

## 1. 각 알고리즘 별 구현 방법

### 1) Minimax Agent

Minimax Agent는 Minimax Algorithm(최소최대 알고리즘)을 통해 구현된다. Minimax Algorithm은 최소최대 결정을 계산하는 알고리즘이다. 이 결정은 최선의 동작을 return한다. 최선의 동작은 상대의 이익을 최소화하고, 자신의 이익이 최대가 되는 동작이다. 구현한 알고리즘은 재귀를 통해 트리의 말단까지 갔다가, 트리를 타고 최소최대 값들을 계산해 올라간다. 그 과정에서 ghost의 value는 최소가 되는 값을 선택하고, human은 선별된 ghost의 값들 중 가장 큰 값을 반환한다. 코드 구현은 다음과 같다.

Minimax Agent에서 Max\_Value함수와 Min\_Value함수가 필요하다. Max\_Value함수는 human이 Min\_Value 값들 중 가장 최대가 되는 값들을 선택해 나가는 역할을 한다. Min\_Value함수는 ghost들의 이동에 의한 점수가 최소가 되는 값을 반환하는 역할을 한다.

Max\_Value함수는 다음 과정을 통해 구현하였다. (1) 종료 조건을 판단하는 terminal test를 한다. (2) high\_value값을  $-\infty$ 로 초기화하고, actions에 human이 할 수 있는 동작들을 저장한다. (3) 반복문을 통해 각각의 action에 접근한다. Action이 수행되었을 때 ghost들을 통해 나올 수 있는 value 중 가장 작은 값을 value에 저장한다. 이에 Min\_Value함수가 사용된다. (4) depth가 0이면 나온 결과 중 가장 큰 value가 되는 action을 수행하고, depth가 0이 아닌 경우 그때까지 구한 value 중 가장 큰 값을 반환한다.

Min\_Value함수는 다음 과정을 통해 구현하였다. (1) 종료 조건을 판단하는 terminal test를 한다. (2) min\_value값을  $\infty$ 로 초기화하고, actions에 ghost가 할 수 있는 동작들을 저장한다. (3) 반복문을 통해 각각의 action에 접근한다. 이때 ghost가 그 step에서 마지막 순서인지 확인한다. 마지막 순서가 아니라면 Min\_Value함수를 통해 그 다음 ghost의 value도 계산한다. Ghost가 마지막 순서라면 depth를 고려한다. 마지막 depth면 AdversarialSearchAgent에서 선언한 evaluationFunction을 통해 Score를 평가한다. 아직 depth가 더 남아 있으면 Max\_Value 함수를 불러와 그 다음 depth도 수행한다. (4) 나온 값들 중 가장 작은 값을 반환한다.

### 2) AlphaBeta Pruning Agent

AlphaBeta Pruning은 Minimax algorithm을 수행하면서 더 이상 탐색을 할 필요 없는 불필요한 부분을 제거해 탐색하는 알고리즘이다. 이 알고리즘은 가지치기를 통해 기존 Minimax보다 더 빠른 탐색을 보여준다. 기존 Minimax Algorithm에 a,b를 추가하여 구현한다. a,b는 경로에 있는 임의의 선택 지점에서 지금까지 발견한 최선의 선택이다. 그중 a는 가장 큰 값이고, b는 가장 작은 값

이다. 이를 통해 현재 노드의 value와 서로 비교하고, 안 좋은 값을 가진 노드는 잘라낸다.

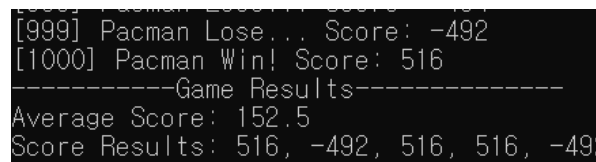
### 3) Expectimax Agent

기존 Minimax와 AlphaBeta Pruning은 ghost가 optimal하다는 가정 하에 최악의 경우 얻을 수 있는 가장 최선의 선택을 했었다. 하지만 실제 ghost는 optimal하게 행동하지 않는다. 따라서 Expectimax algorithm은 기댓값을 계산하여 선택한다. 기댓값을 계산하는 과정에서 모든 경우를 판단해야 한다. 구현한 코드에서는 Max에서 선택할 때 Min에서 나온 값의 평균값을 바탕으로 선택했다. Min\_Value 과정에서 평균 value를 구해야 하므로 취할 수 있는 action의 수만큼 나누어 주어 value의 평균값을 구했다.

결과적으로 MiniMax algorithm을 기반으로 하여 AlphaBeta Pruning과 Expectimax는 약간의 추가 사항을 더해 구현하였다.

## 2. 실행 화면

1) python pacman.py -p MinimaxAgent -m minimaxmap -a depth=4 -n 1000 -q

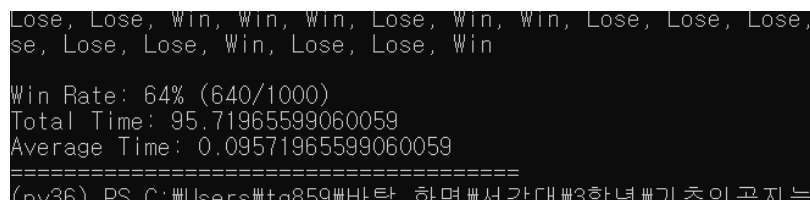


```

[999] Pacman Lose... Score: -492
[1000] Pacman Win! Score: 516
-----Game Results-----
Average Score: 152.5
Score Results: 516, -492, 516, 516, -492

```

Figure 1 MinimaxAgent(depth=4 for minimaxmap) 실행 초기



```

Lose, Lose, Win, Win, Win, Lose, Win, Win, Lose, Lose, Lose,
se, Lose, Lose, Win, Lose, Lose, Win
Win Rate: 64% (640/1000)
Total Time: 95.71965599060059
Average Time: 0.09571965599060059
=====
(py36) PS C:\Users\wtg859\Hunt 하먼서강대3학년부기초의공지는

```

Figure 2 MinimaxAgent(depth=4 for minimaxmap) 실행 결과

1)번 코드를 실행하면 MinimaxAgent로 구현된 깊이 4의 Pacman 총 1000번 실행된다. 실행 초기 Pacman 게임의 결과가 점수와 함께 1000번 출력되고, 평균 점수와 각각의 점수가 출력된다. 이후 총 승률, 총 걸린 시간, 평균 실행 시간이 나온다.

실행 결과 승률은 64%로 주어진 범위 50%~70% 내에 있다.

2) python time\_check.py

```
Win Rate: 18% (55/300)
Total Time: 510.5132873058319
Average Time: 1.7017109576861065
=====
----- END Minimax (depth=3) For Medium Map
```

**Figure 3 time\_check.py 결과 Minimax(depth=3) For Medium Map**

```
Win Rate: 15% (45/300)
Total Time: 488.06469345092773
Average Time: 1.6268823115030924
=====
----- END AlphaBeta (depth=3) For Medium Map
```

**Figure 4 time\_check.py 결과 AlphaBeta(depth=3) For Medium Map**

```
Win Rate: 36% (368/1000)
Total Time: 70.43686318397522
Average Time: 0.07043686318397523
=====
----- END Minimax (depth=4) For Minimax Map
```

**Figure 5 time\_check.py 결과 Minimax(depth=4) For Minimax Map**

```
Win Rate: 36% (367/1000)
Total Time: 58.266621828079224
Average Time: 0.05826662182807922
=====
----- END AlphaBeta (depth=4) For Minimax Map
```

**Figure 6 time\_check.py 결과 AlphaBeta(depth=4) For Minimax Map**

time\_check.py 결과 Minimax보다 AlphaBeta의 시간이 더 적게 걸리는 것을 확인할 수 있다. 이를 통해 AlphaBeta Agent가 Minimax Agent보다 더 효율적인 것을 알 수 있다.

[illegible]

ExpectimaxAgent 100번 실행 결과 승률 53%가 나왔다. 이는 50% 내외의 승률을 잘 보여주고 있다. 또한 이긴 경우 532를 출력하고, 진 경우 -502를 출력하고 있는 것을 볼 수 있다.