Assignment 2 - COSE361 (03)

2021320117 Bae Minseong

Please check the comments of multiAgents.py for the code explanation.

I. Result of autograder.py

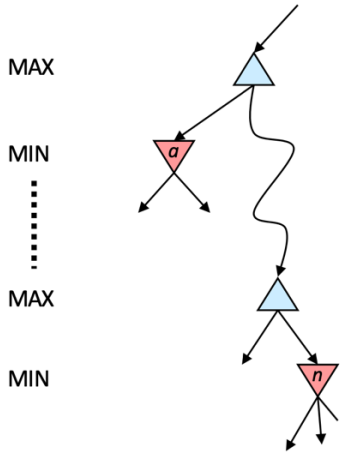
```
(base) ➜  multiagent python3 autograder.py
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 4-18 at 16:08:04

Question q1
===========

Pacman emerges victorious! Score: 1074
Pacman emerges victorious! Score: 1080
Pacman emerges victorious! Score: 1094
Pacman emerges victorious! Score: 1096
Pacman emerges victorious! Score: 1097
Pacman emerges victorious! Score: 1084
Pacman emerges victorious! Score: 1020
Pacman emerges victorious! Score: 1054
Pacman emerges victorious! Score: 1076
Pacman emerges victorious! Score: 1076
Average Score: 1075.1
Scores:        1074.0, 1080.0, 1094.0, 1096.0, 1097.0, 1084.0, 1020.0, 1054.0, 1076.0, 1076.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/q1/grade-agent.test (4 of 4 points)
***     1075.1 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (0 of 0 points)
***         Grading scheme:
***          < 10:  fail
***         >= 10:  0 points
***     10 wins (2 of 2 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  0 points
***         >= 5:  1 points
***         >= 10:  2 points

### Question q1: 4/4 ###

Question q2
===========

*** PASS: test_cases/q2/0-eval-function-lose-states-1.test
*** PASS: test_cases/q2/0-eval-function-lose-states-2.test
*** PASS: test_cases/q2/0-eval-function-win-states-1.test
*** PASS: test_cases/q2/0-eval-function-win-states-2.test
*** PASS: test_cases/q2/0-lecture-6-tree.test
*** PASS: test_cases/q2/0-small-tree.test
*** PASS: test_cases/q2/1-1-minmax.test
*** PASS: test_cases/q2/1-2-minmax.test
*** PASS: test_cases/q2/1-3-minmax.test
*** PASS: test_cases/q2/1-4-minmax.test
*** PASS: test_cases/q2/1-5-minmax.test
*** PASS: test_cases/q2/1-6-minmax.test
*** PASS: test_cases/q2/1-7-minmax.test
*** PASS: test_cases/q2/1-8-minmax.test
*** PASS: test_cases/q2/2-1a-vary-depth.test
*** PASS: test_cases/q2/2-1b-vary-depth.test
*** PASS: test_cases/q2/2-2a-vary-depth.test
*** PASS: test_cases/q2/2-2b-vary-depth.test
*** PASS: test_cases/q2/2-3a-vary-depth.test
*** PASS: test_cases/q2/2-3b-vary-depth.test
*** PASS: test_cases/q2/2-4a-vary-depth.test
*** PASS: test_cases/q2/2-4b-vary-depth.test
*** PASS: test_cases/q2/2-one-ghost-3level.test
*** PASS: test_cases/q2/3-one-ghost-4level.test
*** PASS: test_cases/q2/4-two-ghosts-3level.test
*** PASS: test_cases/q2/5-two-ghosts-4level.test
*** PASS: test_cases/q2/6-tied-root.test
*** PASS: test_cases/q2/7-1a-check-depth-one-ghost.test
*** PASS: test_cases/q2/7-1b-check-depth-one-ghost.test
*** PASS: test_cases/q2/7-1c-check-depth-one-ghost.test
*** PASS: test_cases/q2/7-2a-check-depth-two-ghosts.test
*** PASS: test_cases/q2/7-2b-check-depth-two-ghosts.test
*** PASS: test_cases/q2/7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running MinimaxAgent on smallClassic after 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/q2/8-pacman-game.test

### Question q2: 5/5 ###
```

```
Question q3
===========

*** PASS: test_cases/q3/0-eval-function-lose-states-1.test
*** PASS: test_cases/q3/0-eval-function-lose-states-2.test
*** PASS: test_cases/q3/0-eval-function-win-states-1.test
*** PASS: test_cases/q3/0-eval-function-win-states-2.test
*** PASS: test_cases/q3/0-lecture-6-tree.test
*** PASS: test_cases/q3/0-small-tree.test
*** PASS: test_cases/q3/1-1-minmax.test
*** PASS: test_cases/q3/1-2-minmax.test
*** PASS: test_cases/q3/1-3-minmax.test
*** PASS: test_cases/q3/1-4-minmax.test
*** PASS: test_cases/q3/1-5-minmax.test
*** PASS: test_cases/q3/1-6-minmax.test
*** PASS: test_cases/q3/1-7-minmax.test
*** PASS: test_cases/q3/1-8-minmax.test
*** PASS: test_cases/q3/2-1a-vary-depth.test
*** PASS: test_cases/q3/2-1b-vary-depth.test
*** PASS: test_cases/q3/2-2a-vary-depth.test
*** PASS: test_cases/q3/2-2b-vary-depth.test
*** PASS: test_cases/q3/2-3a-vary-depth.test
*** PASS: test_cases/q3/2-3b-vary-depth.test
*** PASS: test_cases/q3/2-4a-vary-depth.test
*** PASS: test_cases/q3/2-4b-vary-depth.test
*** PASS: test_cases/q3/2-one-ghost-3level.test
*** PASS: test_cases/q3/3-one-ghost-4level.test
*** PASS: test_cases/q3/4-two-ghosts-3level.test
*** PASS: test_cases/q3/5-two-ghosts-4level.test
*** PASS: test_cases/q3/6-tied-root.test
*** PASS: test_cases/q3/7-1a-check-depth-one-ghost.test
*** PASS: test_cases/q3/7-1b-check-depth-one-ghost.test
*** PASS: test_cases/q3/7-1c-check-depth-one-ghost.test
*** PASS: test_cases/q3/7-2a-check-depth-two-ghosts.test
*** PASS: test_cases/q3/7-2b-check-depth-two-ghosts.test
*** PASS: test_cases/q3/7-2c-check-depth-two-ghosts.test
*** Running AlphaBetaAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running AlphaBetaAgent on smallClassic after 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/q3/8-pacman-game.test

### Question q3: 5/5 ###

Question q4
===========

*** Method not implemented: getAction at line 291 of multiAgents.py
*** FAIL: Terminated with a string exception.

### Question q4: 0/5 ###


Question q5
===========

*** Method not implemented: getAction at line 291 of multiAgents.py
*** FAIL: Terminated with a string exception.

### Question q5: 0/6 ###


Finished at 16:08:10

Provisional grades
==================
Question q1: 4/4
Question q2: 5/5
Question q3: 5/5
Question q4: 0/5
Question q5: 0/6
------------------
Total: 14/25
```

II. Discussions

Q1. How can alpha-beta pruning make it more efficient to explore the minimax tree?

alpha-beta pruning을 이용하면 minimax tree에서 방문해야하는 노드의 개수를 줄일 수 있다.
다음과 같은 예시 상황을 생각해보면,



(자료 출처 : Lec5 강의 자료)

현재 n이라고 표기되어 있는 MIN 노드에 대해 탐색을 진행하고 있다고 하자.
가장 상단에 위치하고 있는 MAX 노드는 자신의 MIN 노드 successor들 중 가장 value가 높은 노드를 찾으려고 할 것이다. 그렇기 때문에 n의 successor들의 value를 계산하고 비교하는 과정에서 이미 탐색된 상태인 a라는 노드의 value가 어떤 n의 successor의 value보다 높다면, n의 나머지 다른 successor들의 value 값은 최상단 MAX 노드의 선택에 아무런 영향을 주지 않을 것이다. (결국 a가 선택될 것이기 때문에 나머지 값을 굳이 n에서 비교할 필요가 없다.) 그렇다면 우리는 n 노드의 successor들에 대한 탐색을 종료하고 해당 과정에서 얻은 value 값을 그냥 n의 value라고 판단해도 최종 결과에는 아무 영향이 없다.
이 a라는 값을 alpha-beta pruning에서는 alpha로 설정하고 계속 업데이트하며 지속적으로 탐색 과정에서 고려하는 것이며, MIN에 대한 beta-pruning도 비슷한 방식으로 진행될 것이다.
이렇게 되면, 우리는 minimax search tree에서 모든 노드를 방문할 필요가 없으며, minimax search보다 효율적인 탐색을 진행할 수 있다.

Q2. Is there a situation where the Reflex agent performs better than the minimax or alpha-beta pruning algorithm?

Minimax algorithm은 player agent가 취할 action에 따라 opponent agent가 취할 action을 고려한다. 즉, player agent는 opponent agent가 충분히 rational하여 player agent가 얻을 결과값을 최소화시키는 방향으로 행동할 것이라고 예측한다. 그렇기에 player (pacman)을 MAX, opponent (ghost)를 MIN agent로 설정하고, tree에서 각 노드들을 번갈아 가며 위치시켜 adversarial하게 search를 진행한다.
하지만 reflex agent는 ghost와의 거리, food의 유무와 남아있는 food들의 위치 등의 해당 state의 feature들을 고려하여 계산한 value만을 가지고 해당 value가 최대가 되는 action을 선택한다. 즉, 자신의 action에 따라 ghost가 어떤 action을 취할 것인지는 크게 고려하지 않는다. (단지 ghost에서 멀어지는 것이 player에게 이득이기 때문에 ghost와의 거리에 따라 value 값을 변화시키는 것이다.)
다음과 같은 예시 상황을 생각해보자.

```
(base) ➜  multiagent python pacman.py -p MinimaxAgent -l trappedClassic -a depth
=3
Pacman died! Score: -501
Average Score: -501.0
Scores:        -501.0
Win Rate:      0/1 (0.00)
Record:        Loss
```

미리 제공되는 코드에 포함되어 있는 testcase들 중 trappedClassic이라는 maze가 존재한다.
해당 maze의 경우 아래와 같은 모양으로 생긴 것을 볼 수 있는데, trappedClassic에 대해 test를 여러 번 진행해보면 blue ghost의 경우 위 방향으로 움직이는 경우도 존재하고, 아래 방향으로 움직이는 경우도 존재한다. 만약 ghost가 충분히 rational해 pacman을 반드시 죽이는 방향으로만 움직인다면, ghost는 반드시 pacman이 초기에 존재하는 방향인 위로만 움직일 것인데, blue ghost는 위/아래로 random하게 action을 취한다.
하지만, 그러한 blue ghost의 action과 무관하게 pacman은 minimax algorithm을 사용할 경우 반드시 orange ghost에게 부딪혀서 스스로 죽는다. 그 이유는 minimax algorithm의 경우 해당 상황에서 blue ghost는 반드시 value를 최소화시키기 위해서 자신이 있는 방향으로 움직일 것이라고 생각할 것이고 food를 먹지 않고 계속 살아 있는 것은 pacman rule에 따라 score 측면에서는 더 손해이기 때문에 죽는 것이 value를 maximizing하는 action이라고 판단하기 때문이다. 아래 그림과 같은 경우도 blue ghost가 아래로 움직였음에도 불구하고 pacman은 food가 있는 아래로 움직이지 않고 orange ghost에 부딪혀서 스스로 죽어버린다.

하지만 아래의 실행 결과를 보면 같은 testcase에 대해 ReflexAgent의 경우 당연히 blue ghost가 위로 움직일 때는 pacman이 죽겠지만 blue ghost가 아래로 움직인다면 pacman은 blue ghost를 따라 아래로 움직이면서 모든 food를 먹는데 성공한다.
즉, ghost가 random하게 action을 취하는 특정한 상황들의 경우, ReflexAgent가 더 좋은 perfomance를 보여준다고 볼 수 있다.

```
(base) ➜  multiagent python pacman.py -p ReflexAgent -l trappedClassic
Pacman died! Score: -502
Average Score: -502.0
Scores:        -502.0
Win Rate:      0/1 (0.00)
Record:        Loss
(base) ➜  multiagent python pacman.py -p ReflexAgent -l trappedClassic
Pacman emerges victorious! Score: 532
Average Score: 532.0
Scores:        532.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Q3. Ask yourself one question and answer.

Question: MiniMaxAgent나 AlphaBetaAgent에서 depth limit이 반드시 필요한가? 필요하다면 그 이유는 무엇일까?

Answer:

Pacman 뿐만 아니라 다양한 실제 게임에서는 각 agent들의 action에 따라 무수히 많은 경우의 수가 나올 수 밖에 없고, 해당 모든 경우의 수를 탐색하는 것은 현실적으로 매우 어렵고, 많은 cost를 소비하게 될 수 밖에 없다. 그렇기 때문에 탐색을 진행할 최대 깊이를 설정하여 해당 깊이 아래로는 탐색을 진행하지 않고, 최대 깊이에 도달할 경우 evaluation function을 활용하여 terminal utility 값을 return할 경우 우리는 모든 문제 상황에 대해서 답을 유한 시간 내에 찾을 수 있음을 반드시 보장할 수 있다. 하지만 최대 depth를 설정하는 것은 일종의 trade-off로 볼 수 있다. 최대 깊이를 설정하고 evaluation function을 활용하여 노드의 value 값을 유추하는 것은 evaluation function의 imperfectness에 의해 일종의 근사이기 때문에, 최대 깊이를 더 깊게 설정할 수록 evaluation function의 정확성에 의한 영향은 더 작아지겠지만, 그만큼 알고리즘을 수행하기 위한 time complexity와 cost는 커질 수 밖에 없다.