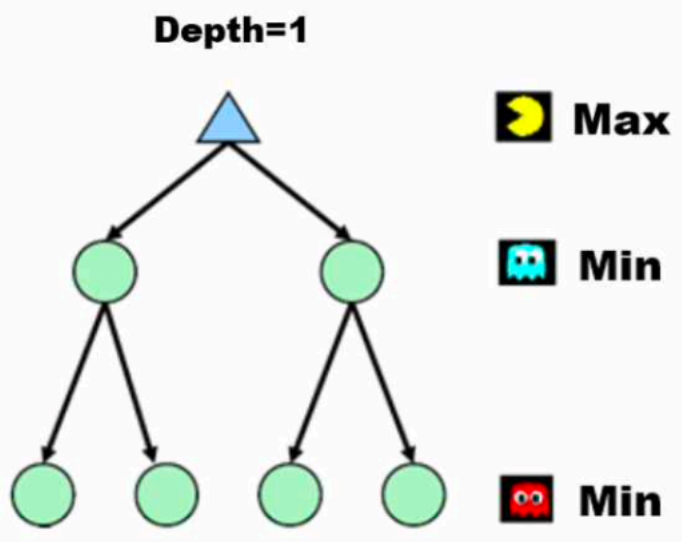
기초 인공지능(CSE4185) Assignment #2

2분반 20200183 박정원

(1) 각 알고리즘마다 구현한 방법에 대한 설명



[그림 02] Pacman에서의 depth와 index 개념

a) MinimaxAgent

과제에서 나온 그림과 같이, 팩맨(파란색 세모)은 다음 상태의 점수에서 최댓값을 가져오고, 고스트들(초록색 원)은 다음 상태의 점수에서 팩맨의 이익을 최소화할 수 있는 최솟값을 가져온다.

```
def Minimax(agentIndex, status, k):
```

구현은 편의를 위해 재귀로 구현하였고, 각 재귀의 layer가 어떤 에이전트의 차례인지를 알 수 있도록 agentIndex를 인자로 넘겨주고, 현재 게임의 상태 정보를 저장하는 status 객체와 현재의 depth를 알려주는 k를 추가적인 인자로 가졌다.

```
nextCandidate = status.getLegalActions(agentIndex) # if Win or Lose, return empty list

if len(nextCandidate) == 0 or k == self.depth: # 이기거나 졌거나, max depth에 도달했을 경우
  return (self.evaluationFunction(status), None)

if agentIndex == 0: # meaning that agent is the pacman / max player

  Max = float("-inf")
  act = None

  for i in range(len(nextCandidate)):
    # 자식 노드들 중에서 최댓값을 가져온다.
    # 팩맨은 Max player 이기 때문에 자신의 이익을 극대화시키는 가능한 큰 점수를 선택한다.
    res = Minimax(agentIndex+1, status.generateSuccessor(agentIndex, nextCandidate[i]), k)
    if res[0] > Max:
      Max = res[0]
      act = nextCandidate[i]

  return (Max, act) # 최대일 때의 행동까지 튜플 형태로 반환
```

먼저 getLegalActions(agentIndex)를 통해 팩맨이면 팩맨, 고스트면 고스트에 맞는 다음 행동들을 리

스트에 담아 받아온다. 행동들은 'North', 'West', 'East', 'South' 와 같이 string 형태로 되어있다.

만약 이 행동 리스트의 길이가 0이거나(게임을 이기거나 졌다는 뜻. getLegalActions의 구현을 보면 가장 상단에 isWin이거나 isLose 일 때 빈 리스트를 반환한다.), k 가 input 으로 주어진 depth에 이르렀다면, 해당 상태의 스코어를 계산하는 evaluationFunction을 리턴해준다. 이 때 행동은 아무것도 안 하기 때문에 None을 반환.

이 후 agentIndex가 0일 경우 팩맨의 차례라는 뜻이므로, Successor State에 대해 호출한 Minimax의 값 중 최댓값을 취해서 그 때의 행동과 함께 반환한다.

```
else: # meaning that agent is the ghost / min player

  Min = float("inf")
  act = None
  # 고스트는 Min player 이기 때문에 팩맨의 이익을 극소화시키는 가능하면 작은 점수를 선택한다.

  if agentIndex == status.getNumAgents() - 1: # 현재 에이전트가 마지막 고스트이다.
    for i in range(len(nextCandidate)):
      # 마지막 고스트까지 Action을 취하면 depth 싸이클이 끝난 것이다.
      # 따라서, 다시 팩맨에게 차례를 넘겨주어야 한다. 첫 번째 인자로 팩맨의 인덱스인 0을 넘겨준다.
      # 한 depth 가 끝난 것이므로 depth 1 증가. k+1
      res = Minimax(0, status.generateSuccessor(agentIndex, nextCandidate[i]), k+1)
      if Min > res[0]:
        Min = res[0]
        act = nextCandidate[i]

  else:
    for i in range(len(nextCandidate)):
      # 다음 고스트에게 차례를 넘겨주어야 한다.
      res = Minimax(agentIndex+1, status.generateSuccessor(agentIndex, nextCandidate[i]), k)
      if Min > res[0]:
        Min = res[0]
        act = nextCandidate[i]

  return (Min, act) # 최소일 때의 행동까지 튜플 형태로 반환
```

agentIndex가 0 이 아닌 경우, 고스트들의 차례이고, 만약 인덱스가 현재 게임의 에이전트 수 – 1 이면 마지막 고스트의 차례니까 다음 호출할 때는 다시 팩맨 인덱스인 0을 넘겨주어야 하고, depth를 1 올린 상태인 k+1 을 넘겨주어야 한다. 마지막 고스트가 아닌 경우에는 그냥 현재 agentIndex에 1을 추가한 값을 넘겨주고, depth는 그대로 유지한다. Depth의 개념은 맨 위에 그림을 참고하면 되겠다. 고스트들은 Min agent 이기 때문에 팩맨의 이익이 최소가 되는 방향으로 선택을 하기 때문에 최솟값을 취해주고 그 때의 행동과 함께 반환한다.

```
return Minimax(0, gameState, 0)[1]
```

다시 바깥 Action 메소드로 돌아오면, 반환 값으로 Minimax(0, gameState, 0)[1] 을 돌려준다. 튜플 형태가 (최대 점수, 그 때의 행동) 이기 때문이다.

b) AlphaBetaAgent

Minimax 에 Alpha Beta Pruning 기법을 적용한 AlphaBetaAgent의 구현은 Minimax 구현의 기본적인 골격은 모두 유지하지만, 이제 alpha 값과 beta 값을 비교하는 구문만 추가해주었다.

```
def AlphaBetaPruning(agentIndex, status, k, alpha, beta):
```

Minimax 함수보다 두 개의 인자를 더 갖고, 각각 alpha와 beta 값이다.

팩맨 차례

```python
if agentIndex == 0: # meaning that agent is the pacman / max player

  Max = float("-inf")
  act = None

  for i in range(len(nextCandidate)):
    res = AlphaBetaPruning(agentIndex+1, status.generateSuccessor(agentIndex, nextCandidate[i]), k, alpha, beta)
    if res[0] > Max:
      Max = res[0]
      act = nextCandidate[i]

    if Max >= beta: # beta 보다 Max 값이 크거나 같으면, 더 이상 탐색할 필요가 없다. 위의 Min player는 어차피 beta 값을 유지할 것이기 때문이다.
      return (Max, act)
    alpha = max(alpha, Max) # Max player는 알파값을 업데이트한다.

  return (Max, act)
```

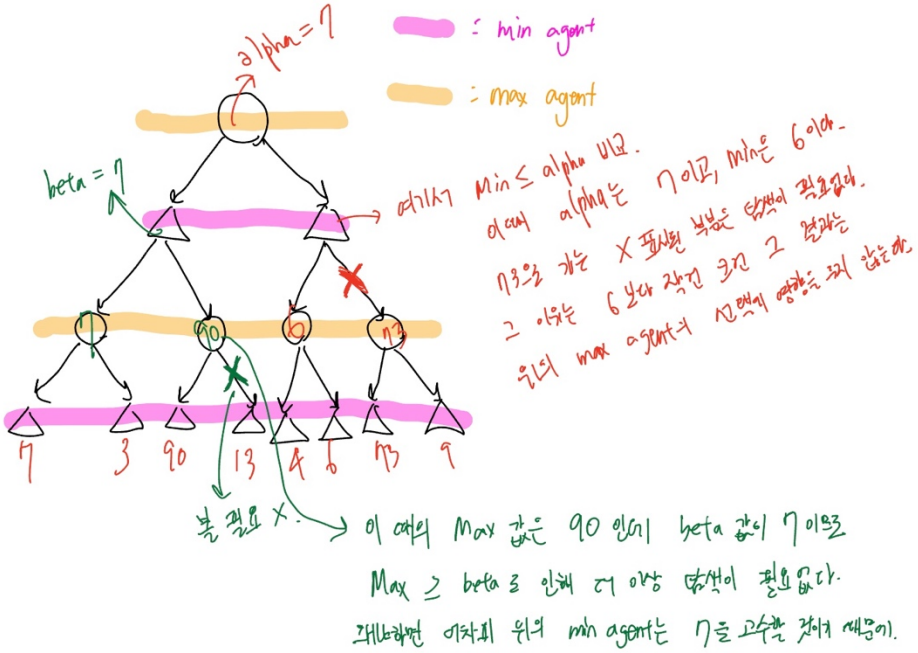
고스트 차례

```python
else: # meaning that agent is the ghost / min player

  Min = float("inf")
  act = None

  if agentIndex == status.getNumAgents() - 1: # 현재 에이전트가 마지막 고스트이다.
    for i in range(len(nextCandidate)):
      res = AlphaBetaPruning(0, status.generateSuccessor(agentIndex, nextCandidate[i]), k+1, alpha, beta) # 한 depth 가 끝난 것이므로 depth 1 증가
      if Min > res[0]:
        Min = res[0]
        act = nextCandidate[i]

      # 저장된 알파값보다 Min 값이 작거나 같으면, 더 이상 탐색할 필요가 없다. 위의 Max player는 어차피 alpha 값을 유지하게 되기 때문이다.
      if Min <= alpha:
        return (Min, act)
      beta = min(beta, Min) # Min player는 베타값을 업데이트한다.

  else:
    for i in range(len(nextCandidate)):
      # 호출 방식은 MinimaxAgent과 동일
      res = AlphaBetaPruning(agentIndex+1, status.generateSuccessor(agentIndex, nextCandidate[i]), k, alpha, beta)
      if Min > res[0]:
        Min = res[0]
        act = nextCandidate[i]

      if Min <= alpha:
        return (Min, act)
      beta = min(beta, Min)
```

팩맨의 차례일 때를 보면, Max 값과 beta를 비교하는 것을 볼 수 있다.

고스트들의 차례일 때는, Min 값과 alpha를 비교한다.

왜 이런 비교를 해주는 지는 그림으로 설명하도록 하겠다.



위 그림과 같은 게임 상태를 가정했을 때, Max >= beta 부분(초록색)과 Min <= alpha 부분(빨간색)을 예시로 들어놨다.

처음 호출할 때의 alpha 값은 float("-inf"), beta 값은 float("inf")이다. 그 이유는 각각 max agent와 min agent가 업데이트 하는 값이기 때문에 매우 작은 값과 매우 큰 값을 넣어주었다.

```
return AlphaBetaPruning(0, gameState, 0, A, B)[1]
```

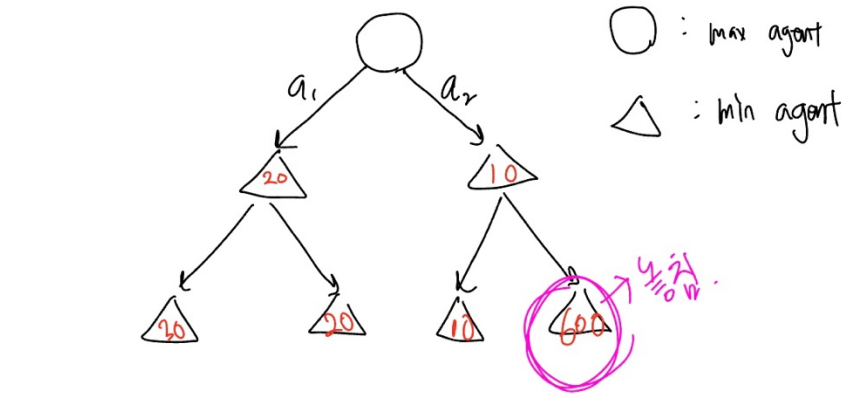
밖에 Action 함수에서는 리턴 값을 Minimax 와 동일하게 튜플에서 행동 값만 반환해주면 된다.
호출할 때 A에는 float("-inf") 값이, B에는 float("inf") 값이 들어가 있다.

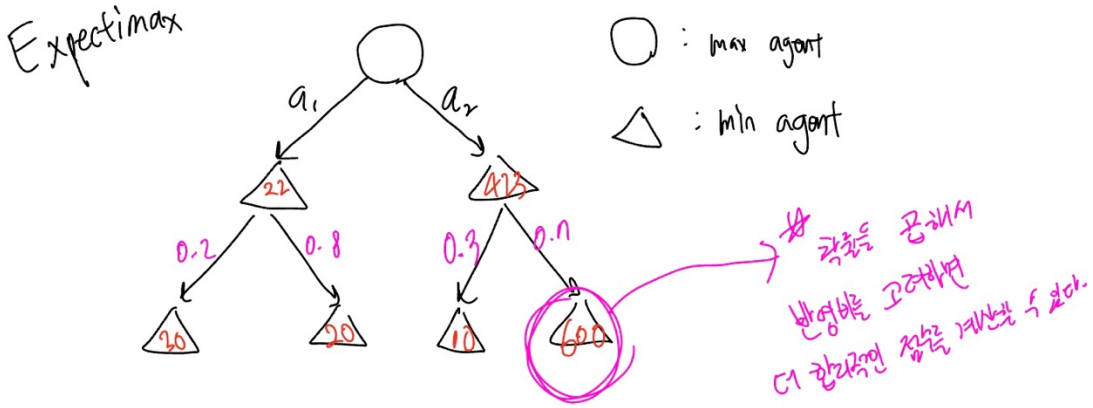
c) ExpectimaxAgent

```
def Expectimax(agentIndex, status, k):
```

ExpectimaxAgent는 아예 Min값 혹은 Max 값을 취하는 것이 아니라, 각각의 상태에 확률을 새로 부여하여 점수와 곱해서 점수를 구하는 방식으로 구한다. 확률을 점수에 곱하는 이유는 간단한게, minimax의 경우 좋은 점수를 아예 놓칠 기회가 있기 때문에 어떤 가능한 상태를 아예 배제하는 것이 아니라 반영비를 고려하겠다는 뜻이다.

이 또한 그림으로 설명하겠다.





하지만 현재 우리가 구현하고자 하는 Expectimax 는 확률이 따로 주어지지 않았기 때문에 본인은 평균값, 즉 기댓값을 반환하도록 할 것이다. (각각의 경우에 대해 동일한 확률을 부여, 즉 모든 점수를 더한 뒤에 행동들의 개수로 나누면 평균 점수가 구해질 것이다.)

```
def Expectimax(agentIndex, status, k):

  nextCandidate = status.getLegalActions(agentIndex) # if Win or Lose, return empty list

  if len(nextCandidate) == 0 or k == self.depth: # 다음 선택지가 없거나, depth에 도달했을 때
    return (self.evaluationFunction(status), None)

  if agentIndex == 0: # meaning that agent is the pacman / max player
    Max = float("-inf")
    act = None

    for i in range(len(nextCandidate)):
      res = Expectimax(agentIndex+1, status.generateSuccessor(agentIndex, nextCandidate[i]), k)
      if res[0] > Max:
        Max = res[0]
        act = nextCandidate[i]

    return (Max, act)
```

만약 현재가 팩맨의 차례이면, 더 큰 점수로 이어지는 행동을 하는 것이 이득이므로, Minimax 일 때와 똑같이 최댓값일 때의 행동을 가져와 반환한다.

```
else: # meaning that agent is the ghost / min player
  Sum = 0
  if agentIndex == status.getNumAgents() - 1: # 현재 에이전트가 마지막 고스트이다.
    for i in range(len(nextCandidate)):
      # 확률을 곱한다는 것 외에 MinimaxAgent 구현과 다르지 않다.
      Sum += Expectimax(0, status.generateSuccessor(agentIndex, nextCandidate[i]), k+1)[0]

  else:
    for i in range(len(nextCandidate)):
      # 확률을 곱한다는 것 외에 MinimaxAgent 구현과 다르지 않다.
      Sum += Expectimax(agentIndex+1, status.generateSuccessor(agentIndex, nextCandidate[i]), k)[0]

  return (Sum / len(nextCandidate), None) # 기댓값을 반환
```

고스트의 경우에는 그림에서와 같이 Successor에서 도출한 점수의 평균값을 취해서 반환할 것이다. 이 때 평균값은 (각각의 행동들에 대한 점수의 합계 / 행동들의 개수)가 될 것이다. 그래서 Sum 변수에 우선 모든 점수를 더하고, 마지막에 nextCandidate 리스트의 길이로 나누어 반환한다.

```
return Expectimax(0, gameState, 0)[1]
```

리턴값은 위와 동일하게 튜플에서의 인덱스 1값, 즉 행동을 반환하면 된다.

(2) 실행 캡쳐 화면

a) Minimax Agent 명령어의 승률 출력 화면

```
Win Rate: 63% (637/1000)
Total Time: 50.493948459625244
Average Time: 0.0504939484596252 44
========================================
```

b) time_check.py에서 출력된 실행 시간 캡처 화면 (mediummap, minimaxmap)

```
Win Rate: 17% (53/300)
Total Time: 245.20275330543518
Average Time: 0.8173425110181173
========================================
----- END  MiniMax (depth=3) For Medium Map

Win Rate: 13% (40/300)
Total Time: 128.98207259178162
Average Time: 0.4299402419726054
========================================
----- END  AlphaBeta (depth=3) For Medium Map
```

```
Win Rate: 38% (388/1000)
Total Time: 33.30535960197449
Average Time: 0.03330535960197449
========================================
----- END  MiniMax (depth=4) For Minimax Map
```

```
Win Rate: 35% (355/1000)
Total Time: 15.93103289604187
Average Time: 0.01593103289604187
========================================
----- END  AlphaBeta (depth=4) For Minimax Map
```

c) Expectimax Agent 명령어의 승률 및 score 출력 화면

```
----------Game Results----------
Average Score: 15.0
Score Results: 532, -502, -502, -502, 532, 532, 532, 532, 532, 532, 532, 532, 532, -502, -502, 532, -502, -502, -502, -502, -502, -502, -502, 532, -502, 532, -502, -502, -502, -502, 532, -502, 532, -502, 532, -502, 532, -502, 532, -502, 532, 532, -502, 532, -502, 532, 532, 532, 532, 532, -502, -502, -502, -502, -502, 532, 532, 532, 532, -502, -502, 532, -502, 532, 532, -502, -502, -502, -502, -502, -502, -502, -502, 532, 532, 532, -502, -502, -502, 532, -502, -502, -502, 532, 532, 532, 532, 532, 532, 532, -502, -502, -502, 532, 532, 532, -502, 532, 532, 532, 532
Record: Win, Lose, Lose, Lose, Win, Win, Win, Win, Win, Win, Win, Win, Win, Lose, Lose, Win, Lose, Lose, Lose, Lose, Lose, Lose, Lose, Win, Lose, Win, Lose, Lose, Lose, Win, Lose, Win, Lose, Win, Lose, Win, Lose, Win, Lose, Win, Lose, Win, Win, Win, Win, Win, Lose, Lose, Lose, Lose, Lose, Win, Win, Win, Lose, Lose, Lose, Win, Lose, Lose, Lose, Win, Win, Win, Lose, Lose, Lose, Win, Lose, Win, Win, Lose, Lose, Lose, Lose, Lose, Lose, Lose, Win, Win, Win, Lose, Lose, Lose, Win, Lose, Lose, Lose, Win, Win, Win, Win, Win, Win, Lose, Lose, Lose, Win, Win, Win, Lose, Win, Win, Win

Win Rate: 50% (50/100)
Total Time: 0.27133893966674805
Average Time: 0.0027133893966674806
====================================
----------Game Results----------
Average Score: 25.34
Score Results: 532, -502, -502, 532, 532, -502, 532, -502, -502, 532, -502, -502, -502, -502, -502, -502, 532, -502, 532, 532, -502, 532, -502, -502, -502, -502, 532, -502, 532, 532, 532, -502, 532, 532, -502, 532, 532, -502, 532, 532, -502, 532, 532, -502, -502, 532, -502, 532, 532, -502, 532, 532, -502, 532, 532, -502, 532, 532, 532, -502, 532, 532, 532, 532, -502, -502, -502, -502, 532, 532, 532, 532, -502, 532, 532, -502, 532, 532, -502, 532, -502, 532, 532, 532, -502, 532, 532, -502, -502, -502, -502, -502, -502, 532, 532, 532, -502, 532, 532, 532, 532, -502, 532, -502, -502
Record: Win, Lose, Lose, Win, Win, Lose, Win, Lose, Lose, Win, Lose, Lose, Lose, Lose, Lose, Lose, Win, Lose, Win, Win, Lose, Win, Win, Lose, Lose, Win, Lose, Win, Lose, Lose, Win, Lose, Win, Lose, Lose, Lose, Lose, Lose, Win, Win, Win, Lose, Win, Win, Lose, Lose, Lose, Lose, Lose, Win, Win, Win, Lose, Win, Lose, Lose, Lose, Win, Win, Win, Win, Lose, Win, Win, Win, Win, Lose, Win, Lose, Lose, Lose, Lose, Lose, Win, Win, Lose, Win, Win, Win, Lose, Win, Lose, Lose

Win Rate: 51% (51/100)
Total Time: 0.26619720458984375
Average Time: 0.0026619720458984377
====================================
```