**CSE 537: Project 02: Multi-Agent Pac-Man**

**Project Report**

Submitted by

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

Introduction:

Multi-agent environment has multiple agents each making actions competitively to affect one another’s path as their goals are in conflict. Pac-man can be viewed as a multi-agent environment where the ghosts in the game form an agent and oppose the movement of Pac-Man which is the other agent. These two agents make their optimal move alternatively until one wins over the other. The agent can be one of two types, either a Reflex agent or an Adversarial search agent. Reflex agents play on the basis of evaluation of the next immediate game state and pick the best move based on an evaluation of the successor states. Adversarial search on the other hand works on the basis of looking ahead by several moves using a tree and it accounts for the possible moves made by the adversary as well. The adversarial agents that have been used in the project are Minimax, Alpha-Beta pruned Minimax and Expectimax.

Q1: Reflex Agent:

A reflex agent works purely on the basis of evaluation of successor game states given the current game state. A score is assigned to each successor game state and the program picks the state that has the maximum value. Since, eating all the food dots is our goal, we consider the number of food dots left as a factor. We also consider the distance of the closest food from Pac-Man’s current position along with the ghost distance as a factor. The inbuilt getScore() method is also used here to compute score as it keeps reducing by 1 as time passes. If the successor position is at a food dot, the score is increased. Pac-Man is penalized for staying still as well. As can be seen below, Pac-Man wins all ten games with an average score of 1246.9. The evaluation method fails in the case of two ghosts on the board.

python pacman.py -p ReflexAgent -l testClassic

Pacman emerges victorious! Score: 562

Average Score: 562.0

Scores: 562.0

Win Rate: 1/1 (1.00)

Record: Win

python pacman.py --frameTime 0 -p ReflexAgent -k 1

Pacman emerges victorious! Score: 1205

Average Score: 1205.0

Scores: 1205.0

Win Rate: 1/1 (1.00)

Record: Win

python pacman.py --frameTime 0 -p ReflexAgent -k 2

Pacman died! Score: 237

Average Score: 237.0

Scores: 237.0

Win Rate: 0/1 (0.00)

Record: Loss

Q2: Minimax:

Minimax is an adversarial search where the minimax value is recursively computed for each successor state. The traversal is done from the root till the leaves in the depth first search manner and minimax values are returned all the way to the top. Each node in the game tree picks the minimax value which is most optimal. Moves are done alternatively between MAX and the MIN nodes where the MAX nodes chooses the highest value among the options and the MIN nodes selects the least among the options. This is done until the terminal states are reached where the function is the utility value. In this case, Pac-Man is MAX node and the ghosts are MIN nodes. In the case of trappedClassic, Pac-Man rushes to the nearest ghost as it evaluates that it will die eventually and hence decides to maximize it’s score by dying immediately.

python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4

Pacman died! Score: -515

Average Score: -515.0

Scores: -515.0

Win Rate: 0/1 (0.00)

Record: Loss

python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3

Pacman died! Score: -503

Average Score: -503.0

Scores: -503.0

Win Rate: 0/1 (0.00)

Record: Loss

Q3: Alpha Beta Pruning:

In Minimax algorithm, each of the nodes has to be visited and the complexity is exponential with the depth of the game tree. So, instead of looking at each node of the tree, we can prune the parts of the tree which will not influence the final decision. This way looking up of large portions of the tree can be avoided. Alphavalue represents the highest choice among the children at any point along the path of MAX node and Beta value represents the least value of the children at any point along the path of MIN node and pruning is done based on these values. Initially, Alpha is set to negative infinity and Beta is set to positive infinity. As MAX nodes are examined, Alpha increases and as MIN nodes are examined Beta decreases. The search stops when Alpha crosses Beta.

python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic

Pacman died! Score: 181

Average Score: 181.0

Scores: 181.0

Win Rate: 0/1 (0.00)

Record: Loss

Q4: Expectimax:

In Expectimax, The MAX nodes work exactly the same way as Minimax and Chance nodes are introduced in Expectimax where the average of values of all its children is taken as the chance node value. So, MAX picks the chance node where the probability of getting a high score is high. So, in the case of trapped classic, Expectimax takes a chance that the ghost on the left will move down and give way for it to win whereas the AlphaBetaAgent suicides itself immediately. As seen below, ExpectimaxAgent wins six out of ten times and AlphaBetaAgent fails all ten times.

python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3

Pacman emerges victorious! Score: 515

Average Score: 515.0

Scores: 515.0

Win Rate: 1/1 (1.00)

Record: Win

AlphaBetaAgent:

python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Average Score: -501.0

Scores: -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0

Win Rate: 0/10 (0.00)

Record: Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss

ExpectimaxAgent:

python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10

Pacman died! Score: -502

Pacman emerges victorious! Score: 531

Pacman emerges victorious! Score: 531

Pacman died! Score: -502

Pacman died! Score: -502

Pacman emerges victorious! Score: 531

Pacman emerges victorious! Score: 531

Pacman died! Score: -502

Pacman emerges victorious! Score: 531

Pacman emerges victorious! Score: 531

Average Score: 117.8

Scores: -502.0, 531.0, 531.0, -502.0, -502.0, 531.0, 531.0, -502.0, 531.0, 531.0

Win Rate: 6/10 (0.60)

Record: Loss, Win, Win, Loss, Loss, Win, Win, Loss, Win, Win

Conclusion:

Multi-agent Pac-Man was played using four different search agents, namely Reflex, Minimax, AlphaBeta and Expectimax and all the implementation passed the autograder testcases. The Reflex agent is very slow to reach the goal but uses less space to store successor states. The adversarial search agents although quick to win, end up using more storage as they store nodes up to a depth of d. From the results collected, it is evident that Expectimax wins in situations where other agents like Minimax and AlphaBeta fail. Even though Expectimax fails in some games, it works off of maximizing its chance of winning and thereby wins over multiple games played.