CSE 537 Artificial Intelligence

Project 2 — Multi-Agent Pac-Man

Group Members:

Weituo Hao (ID: 109241801), Xiufeng Yang (ID: 107992992),
Hongjin Zhu (ID: 109748818), Ruonan Hao (ID: 108719989)

Date:

October 16, 2014

Problem Description

In this project, we have designed agents for the classic version of Pac-Man, including ghosts. Along the way, we have also implemented both the minimax algorithm and alpha-beta pruning and tried our hands at evaluation function design.

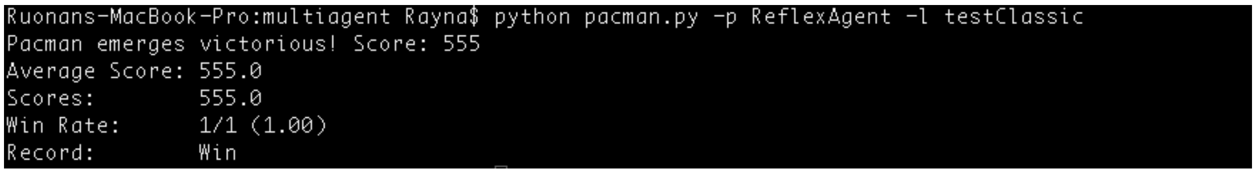
Question 1 — Evaluation Function for ReflexAgent

In our evaluation function, we consider the final score as "food score" - "ghost score" - the amount of food left, where "food score" = 1.0 / closestFood (the distance between Pac-Man and the closest food), "ghost score" = 4.0 / closestGhost (the distance between Pac-Man and the closest ghost). Hence, there are totally three factors that are able to impact the final score: closestFood, closestGhost, and the amount of food left. It is obvious that the most important factor should be closestGhost, so that it has a larger weight (4.0). This allows Pac-Man to keep alive. Even when the closest ghost is on the closest food, Pac-Man will not touch that food. However, when closestGhost is equal to the safe distance (3) or bigger, it means even the closest ghost can hardly reach Pac-Man so that we just ignore the "ghost score" in this case. The other two factors closestFood and the amount of food left encourage Pac-Man to eat all the food left and guarantees it always goes to the closest food first when the other factors are the same.

Our evaluation function considers both food locations and ghost locations to perform well. Test results show that the agent can easily and reliably clear the testClassic, openClassic, and mediumClassic layout.

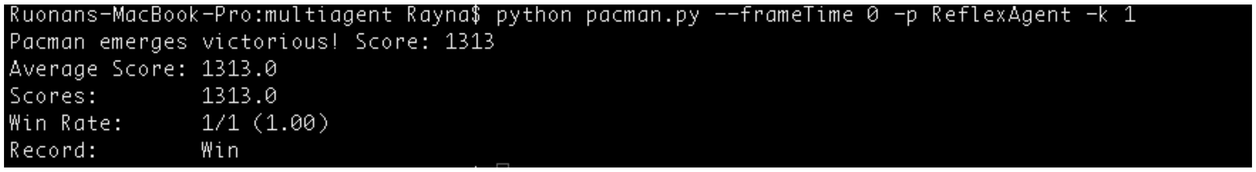
<u>Test Results</u>

1. testClassic



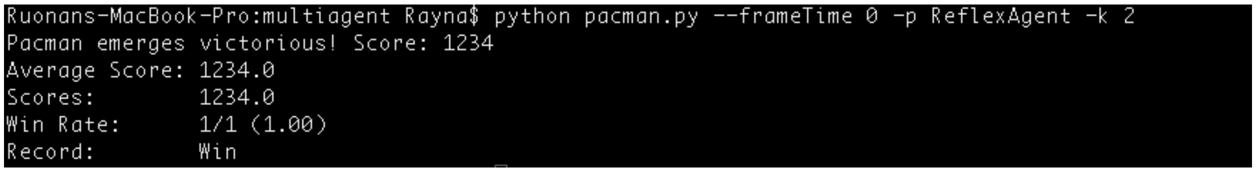
```
Ruonans-MacBook-Pro:multiagent Rayna$ python pacman.py -p ReflexAgent -l testClassic
Pacman emerges victorious! Score: 555
Average Score: 555.0
Scores:        555.0
Win Rate:      1/1 (1.00)
Record:        Win
```

2. mediumClassic with one ghost



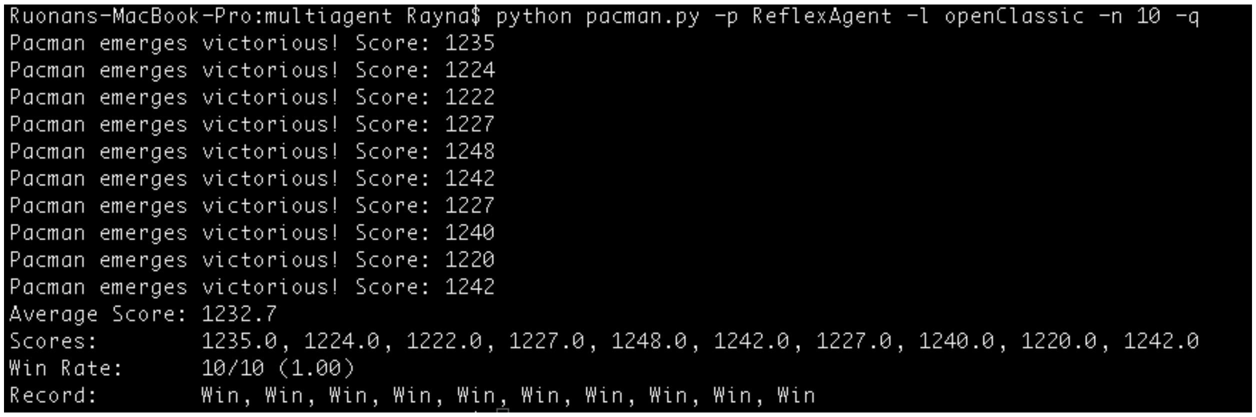
```
Ruonans-MacBook-Pro:multiagent Rayna$ python pacman.py --frameTime 0 -p ReflexAgent -k 1
Pacman emerges victorious! Score: 1313
Average Score: 1313.0
Scores:        1313.0
Win Rate:      1/1 (1.00)
Record:        Win
```

3. mediumClassic with two ghosts



```
Ruonans-MacBook-Pro:multiagent Rayna$ python pacman.py --frameTime 0 -p ReflexAgent -k 2
Pacman emerges victorious! Score: 1234
Average Score: 1234.0
Scores:        1234.0
Win Rate:      1/1 (1.00)
Record:        Win
```

4. openClassic



```
Ruonans-MacBook-Pro:multiagent Rayna$ python pacman.py -p ReflexAgent -l openClassic -n 10 -q
Pacman emerges victorious! Score: 1235
Pacman emerges victorious! Score: 1224
Pacman emerges victorious! Score: 1222
Pacman emerges victorious! Score: 1227
Pacman emerges victorious! Score: 1248
Pacman emerges victorious! Score: 1242
Pacman emerges victorious! Score: 1227
Pacman emerges victorious! Score: 1240
Pacman emerges victorious! Score: 1220
Pacman emerges victorious! Score: 1242
Average Score: 1232.7
Scores:        1235.0, 1224.0, 1222.0, 1227.0, 1248.0, 1242.0, 1227.0, 1240.0, 1220.0, 1242.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

Question 2 — MinimaxAgent

We implemented the minimax algorithm in this problem, where the Pac-Man is the max player and the ghosts are the min players. Pac-Man moves first and they take turns to move until the game is over.

We tested the MinimaxAgent for different depths in minimaxClassic and can obviously observe that when depth = 3 or 4, it takes a long time for running 1000 times of the game. Furthermore, Pac-Man often wins over 665/1000 games for us and it tends to get best results when depth = 2.

For the trappedClassic, Pac-Man will definitely die when depth is greater than 1. When Pac-Man believes that its death is unavoidable, it will try to end the game as soon as possible because of the constant penalty for living. Sometimes, this is the wrong thing to do with random ghosts, but minimax agents always assume the worst.

Test Results

1. minimaxClassic
depth = 1:
Average Score: 120.618
Win Rate: 611/1000 (0.61)
depth = 2:
Average Score: 204.054
Win Rate: 691/1000 (0.69)
depth = 3:
Average Score: -130.446
Win Rate: 362/1000 (0.36)
depth = 4:
Average Score: 193.192
Win Rate: 681/1000 (0.68)

2. trappedClassic
depth = 1:
Average Score: 36.714
Win Rate: 521/1000 (0.52)
depth = 2:
Average Score: -501.0
Win Rate: 0/1000 (0.00)
depth = 3:
Average Score: -501.0
Win Rate: 0/1000 (0.00)
depth = 4:
Average Score: -501.0
Win Rate: 0/1000 (0.00)

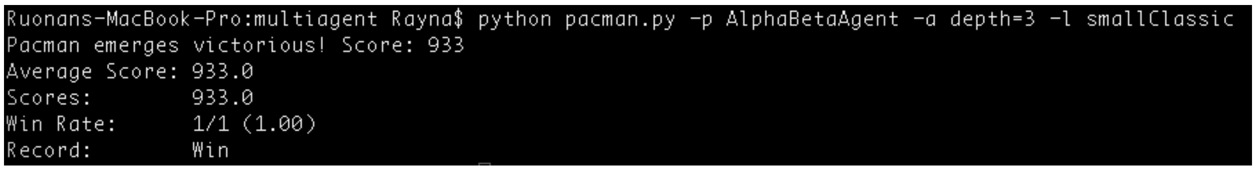
Question 3 — Alpha-Beta Pruning

The alpha-beta pruning makes exploring more efficient by not examining all branches of the minimax tree. It returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision. The general principle is this. Consider a node *n* somewhere in the tree, such that the player has a choice of moving to that node. If the player has a better choice m either at the parent node of n, or at any choice point further up, then n will never be reached in actual play. So once we have found out enough about *n* to reach this conclusion, we can prune it.

Test results show that the AlphaBetaAgent minimax values are the same with MinimaxAgent but runs much faster.

Test Results

1. smallClassic



```
Ruonans-MacBook-Pro:multiagent Rayna$ python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
Pacman emerges victorious! Score: 933
Average Score: 933.0
Scores:        933.0
Win Rate:      1/1 (1.00)
Record:        Win
```