3_smartScorer.txt
ReflexAgent_openClassic_1000.txt