기초인공지능

Assignment02 보고서

20191621 이민영

1. 알고리즘마다 구현한 방법에 대한 설명

1) Minimax Agent

Minimax 알고리즘은 체스나 바둑과 같은 상대방과 번갈아 가면서 하는 게임의 경우에 주로 사용된다. 이때 자신과 상대방 모두 각자에게 유리한 방향으로 선택하게 된다. 따라서 상대방의 최대 유리한 방향을 자신에게 최소한의 영향을 끼치도록 해야한다.

차례로 max, min, max 순서를 번갈아 가면서 반복하게 된다. 이를 위해서 Max_Value 와 Min_Value 함수를 구현하였다.

<Max_Value>

```
def Max_Value(self,state,depth):
  if (depth == self.depth or state.isWin() or state.isLose()):
    return self.evaluationFunction(state)

  v = float("-inf")
  pac = 0
  for s in state.getLegalActions(pac):
    v2 = self.Min_Value(state.generateSuccessor(pac,s),depth,1)
    if(v2 > v):
      v = v2

  return v
```

우선 마지막 depth이거나 승패가 결정나면 evaluationFunction을 return한다. v의 값을 -무한대로 초기화하고, max는 pacman에 대해서 결정되는 것이기 때문에 기존 함수에 맞추어서 pac=0으로 설정하고 이를 state의 함수들에 사용한다.

state들이 취할 수 있는 행동들 즉, state.getLegalActions(pac)으로 반복문을 돌면서 이에 해당하는 Min_Value 함수를 호출하고, 이 중 가장 큰 값을 구하기 위해서 값을 업데이트 하고, 마지막으로 해당 v를 반환한다.

<Min_Value>

```
def Min_Value(self,state,depth,agent_num):
  if (depth == self.depth or state.isWin() or state.isLose()):
    return self.evaluationFunction(state)

  v = float("inf")

  for s in state.getLegalActions(agent_num):
    if (agent_num == state.getNumAgents()-1):
      v2 = self.Max_Value(state.generateSuccessor(agent_num,s),depth+1)
    else:
      v2 = self.Min_Value(state.generateSuccessor(agent_num,s),depth,agent_num+1)
    if v2 < v:
      v = v2

  return v
```

유령은 한마리가 아니기 때문에 Max_Value와 달리 Min_Value는 agent_num을 인자로 추가로 받는다. 마찬가지로 terminate 되는 조건을 설정하고 이에 해당할 경우에는 evaluationFunction을 return한다. v의 초기 값을 무한대로 설정한다.

state.getLegalActions를 돌고, 이 때, 마지막 agent의 경우에는 pacman을 불러야 하기 때문에 Max_Value함수를 깊이를 +1 한 상태로 호출하고, 이외의 agent의 경우에는 다음

agent를 불러야 하기 때문에 같은 depth의 값이고 agent_num은 1을 더한 값을 인자로 넘겨 Min_Value를 부른다. 이 때 반환되는 값이 v보다 작을 경우에 v를 업데이트 해주고 최종적으로 이를 반환한다.

<Action>

```
def Action(self,gameState):


  move_able = gameState.getLegalActions(0)
  move = Directions.STOP

  v = float("-inf")

  for i in move_able:
    aa = self.Min_Value(gameState.generateSuccessor(0,i),0,1)
    if aa > v:
      v = aa
      move = i

  return move
```

depth는 0, agent_num을 1로 설정하여 Min_Value를 부르고, 이를 최소화하기 위해서 더 작은 값이 있다면 작은 값으로 업데이트 해주고 이에 해당하는 action도 업데이트한다. 최종적으로 이 action값을 return 한다.

2) AlphaBeta Pruning Agent

위의 Minimax 알고리즘은, 복잡하고 많은 선택이 필요할 때에는 시간과 연산량이 매우 많아진다는 단점이 있다. 따라서 final decision에 영향을 주지 않는 branch들을 제거(pruning)해서 알고리즘을 향상시킬 수 있다. 여기서 알파는 max에 대한 최선의 선택 값이고, 베타는 min에 대한 최선의 결과이다. Min_Value에서 beta보다 더 작은 v값이 있으면 이를 갱신한다. Max_Value에서는 alpha보다 더 큰 v 값이 있으면 이를 갱신한다.

<Max_Value>

```
def Max_Value(self,state,depth,alpha,beta):
  if (depth == self.depth or state.isWin() or state.isLose() or len(state.getLegalActions(0))==0):
    return (self.evaluationFunction(state),None)

  v = float("-inf")
  pac = 0
  move = None
  for s in state.getLegalActions(0):
    v2 = self.Min_Value(state.generateSuccessor(pac,s),depth,1,alpha,beta)[0]
    if v2 > v:
      v,move = v2,s
    if(depth==0 and v> beta):
      return (v,move)
    if (depth != 0 and v >= beta):
      return (v,move)
    alpha = max(alpha,v)

  return (v,move)
```

기본적으로 Minimax알고리즘과 비슷하게 수행된다. AlphaBeta pruning 의 경우에는 Min_Value, Max_Value 함수에서 v,move 두가지를 return 값으로 주었기 때문에 이에 맞

추어서 일부를 수정하였다. 또한, alpha의 값보다 v가 크다면 alpha의 값을 업데이트 해주었다. 그리고 현재 살펴보는 v가 beta보다 같거나 클 경우에는 return을 통해 pruning이 될 수 있도록 하였다.

<Min_Value>

```
def Min_Value(self,state,depth,agent_num,alpha,beta):
  if (depth == self.depth or state.isWin() or state.isLose() or len(state.getLegalActions(0))==0):
    return self.evaluationFunction(state),None

  v = float("inf")


  move = None


  for s in state.getLegalActions(agent_num):
    if(agent_num == state.getNumAgents()-1):
      v2 = self.Max_Value(state.generateSuccessor(agent_num,s),depth+1,alpha,beta)[0]
    else:
      v2 = self.Min_Value(state.generateSuccessor(agent_num,s),depth,agent_num+1, alpha,beta)[0]
    if v2 < v:
      v,move = v2,s
    if (depth == 0 and v < alpha):
      return (v,move)
    if(depth != 0 and v<=alpha):
      return (v,move)
    beta = min(beta,v)

  return (v,move)
```

또한, beta의 값보다 v가 작다면 beta의 값을 업데이트 해주었다. 그리고 현재 살펴보는 v가 alpha보다 같거나 작을 경우에는 return을 통해 pruning이 될 수 있도록 하였다.

<Action>
Action함수에서는 Max함수를 depth가 0인 상태로 호출해서 return 값의 두번째 값 즉, move를 Action함수의 return 값으로 주었다.

3) Expectimax Agent

위의 두 알고리즘은 상대가 optimal한 결정을 할거라 가정하고 계산을 하게 된다. 다라서 상대가 확률적으로 잘못된 선택을 할 수 도 있다는 가정을 하여 action을 결정하는 것인 Expectimax Agent이다. 따라서 상대에서 min을 통해서 구하는 것이 아니라 chance 노드로 생각해서 선택을 하게 된다.

```
def Action(self,gameState):


  move_able = gameState.getLegalActions(0)
  move = Directions.STOP

  v = float("-inf")

  for i in move_able:
    aa = self.Min_Value(gameState.generateSuccessor(0,i),0,1)
    if aa > v:
      v = aa
      move = i
  return move


def Max_Value(self,state,depth):
  if (depth == self.depth or state.isWin() or state.isLose()):
    return self.evaluationFunction(state)

  v = float("-inf")
  pac = 0
  for s in state.getLegalActions(pac):
    v = max(v,self.Min_Value(state.generateSuccessor(pac,s),depth,1))

  return v

def Min_Value(self,state,depth,agent_num):
  if (depth == self.depth or state.isWin() or state.isLose()):
    return self.evaluationFunction(state)

  v = 0

  if (agent_num == state.getNumAgents()-1):
    for s in state.getLegalActions(agent_num):
      v = v + self.Max_Value(state.generateSuccessor(agent_num,s),depth+1)
  else:
    for s in state.getLegalActions(agent_num):
      v = v + self.Min_Value(state.generateSuccessor(agent_num,s),depth,agent_num+1)

  return float(v/len(state.getLegalActions(agent_num)))
```

기본적인 틀은 minimax 알고리즘과 매우 유사하며, Min_Value의 경우, 합을 통해서 v값을 정했고, 이를 LegalActions로 나눈 값으로 반환하여 결과적으로 확률적인 값으로 진행되게 된다.

2. 실행 캡처 화면

1) Minimax Agent 명령어의 승률 출력 화면

<table>
<tr><td>

```
Win Rate: 64% (644/1000)
Total Time: 35.95794486999512
Average Time: 0.035957944869995116
========================================
```

</td></tr>
<tr><td>승률 : 64%</td></tr>
<tr><td>

```
Win Rate: 67% (670/1000)
Total Time: 36.02318620681763
Average Time: 0.036023186206817624
========================================
```

</td></tr>
<tr><td>승률 : 67%</td></tr>
</table>

```
Win Rate: 65% (654/1000)
Total Time: 36.80268692970276
Average Time: 0.03680268692970276
======================================
```

승률 : 65%

python pacman.py -p MinimaxAgent -m minimaxmap -a depth=4 -n 1000 -q를 세 번 반복하여 실행했을 때 승률이 50~70%를 만족하는 것을 확인할 수 있다.

2) time_check.py에서 출력된 실행시간 캡처 화면 (small map, mediummap)

```
Total Time: 51.7571713924407996
Average Time: 0.17252390464146933
======================================
----- END  MiniMax (depth=2) For Small Map
```

Minimax – Small Map

```
Total Time: 36.93789529800415
Average Time: 0.12312631766001383
======================================
----- END  AlphaBeta (depth=2) For Small Map
```

AlphaBeta – Small Map

```
Total Time: 108.63911890983582
Average Time: 0.3621303963661194
======================================
----- END  MiniMax (depth=2) For Medium Map
```

Minimax – Medium Map

```
Total Time: 93.60122752189636
Average Time: 0.31200409173965454
======================================
----- END  AlphaBeta (depth=2) For Medium Map
```

AlphaBeta – Medium Map

```
Total Time: 55.08908295631409
Average Time: 0.05508908295631409
======================================
----- END  MiniMax (depth=4) For Minimax Map
```

Minimax – Minimax Map

```
Total Time: 29.74213433265686
Average Time: 0.02974213433265686
======================================
----- END  AlphaBeta (depth=4) For Minimax Map
```

AlphaBeta – Minimax Map

두 값을 비교해보았을 때, AlphaBeta가 Minimax에 비해서 더 빠르게 수행된 다는 것을 유의미하게 확인해볼 수 있었다.

3) Expectimax Agent 명령어의 승률 및 Score 출력 화면

```
-----------Game Results--------------
Average Score: 25.34
Score Results: -502, 532, 532, 532, 532, -502, -502, 532, 532, -502, 532, -502, 532, -502, -502, -502, 532, -502, -502, -502, -502, -502
, -502, 532, 532, -502, 532, 532, -502, -502, 532, -502, -502, 532, -502, 532, 532, -502, -502, -502, -502, 532, 532, -502, 532, 532, 53
2, 532, -502, 532, -502, -502, -502, 532, 532, -502, 532, -502, -502, -502, 532, -502, -502, -502, -502, -502, -502, 532, -502, 532, -50
2, 532, 532, 532, 532, 532, 532, 532, 532, 532, 532, -502, 532, 532, -502, 532, 532, -502, -502, 532, 532, -502, -502, 532, 532, -502, 5
32, 532, -502, 532
Record: Lose, Win, Win, Win, Win, Lose, Lose, Win, Win, Lose, Win, Lose, Win, Lose, Lose, Lose, Win, Lose, Lose, Lose, Lose, Lose, Lose,
 Win, Win, Lose, Win, Win, Lose, Lose, Win, Lose, Lose, Win, Lose, Win, Win, Lose, Lose, Lose, Lose, Win, Win, Lose, Win, Win, Win, Win,
 Lose, Win, Lose, Lose, Lose, Win, Win, Lose, Win, Lose, Lose, Lose, Win, Lose, Lose, Lose, Lose, Lose, Lose, Win, Lose, Win, Lose, Win,
 Win, Win, Win, Win, Win, Win, Win, Win, Win, Lose, Win, Win, Lose, Win, Win, Lose, Lose, Win, Win, Lose, Lose, Win, Win, Lose, Win, Win
, Lose, Win

Win Rate: 51% (51/100)
Total Time: 0.1944887638092041
Average Time: 0.001944887638092041
```

승률 : 51%

```
----------Game Results----------
Average Score: 15.0
Score Results: -502, 532, -502, -502, 532, -502, 532, -502, 532, -502, -502, -502, -502, 532, -502, 532, -502, 532, 532, 53
 532, 532, -502, 532, -502, 532, -502, -502, -502, -502, 532, 532, -502, -502, -502, 532, 532, -502, -502, 532, -502, 532,
502, 532, -502, 532, 532, 532, -502, -502, 532, 532, -502, -502, -502, -502, -502, 532, -502, -502, -502, 532, 532, -502, 5
, 532, 532, 532, 532, -502, 532, 532, 532, 532, 532, 532, 532, 532, -502, -502, 532, 532, -502, -502, -502, 532, -502, 532,
32, -502, 532, -502
Record: Lose, Win, Lose, Lose, Win, Lose, Win, Lose, Win, Lose, Lose, Lose, Lose, Win, Lose, Win, Lose, Win, Win, Win, Lose
in, Lose, Win, Lose, Win, Lose, Lose, Lose, Lose, Win, Win, Lose, Lose, Lose, Win, Win, Lose, Lose, Win, Lose, Win, Lose, L
n, Lose, Win, Win, Win, Lose, Lose, Win, Win, Lose, Lose, Lose, Lose, Lose, Win, Lose, Lose, Lose, Win, Win, Lose, Win, Los
Win, Win, Win, Lose, Win, Win, Win, Win, Win, Win, Win, Win, Lose, Lose, Win, Win, Lose, Lose, Lose, Win, Lose, Win, Win, L
e, Win, Lose

Win Rate: 50% (50/100)
Total Time: 0.19225382804870605
Average Time: 0.0019225382804870606
```

승률 : 50%

```
----------Game Results----------
Average Score: -5.68
Score Results: 532, -502, -502, 532, -502, 532, -502, -502, -502, 532, -502, -502, 532, -502, 532, -502, -502, -502, -502,
2, 532, -502, 532, -502, -502, 532, 532, 532, -502, -502, 532, 532, 532, 532, 532, -502, -502, 532, 532, 532, -502, -502, -
2, 532, -502, 532, 532, 532, 532, -502, -502, -502, 532, -502, -502, 532, -502, -502, -502, 532, -502, -502, -502, -502, 53
 532, -502, 532, -502, -502, -502, -502, 532, -502, 532, 532, 532, 532, 532, 532, -502, 532, -502, -502, -502, 532, 532, 53
, 532, 532, -502, 532
Record: Win, Lose, Lose, Win, Lose, Win, Lose, Lose, Lose, Win, Lose, Lose, Win, Lose, Win, Lose, Lose, Lose, Lose, Win, Lo
 Lose, Win, Lose, Lose, Win, Win, Win, Lose, Lose, Win, Win, Win, Win, Win, Lose, Lose, Win, Win, Win, Lose, Lose, Lose, Wi
 Lose, Win, Win, Win, Win, Lose, Lose, Lose, Win, Lose, Lose, Win, Lose, Lose, Lose, Win, Lose, Lose, Lose, Lose, Win, Win,
ose, Win, Lose, Lose, Lose, Lose, Win, Lose, Win, Win, Win, Win, Win, Win, Lose, Win, Lose, Lose, Lose, Win, Win, Win, Lose
Win, Lose, Win

Win Rate: 48% (48/100)
Total Time: 0.19159722328186035
Average Time: 0.0019159722328186035
```

승률 : 48%

결과값으로 이긴 경우에는 532를 출력하고 패배한 경우에는 -502를 출력 함을 확인할 수 있었고, 승률 또한 3번 반복하여 출력해보았을 때 50% 정도의 승률을 보인다는 것을 확인해 볼 수 있었다.

4) minimax의 initial state

```
(venv) lee-minyoung@imin-yeong-ui-iMac AI_Assignment02 % python pacman.py -p MinimaxAgent -m minimaxmap -a depth=1
9
(venv) lee-minyoung@imin-yeong-ui-iMac AI_Assignment02 % python pacman.py -p MinimaxAgent -m minimaxmap -a depth=2
8
(venv) lee-minyoung@imin-yeong-ui-iMac AI_Assignment02 % python pacman.py -p MinimaxAgent -m minimaxmap -a depth=3
7
(venv) lee-minyoung@imin-yeong-ui-iMac AI_Assignment02 % python pacman.py -p MinimaxAgent -m minimaxmap -a depth=4
-492
```

depth가 각각 1,2,3,4 일 때 초기값이 9, 8, 7, -492 임을 확인해 볼 수 있었다.