과목 명: 기초인공지능

담당 교수 명: 양 지 훈

<<AI_assignment02>>

서강대학교 컴퓨터학과

[학번] 20171700

[이름] 최재원

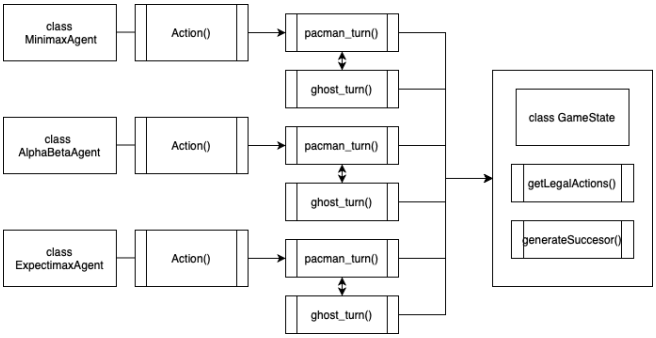
목 차

1. 프로그램 개요

본 과제에서는 PCAMAN Game 에 대해서 서로 다른 알고리즘을 적용하여 그 결과를 확인한다. Minimax search, Alpha-Beta search 그리고 Stochastic(expectimax) 총 3 개의 알고리즘을 구현하여 다양한 종류의 map 에서 알고리즘마다의 게임 결과를 확인하고, Minimax 와 Aplha-Beta 알고리즘의 유의미한 시간 차이와 알고리즘마다의 승률 등을 통해 각 성능을 분석한다.

2. 구현 방법

2.1 Flow chart



대략적인 flow chart 는 위와 같고, 아래 모든 알고리즘에서 evaluation function 과 depth 는 AdversialsearchAgent class 의 그것을 사용한다. 각각의 알고리즘마다 존재하는 Action 함수 내에서 모든 것을 구현하였고, evaluation, depth_limit, total_agent 의 사용은 동일하다.

2.2 Minimax Agent

Minimax 알고리즘에서 Max 는 pacman 에 해당하고, Min 은 ghost 에 해당한다. 우선 total_agent 는 게임에서 pacman 과 ghost 수의 총합을 의미하고, 이는 minimax search algorithm 의 depth 1 에 해당하는 agent 수이다. 즉, 모든 agent 가 움직여야 depth 가 1 인 것이다.

Action() 함수 내부에는 terminal_sate(), pacaman_turn () 그리고 ghost_turn()의 총 3 개의 함수가 존재하고 Action() 함수는 pacman_turn(1,gameState,0)을 return 한다. 이는 tree 의 구성을 Max 인 pacman 부터 시작함을 의미하고, 최종적으로 재귀 함수의 탈출 조건에 부합하여 재귀가 멈추고 생성된 tree 를 다시 거슬러 올라오며 min 과 max 에 해당하는 결정을 하고 다시 pacman_turn 에 depth == 1 이 되면 본 함수에서 거슬러 올라온 결과를 토대로 결정된 action 을 return 하며, 이것이 Action()함수의 return 값이라고 할 수 있다. 강의자료와 비교했을 때 Pacman_turn() 함수가 Max-value 함수에 해당하고, ghost_turn() 함수가 Min-value 함수에 해당하며, Minimax- decision 함수의 기능은 pacman_turn()함수에서 해준다고 볼 수 있다.

```
def Action(self, gameState):
    ###################### Write Your Code Here ############################
    total_agent = gameState.getNumAgents()
    evaluation = self.evaluationFunction
    depth_limit = self.depth
    def terminal_state(S,D):
        if S.isWin() or S.isLose():
            return True
        else:
            return False
    def pacman_turn(depth,current,agent):
        if not terminal_state(current, depth):
            moving = current.getLegalActions(agent)
            candidate = []
            for move in moving:
                following = current.generateSuccessor(agent,move)
                heappush(candidate, [(-1) * ghost_turn(depth, following, agent + 1), move])
            if depth == 1:
                return heappop(candidate)[1]
            return heappop(candidate)[0]*(-1)
        else:
            return evaluation(current)

    def ghost_turn(depth,current,agent):
        if not terminal_state(current, depth):
            moving = current.getLegalActions(agent)
            candidate = []
            for move in moving:
                following = current.generateSuccessor(agent, move)
                if agent + 1 == total_agent:
                    if depth < depth_limit:
                        heappush(candidate,[pacman_turn(depth+1,following,0),move])
                    else:
                        heappush(candidate, [evaluation(following), move])
                else:
                    heappush(candidate,[ghost_turn(depth, following, agent + 1),move])
            return heappop(candidate)[0]

        else:
            return evaluation(current)
```

전체적인 구성은 다음과 같다. 함수의 시작에서 terminal_state()함수를 사용하여 terminal test 를 진행하며, true 를 return 받은 경우 evaluation 함수를 호출하고 그 값을 return 한다. False 를 받은 경우 terminal state 가 아닌 것이고, 다음 과정을 진행한다. Pacman.py 에 존재하는 GameState class function 인 getLegalActions()로부터 pacman 과 agent 는 움직일 수 있는 다음 행위들의 list 를 각각의 rule 에 따라서 받아와 moving 변수에 저장하고 moving list 에 저장된 하나의 action 을 사용하여 game 전체의 다음 state 를 GameState class function 인 generateSuccessor()로부터 받아와 following 변수에 저장한다. 여기까지는 pacman_turn 과 ghost_turn 이 동일하며 둘의 차이점은 서로를 호출한 결과값에서 max 를 return 하는지와 min 을 return 하는 지이다.

우선 pacman_turn 함수에서 successive state 에 따른 ghost_turn 의 값을 heap 자료구조인 candidate 에 저장을 한다. 이 때 max-heap 으로 사용하기 위하여 ghost_turn 의 return 값에 (-1)을 곱해준 값을 넣어주었다. Depth 가 1 이 아닌경우, 즉 tree 가 형성되고 있는 경우에는 max-heap 으로부터 구한 ghost_turn 의 return 값(min) 중 최대값을 return 하게 된다.

다음으로 ghost_turn 함수에서는 현재 depth 의 agent 수에 따라 경우를 나누었는데, 이는 앞서 설명했듯이 해당 tree 의 depth 는 모든 agent 가 1 번씩 움직인 상황을 전제로 하기 때문이다. 따라서 agent + 1 과 total_agent 의 수가 같아진 경우가 tree 의 한 depth 에 해당하는 case 를 모두 계산한 경우인 것이고(agent = 0 인 경우가 pacman 인 경우이고, 0 보다 큰 경우 ghost 인 경우인데, 이는 agent 변수를 index 로 사용하는 generateScuccesor() 및 getLegalActions()함수를 호출하기 때문이다), 따라서 depth+1 을 하여 pacman_turn(max-case)을 호출한다. 이 때 depth 가 초기에 설정한 limit_depth 인 경우 pacman_turn 함수로부터 그 값을 받아오는 것이 아닌, evaluation 으로부터 score 를 받아오게 된다. 아직 모든 agent 를 호출한 경우가 아닐때는 ghost_turn 함수를 agent + 1 을 해주어 재귀호출한다.

Pacman_turn 함수에서와 마찬가지로 heap 자료구조를 사용하지만, 여기선 min-heap 으로 사용한다.

최종적으로 위에 설명한 바와 같이 재귀를 통하여 tree 를 구성하고, depth 와 win,, lose 에 따른 terminal state 에 도달하면 tree 생성을 중단하고 다시 depth == 1 의 pacman_turn 으로 돌아간다. 그리고 거슬러 올라가면서 선택된 case 를 토대로 pacman_turn 에서 최종선택을 하게되며, 해당 action 이 minimax 알고리즘을 통해서 결정된 pacman 의 다음 움직임이 된다. 이 때 동률에 대한 처리는 heap 에 먼저 넣은

action 의 경우가 선택되며, 랜덤성을 부여할 수 있지만, 랜덤성을 부여하는 행동이 승률을 더 높인다는 보장이 없기 때문에 heap 에 먼저 들어간 action 이 선택되도록 유지하였다.

2.3 AlphaBeta Pruning Agent

```
def pacman_turn(depth,current,agent,a,b):
  if not terminal_state(current, depth):
    candidate = []
    v = float("-inf")
    for move in current.getLegalActions(agent):
      following = current.generateSuccessor(agent,move)
      value = ghost_turn(depth, following, agent + 1,a,b)
      heappush(candidate, [(-1) * value, move])
      if value > v:
        a = max(a, value)
        v = value
      if v > b: ##pruning
        return heappop(candidate)[0]*(-1)

    if depth == 1:
      return heappop(candidate)[1]
    return heappop(candidate)[0]*(-1)
  else:
    return evaluation(current)
```

```
def ghost_turn(depth,current,agent,a,b):
  if not terminal_state(current, depth):
    candidate = []
    v = float("inf")
    for move in current.getLegalActions(agent):
      following = current.generateSuccessor(agent, move)
      if agent + 1 == total_agent:
        if depth < depth_limit:
          value = pacman_turn(depth+1,following,0,a,b)
          heappush(candidate,[value,move])
        else:
          value = evaluation(following)
          heappush(candidate, [value, move])
      else:
        value = ghost_turn(depth, following, agent + 1,a,b)
        heappush(candidate,[value,move])
      if value < v:
        b = min(b, value)
        v = value
      if v < a: ##pruning
        return heappop(candidate)[0]
    return heappop(candidate)[0]

  else:
    return evaluation(current)
```

AlphaBeta Pruning 은 위의 minimax search 를 토대로 성능을 개선 시킨 알고리즘이다. 해당 알고리즘에 추가된 것은 누적 최대값과 누적 최소값에 해당하는 알파와 베타 변수이다. 이들은 초기에 각각 음의 무한대값, 양의 무한대값으로 설정되어 있으며, search 를 진행함에 따라서 그 값이 update 된다.

먼저 pacman_turn 함수에서는 새로운 action 에 대한 next state 에 따른 값들을 순회하는 for 문 안에서 현재의 beta 에 해당하는 변수 b 값보다 현재 선택된 최대값이 크다면, for 문을 중단하고 그 값을 return 하게 된다. 이는 pacman_turn 함수의 return 값을 토대로 minimum 값을 취하는 ghost_turn 함수에 대하여, 현재 선택된 최소값 b 보다 더 큰 값이 pacman_turn()함수에서 나왔을 경우 pacman_turn()함수에서는 최대값을 구하기 때문에 더이상 그보다 작은 값이 나올수 없으므로 다른 state case 를 무시하고 건너뛰는 것이다. 이 때 alpha 값에 해당하는 a 변수 값은 현재까지 search 한 node 들 중 최대 값을 pacman_turn 함수에서 저장한다.

이와 반대로 ghost_turn 함수에서는 for 문안에서 현재 최대값인 a 보다 더 작은 값이 나올 경우, for 문을 멈추고 값을 return 한다. 이유는 위와 비슷하게, ghost_turn 함수에서 return 되는 값들은 pacman_turn 함수에서 최대값을 선택하는데 사용되는데, 이때 최소값을 구하는 ghost_turn 함수에서 이미 현재 최대값인 a 보다 작은 값이 나온다면, 더이상 다른 state 들을 고려할 필요가 없어지기 때문이다.

물론 agent 수가 아직 채워지지 않은 경우는 ghost_turn 값들을 보고, terminal state 인 depth 인 경우 evaluation 함수 값을 보게된다.

위와 같은 동작으로 pruning 을 수행하는데 영향을 주는 요소로써 dfs 로 동작하는 위 알고리즘에서 어떤 branch 를 통해 node 를 먼저 살펴보는 지가 고려될 수 있다. 본 과제에서는 getLEgalActions 로부터 얻은 결과값의 순서대로 보게 되도록 고정되어 있고, 이를 보는 순서를 임의로 바꿀 수는 있지만 이는 randomly 한 경향이 있으므로 기존의 순서대로 살펴본다. 해당 알고리즘은 결과적으로 minimax search 보다 약 절반 정도의 branch 를 덜 보게 되며, 시간 복잡도는 더 작아진다.

2.4 Expectimax Agent

```
def pacman_turn(depth, current, agent):
  if not terminal_state(current, depth):
    moving = current.getLegalActions(agent)
    candidate = []
    for move in moving:
      following = current.generateSuccessor(agent, move)
      heappush(candidate, [(-1) * ghost_turn(depth, following, agent + 1), move])
    if depth == 1:
      return heappop(candidate)[1]
    return heappop(candidate)[0] * (-1)
  else:
    return evaluation(current)

def ghost_turn(depth, current, agent):
  if not terminal_state(current, depth):
    moving = current.getLegalActions(agent)
    candidate = []
    exp = 1.0 / float(len(moving))
    value = 0
    for move in moving:
      following = current.generateSuccessor(agent, move)
      if agent + 1 == total_agent:
        if depth < depth_limit:
          value += pacman_turn(depth + 1, following, 0)
        else:
          value += evaluation(following)
      else:
        value += ghost_turn(depth, following, agent + 1)
    return float(value) / exp

  else:
    return evaluation(current)

return pacman_turn(1, gameState, 0)
```

Expectimax agent 는 Stochastic game search 알고리즘이며, minimax search 알고리즘을 개선시키기 위하여 tree 의 각 node case 에 대한 선택에 있어서 probability 를 추가한것이다.

코드로 봤을 때 minimax search 와 다른 부분은 ghost_turn 함수에 있다. Expectiminmax search 의 경우 최소선택과 최대 선택 모두에서 확률을 개입시키지만, Expectimax search 에서는 max 에서 min_value 에 대해서 선택할 때, 즉 min 값에 대해서만 확률을 개입시킨다.

Minimax search 알고리즘에서는 pacman_turn 함수의 return 값들을 heap 자료구조에 저장하여 min-max 로써 최소값을 도출하고, 그 값을 return 하였다. 하지만 exepectimax agent 에서는 하나의 depth 를 채웠을 경우 pacman_turn 함수의 return 값들을 또는 아직 agent 수가 total 이 아닌경우 다음 ghost 에 대한 ghost_turn 값들을 또는 depth 가 terminal state 에 해당하였을 때 evaluation 함수의 return 값들을 누적해서 더하고, 해당 sum 값을 branch 의 수, 즉 현재 상태인 current 에서 가능한 next move 의 경우의 수인 getLegalActions 의 return 값 list 의 길이만큼을 나누어 준다. 이는 하나의 값을 선택하는 것이 아니라, 모든 case 들의 값들을 평균값으로 선택해가며 최종적으로 max 에 해당하는 pacman_turn 에서 배제되는 것 없이 전체 state 들을 고려하여 선택하겠다는 것이다. 이론적으로 봤을 때 앞서 구현하였던 minimax searh 나 alpha beta search 에 비해서 보다 정확한 선택을 할 것이라 추측할 수 있다.

3. 실행 결과

3.1 Minimax Agent

```
python3 pacman.py -p MinimaxAgent -m minimaxmap -a depth=4 -n 1000 -q
```

```
[987] Pacman Win! Score: 516
[988] Pacman Win! Score: 516
[989] Pacman Win! Score: 514
[990] Pacman Win! Score: 516
[991] Pacman Win! Score: 514
[992] Pacman Win! Score: 516
[993] Pacman Win! Score: 516
[994] Pacman Win! Score: 516
[995] Pacman Win! Score: 516
[996] Pacman Win! Score: 516
[997] Pacman Win! Score: 516
[998] Pacman Lose... Score: -494
[999] Pacman Win! Score: 516
[1000] Pacman Win! Score: 516
-----------Game Results-----------
Average Score: 174.957
```

```
 Win, Lose, Win, Win, Win, Lose, Win, W
Lose, Win, Win, Win, Win, Win, Lose, Lo
, Win, Lose, Win, Lose, Lose, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win,

Win Rate: 66% (662/1000)
Total Time: 60.15219569206238
Average Time: 0.06015219569206238
========================================
```

게임 결과, 승률이 50%-70% 사이인 66%가 나온것을 확인하였다. 해당 명령어를 수행할때마다 승률이 조금씩 바뀌었지만, 해당 범위내에 포함되었다.

3.2 AlphaBeta Pruning Agent

AlphaBeta Pruning 과 Minmax search 의 유의미한 차이점을 분석하기 위하여 time_check.py 파일을 실행시켜, smallmap, mediummap 에 대한 300 가지의 test case 를 실행시켰고, minmaxmap 에 대해서도 test 1000 개를 실행시켰다. 이 때 명확한 비교를 위해 depth 는 둘 다 동일하게 설정하였다. 전자의 300 개 test case 에 대해서는 depth 를 2 로 설정하였고, 후자의 1000 개 test case 에 대해서는 depth 를 4 로 설정하였다.

<small map - Minmax>

```
============================== START TIME CHECK ===
----- START PART  MiniMax (depth=2) For Small Map:

====================================
[1] Pacman Lose... Score: -280
[2] Pacman Lose... Score: -377
[3] Pacman Lose... Score: -3
[4] Pacman Lose... Score: -181
[5] Pacman Lose... Score: -319
[6] Pacman Lose... Score: -305
[7] Pacman Lose... Score: -343
[8] Pacman Lose... Score: -329
[9] Pacman Lose... Score: -380
[10] Pacman Lose... Score: -368
```

```
[285] Pacman Lose... Score: -132
[286] Pacman Lose... Score: -138
[287] Pacman Lose... Score: -107
[288] Pacman Lose... Score: 247
[289] Pacman Lose... Score: -154
[290] Pacman Lose... Score: -208
[291] Pacman Lose... Score: 67
[292] Pacman Lose... Score: -381
[293] Pacman Lose... Score: -373
[294] Pacman Lose... Score: -377
[295] Pacman Lose... Score: 64
[296] Pacman Lose... Score: -331
[297] Pacman Lose... Score: -368
[298] Pacman Lose... Score: -160
[299] Pacman Lose... Score: -317
[300] Pacman Lose... Score: -110
-----------Game Results-------------
Average Score: -165.46333333333334
```

```
Win Rate: 0% (0/300)
Total Time: 46.37375068664551
Average Time: 0.154579168955485
====================================
----- END  MiniMax (depth=2) For Small Map
```

결과 2-1. Minimax Agent – small map

<small map - AlphaBeta>

```
----- START PART  AlphaBeta (depth=2) For Small Map:

====================================
[1] Pacman Lose... Score: 72
[2] Pacman Lose... Score: -372
[3] Pacman Lose... Score: -319
[4] Pacman Lose... Score: -141
[5] Pacman Lose... Score: -367
[6] Pacman Lose... Score: -45
[7] Pacman Lose... Score: -356
[8] Pacman Lose... Score: 26
[9] Pacman Lose... Score: -128
[10] Pacman Lose... Score: -353
```

```
Win Rate: 0% (2/300)
Total Time: 39.972041606903076
Average Time: 0.1332401386896769
====================================
----- END  AlphaBeta (depth=2) For Small Map
```

```
[287] Pacman Lose... Score: 41
[288] Pacman Lose... Score: -376
[289] Pacman Lose... Score: 185
[290] Pacman Lose... Score: -169
[291] Pacman Lose... Score: -343
[292] Pacman Lose... Score: 281
[293] Pacman Lose... Score: -87
[294] Pacman Lose... Score: -133
[295] Pacman Lose... Score: -129
[296] Pacman Lose... Score: -380
[297] Pacman Lose... Score: -368
[298] Pacman Lose... Score: -379
[299] Pacman Lose... Score: 742
[300] Pacman Lose... Score: -137
----------Game Results----------
Average Score: -153.19
```

결과 2-1. AlphaBeta Agent – small map

위의 두 결과는 각각 small map 에 대한 minmax 와 alphabeta agent 의 결과이다. Depth == 2 로 설정하였기 때문에 승률은 두가지 경우 모두 매우 낮은것을 확인할 수 있다.

두 결과의 총 실행시간과 평균 실행시간을 비교해보면 다음과 같다. Minmax agent 는 총시간 약 46.37 초이고 평균 실행시간 약 0.155 초이다. 반면에 AlphaBeta agent 는 총시간 약 39.97 초이고, 평균 실행시간 약 0.133 초이다. Small map 에 대해서 AlphaBeta agent 가 수행시간이 더 단축된 것을 확인할 수 있었다.

<medium map - Minmax>

```
----- START PART  MiniMax (depth=2) For Medium Map:

====================================
[1] Pacman Lose... Score: -72
[2] Pacman Lose... Score: 155
[3] Pacman Lose... Score: -109
[4] Pacman Lose... Score: -25
[5] Pacman Lose... Score: -93
[6] Pacman Lose... Score: -29
[7] Pacman Lose... Score: 368
[8] Pacman Lose... Score: 404
[9] Pacman Lose... Score: -206
[10] Pacman Lose... Score: -317
```

```
[290] Pacman Lose... Score: 100
[291] Pacman Lose... Score: 67
[292] Pacman Lose... Score: -132
[293] Pacman Lose... Score: 236
[294] Pacman Lose... Score: 304
[295] Pacman Lose... Score: 256
[296] Pacman Lose... Score: 818
[297] Pacman Lose... Score: 202
[298] Pacman Lose... Score: 448
[299] Pacman Lose... Score: 278
[300] Pacman Lose... Score: 415
-----------Game Results-------------
Average Score: 87.36333333333333
```

```
Win Rate: 0% (1/300)
Total Time: 79.20858645439148
Average Time: 0.26402862151463824
====================================
----- END  MiniMax (depth=2) For Medium Map
```

결과 2-2. Minimax Agent – medium map

<medium map - AlphaBeta>

```
----- START PART  AlphaBeta (depth=2) For Medium Map:

====================================
[1] Pacman Lose... Score: 141
[2] Pacman Lose... Score: 137
[3] Pacman Lose... Score: -128
[4] Pacman Lose... Score: 137
[5] Pacman Lose... Score: 180
[6] Pacman Lose... Score: 141
[7] Pacman Lose... Score: -88
[8] Pacman Lose... Score: 163
[9] Pacman Lose... Score: -92
[10] Pacman Lose... Score: 46
```

```
[290] Pacman Lose... Score: -14
[291] Pacman Lose... Score: -318
[292] Pacman Lose... Score: 193
[293] Pacman Lose... Score: 92
[294] Pacman Lose... Score: -78
[295] Pacman Lose... Score: -135
[296] Pacman Lose... Score: 99
[297] Pacman Lose... Score: 112
[298] Pacman Lose... Score: 87
[299] Pacman Lose... Score: 422
[300] Pacman Lose... Score: 32
-----------Game Results-------------
Average Score: 70.17666666666666
```

```
Win Rate: 0% (1/300)
Total Time: 72.88695526123047
Average Time: 0.24295651753743489
====================================
----- END  AlphaBeta (depth=2) For Medium Map
```

결과 2-2. AlphaBeta Agent – medium map

위 두 결과는 medium map 에 대한 Minimax Agent 와 AlphaBeta Agent 의 실행 결과이다. Depth 는 동일하게 2 로 하였으므로 small map 에서와 비슷한 승률 양상을 띄는것을 확인할 수 있다.

시간을 비교하면, Minimax agent 는 총 시간 약 79.2 초, 평균 시간 약 0.264 초가 걸렸고, AlphaBeta agent 는 총 시간 약 72.8 초, 평균 시간 약 0.249 초가 걸렸다. Medium map 에 대해서도 AlphaBeta Agent 의 실행시간이 Minimax Agent 보다 더 짧게 나왔음을 확인하였다.

<minimaxmap - Minmax>

```
----- START PART  MiniMax (depth=4) For Minimax Map:

====================================
[1] Pacman Lose... Score: -499
[2] Pacman Win! Score: 511
[3] Pacman Lose... Score: -494
[4] Pacman Lose... Score: -499
[5] Pacman Lose... Score: -494
[6] Pacman Lose... Score: -499
[7] Pacman Win! Score: 513
[8] Pacman Lose... Score: -499
[9] Pacman Lose... Score: -499
[10] Pacman Win! Score: 513
```

```
[990] Pacman Win! Score: 512
[991] Pacman Win! Score: 513
[992] Pacman Win! Score: 512
[993] Pacman Lose... Score: -496
[994] Pacman Lose... Score: -499
[995] Pacman Lose... Score: -494
[996] Pacman Lose... Score: -494
[997] Pacman Lose... Score: -499
[998] Pacman Win! Score: 513
[999] Pacman Lose... Score: -494
[1000] Pacman Win! Score: 512
----------Game Results-------------
Average Score: -102.478
```

```
Win Rate: 39% (390/1000)
Total Time: 49.38292956352234
Average Time: 0.04938292956352234
====================================
----- END  MiniMax (depth=4) For Minimax Map
```

결과 2-3. Minimax Agent – minimax map

<minimaxmap - AlphaBeta>

```
----- START PART  AlphaBeta (depth=4) For Minimax Map:

====================================
[1] Pacman Lose... Score: -494
[2] Pacman Win! Score: 512
[3] Pacman Win! Score: 513
[4] Pacman Lose... Score: -494
[5] Pacman Lose... Score: -494
[6] Pacman Win! Score: 513
[7] Pacman Lose... Score: -494
[8] Pacman Win! Score: 513
[9] Pacman Lose... Score: -499
[10] Pacman Lose... Score: -499
```

```
[990] Pacman Win! Score: 513
[991] Pacman Win! Score: 513
[992] Pacman Lose... Score: -494
[993] Pacman Lose... Score: -494
[994] Pacman Win! Score: 513
[995] Pacman Win! Score: 513
[996] Pacman Lose... Score: -494
[997] Pacman Lose... Score: -495
[998] Pacman Lose... Score: -495
[999] Pacman Lose... Score: -499
[1000] Pacman Win! Score: 512
----------Game Results-------------
Average Score: -128.678
```

```
Win Rate: 36% (364/1000)
Total Time: 37.953892946243286
Average Time: 0.03795389294624329
====================================
----- END  AlphaBeta (depth=4) For Minimax Map
```

결과 2-3. AlphaBeta Agent – minimax map

마지막으로 minimax map 에 대하여 depth = 4 로 설정하여 구동시킨 결과이다. 물론 map 이 다르기

때문에, 이에 대한 영향도 있겠지만, 앞서 depth = 2 로 설정하였을 때보다 승률이 현저히 높게 나온것을 확인할 수 있다.

시간을 비교하면, Minimax agent 는 총 시간 약 49.38 초, 평균 시간 약 0.0494 초가 나왔고, AlphaBeta agent 는 총 시간 약 37.95 초, 평균 시간 약 0.379 초가 나왔다. 앞서 depth = 2 로 설정하였던 small map 과 medium map 에서는 총 시간의 차이가 약 7 초정도 났던 것에 비해서 depth = 4 로 설정한 minimax map 에서는 12 초 정도 차이가 난 것으로 보아, depth 가 깊어질 수록 더 많은 node 들을 확인해야 하기 때문에 branch pruning 을 진행한 AlphaBeta agent 와 모든 node 들을 살펴보는 Minimax agent 의 실행시간 차이가 더 증가할 것으로 예측할 수 있다.

최종적으로, 모든 map 에 대해서 AlphaBeta agent 가 Minimax agent 보다 실행시간이 더 짧았고, 특히 depth 가 깊어질수록 그 차이는 더 커졌다. 다만, 게임의 승률에 있어서 AlphaBeta agent 가 더 좋게 나온다는 것은 보장할 수 없었다.

3.3 Expectimax Agent

```
python3 pacman.py -p ExpectimaxAgent -m stuckmap -a depth=3 -n 100 -q
```

```
====================================
[1] Pacman Win! Score: 532
[2] Pacman Lose... Score: -502
[3] Pacman Win! Score: 532
[4] Pacman Lose... Score: -502
[5] Pacman Lose... Score: -502
[6] Pacman Lose... Score: -502
[7] Pacman Lose... Score: -502
[8] Pacman Lose... Score: -502
[9] Pacman Win! Score: 532
[10] Pacman Win! Score: 532
[11] Pacman Win! Score: 532
[12] Pacman Win! Score: 532
[13] Pacman Lose... Score: -502
[14] Pacman Lose... Score: -502
[15] Pacman Win! Score: 532
[16] Pacman Win! Score: 532
[17] Pacman Win! Score: 532
[18] Pacman Lose... Score: -502
[19] Pacman Lose... Score: -502
[20] Pacman Win! Score: 532
[21] Pacman Lose... Score: -502
[22] Pacman Win! Score: 532
[23] Pacman Lose... Score: -502
[24] Pacman Win! Score: 532
[25] Pacman Win! Score: 532
```

```
[77] Pacman Lose... Score: -502
[78] Pacman Lose... Score: -502
[79] Pacman Win! Score: 532
[80] Pacman Lose... Score: -502
[81] Pacman Lose... Score: -502
[82] Pacman Win! Score: 532
[83] Pacman Win! Score: 532
[84] Pacman Win! Score: 532
[85] Pacman Win! Score: 532
[86] Pacman Win! Score: 532
[87] Pacman Lose... Score: -502
[88] Pacman Win! Score: 532
[89] Pacman Win! Score: 532
[90] Pacman Win! Score: 532
[91] Pacman Win! Score: 532
[92] Pacman Win! Score: 532
[93] Pacman Lose... Score: -502
[94] Pacman Lose... Score: -502
[95] Pacman Lose... Score: -502
[96] Pacman Win! Score: 532
[97] Pacman Lose... Score: -502
[98] Pacman Win! Score: 532
[99] Pacman Win! Score: 532
[100] Pacman Lose... Score: -502
```

결과 3-1. Expectimax Score

```
----------Game Results----------
Average Score: 46.02
Score Results: 532, -502, 532, 532, -50
32, -502, 532, 532, 532, -502, -502, 53
 532, 532, 532, 532, 532, 532, -502, 53
32, 532, -502, -502, -502, 532, -502, 5
 532, 532, 532, -502, -502, 532, 532, 5
Record: Win, Lose, Win, Win, Lose, Win,
e, Win, Win, Win, Lose, Lose, Win, Win,
in, Win, Win, Win, Win, Lose, Win, Lose
, Lose, Lose, Lose, Win, Lose, Win, Los
in, Win, Lose, Lose, Win, Win, Win, Los

Win Rate: 53% (53/100)
Total Time: 0.4981446266174316
Average Time: 0.004981446266174317
====================================
```

결과 3-2. Expectimax Win rate

.

명세서에 설명된 대로 패배한 경우 항상 -502 가, 승리한 경우 항상 532 가 출력됨을 확인하였다. 해당 실행에서는 53%의 승률이 나왔으며, 항상 동일한 승률이 나오지는 않았지만, 항상 50% 부근의 승률이 나오는 것을 확인하였다.