Assignment02 - Pacman

20160768 김홍엽

1. 각 알고리즘 설명 (MinimaxAgent, AlphaBetaAgent, ExpectimaxAgent)

MinimaxAgent

```
class MinimaxAgent(AdversialSearchAgent):
  """
    [문제 01] MiniMax의 Action을 구현하시오. (20점)
    (depth와 evaluation function은 위에서 정의한 self.depth and self.evaluationFunction을 사용할 것.)
  """
  def Action(self, gameState):
    ######################## Write Your Code Here ############################
    move_candidate = gameState.getLegalActions()
    scores = []
    for action in move_candidate:
      successorGameState = gameState.generatePacmanSuccessor(action)
      scores.append(self.minValue(successorGameState, 0, 1))

    bestScore = max(scores)
    Index = [index for index in range(len(scores)) if scores[index] == bestScore]
    get_index = random.choice(Index)

    return move_candidate[get_index]

  def isTerminal(self, gameState, currentDepth):
    return True if currentDepth == self.depth or gameState.isWin() or gameState.isLose() else False

  def isPacman(self, agentIndex):
    return True if agentIndex == 0 else False

  def maxValue(self, gameState, currentDepth):
    if self.isTerminal(gameState, currentDepth): # Terminal Test
      return self.evaluationFunction(gameState)

    move_candidate = gameState.getLegalActions()
    scores = []

    for action in move_candidate:
      successorGameState = gameState.generatePacmanSuccessor(action)
      scores.append(self.minValue(successorGameState, currentDepth, 1))

    return max(scores)

  def minValue(self, gameState, currentDepth, agentIndex):
    if self.isTerminal(gameState, currentDepth): # Terminal Test
      return self.evaluationFunction(gameState)

    move_candidate = gameState.getLegalActions(agentIndex)
    scores = []
    ghostNum = gameState.getNumAgents() - 1

    for action in move_candidate:
      successorGameState = gameState.generateSuccessor(agentIndex, action)

      if agentIndex == ghostNum: # 팩맨으로 넘어감
        nextDepth = currentDepth + 1
        scores.append(self.maxValue(successorGameState, nextDepth))
      else: # 다음 고스트
        nextGhost = agentIndex + 1
        scores.append(self.minValue(successorGameState, currentDepth, nextGhost))

    return min(scores)

    ##########################################################################
```

Action(self, gameState)	팩맨이 주어진 depth 안에서 점수를 최대화하는 움직임을 최종적으로 반환하는 함수이다. 1. 현재 state에서 팩맨이 이동할 수 있는 action을 취했을 때의 다음 state와 현재 depth인 0, 첫 번째 유령의 index인 1을 파라미터로 하여 minValue 함수를 호출하고 그 값을 저장한다. 2. 1번의 과정을 팩맨이 이동할 수 있는 모든 action에 대해 반복한다. 3. minValue 함수를 통해 반환된 값들 중 가장 높은 점수를 내는 움직임을 최종적으로 반환한다.

MinimaxAgent	

maxValue(self, gameState, currentDepth)	Max agent, 즉 팩맨이 현재 state에서 얻을 수 있는 점수들 중 가장 높은 점수를 반환하는 함수이다.
	1. 먼저 isTerminal 함수를 통해 현재 state에 대한 Terminal Test를 진행한다.
	  1. Terminal 상태라면, evaluationFunction 함수를 호출하여 그 값을 반환한다.
	  2. Terminal 상태가 아니라면 2번을 진행한다.
	2. 현재 state에서 팩맨이 이동할 수 있는 action을 취했을 때의 다음 state와 현재 depth인 currentDepth, 첫 번째 유령의 index인 1을 파라미터로 하여 minValue 함수를 호출하고 그 값을 저장한다.
	3. 2번의 과정을 팩맨이 이동할 수 있는 모든 action에 대해 반복한다.
	4. minValue 함수를 통해 반환된 값들 중 가장 높은 점수를 반환한다.

minValue(self, gameState, currentDepth, agentIndex)	Min agent, 즉 유령이 현재 state에서 얻을 수 있는 점수들 중 가장 낮은 점수를 반환하는 함수이다.
	1. 먼저 isTerminal 함수를 통해 현재 state에 대한 Terminal Test를 진행한다.
	  1. Terminal 상태라면, evaluationFunction 함수를 호출하여 그 값을 반환한다.
	  2. Terminal 상태가 아니라면 2번을 진행한다.
	2. 현재 state에서 현재 유령이 이동할 수 있는 action을 취했을 때의 다음 state를 구한다.
	3. 만약 현재 유령이 두 번째 유령(agentIndex=2)이라면, 모든 agent가 움직였으므로 다음 state와 다음 depth에 대한 maxValue 함수를 호출하고 그 값을 저장한다.
	4. 현재 유령이 첫 번째 유령(agentIndex=1)이라면, 다음 state와 현재 depth인 currentDepth, 두 번째 유령의 index인 nextGhost(=2)를 파라미터로 하여 minValue 함수를 호출하고 그 값을 저장한다.
	5. 2-4번의 과정을 유령이 이동할 수 있는 모든 action에 대해 반복한다.
	6. 저장된 값들 중에서 가장 낮은 점수를 반환한다.

isTerminal(self, gameState, currentDepth)	gameState와 currentDepth가 주어졌을 때, terminal 상태인지 아닌지를 반환하는 함수이다. 다음과 같은 상황일 때 terminal 상태로 판단하여 True를 반환한다. 1. currentDepth가 미리 지정된 depth와 같을 때 2. 팩맨이 이긴 상황일 때 3. 팩맨이 진 상황일 때
isPacman(self, agentIndex)	agentIndex가 주어졌을 때, 팩맨이라면(agentIndex=0) True를 반환하고 아니라면 False를 반환한다.

AlphaBetaAgent

```python
class AlphaBetaAgent(AdversialSearchAgent):
  """
    [문제 02] AlphaBeta의 Action을 구현하시오. (25점)
    (depth와 evaluation function은 위에서 정의한 self.depth and self.evaluationFunction을 사용할 것.)
  """
  def Action(self, gameState):
    ######################## Write Your Code Here ################################
    move_candidate = gameState.getLegalActions()
    scores = []
    alpha = float('-inf')
    beta = float('inf')
    bestScore = float('-inf')

    for action in move_candidate:
      successorGameState = gameState.generatePacmanSuccessor(action)
      scores.append(self.minValue(successorGameState, 0, 1, alpha, beta))

      bestScore = max(scores)
      alpha = max(alpha, bestScore)

    Index = [index for index in range(len(scores)) if scores[index] == bestScore]
    get_index = random.choice(Index)

    return move_candidate[get_index]

  def isTerminal(self, gameState, currentDepth):
    return True if currentDepth == self.depth or gameState.isWin() or gameState.isLose() else False

  def isPacman(self, agentIndex):
    return True if agentIndex == 0 else False

  def maxValue(self, gameState, currentDepth, alpha, beta):
    if self.isTerminal(gameState, currentDepth): # Terminal Test
      return self.evaluationFunction(gameState)

    move_candidate = gameState.getLegalActions()
    scores = []
    value = float('-inf')

    for action in move_candidate:
      successorGameState = gameState.generatePacmanSuccessor(action)
      scores.append(self.minValue(successorGameState, currentDepth, 1, alpha, beta))

      value = max(scores)
      alpha = max(alpha, value)

      if value > beta: return value # pruning

    return value

  def minValue(self, gameState, currentDepth, agentIndex, alpha, beta):
    if self.isTerminal(gameState, currentDepth): # Terminal Test
      return self.evaluationFunction(gameState)

    move_candidate = gameState.getLegalActions(agentIndex)
    scores = []
    value = float('inf')
    ghostNum = gameState.getNumAgents() - 1

    for action in move_candidate:
      successorGameState = gameState.generateSuccessor(agentIndex, action)

      if agentIndex == ghostNum:
        nextDepth = currentDepth + 1
        scores.append(self.maxValue(successorGameState, nextDepth, alpha, beta))
      else:
        nextGhost = agentIndex + 1
        scores.append(self.minValue(successorGameState, currentDepth, nextGhost, alpha, beta))

      value = min(scores)
      beta = min(beta, value)

      if value < alpha: return value # pruning

    return value

    ##########################################################################
```

AlphaBetaAgent	
Action(self, gameState)	팩맨이 주어진 depth 안에서 점수를 최대화하는 움직임을 최종적으로 반환하는 함수이다. 1. 현재 state에서 팩맨이 이동할 수 있는 action을 취했을 때의 다음 state와 현재 depth인 0, 첫 번째 유령의 index인 1, 초기 alpha값인 -inf, 초기 beta값인 inf를 파라미터로 하여 minValue 함수를 호출하고 그 값을 저장한다. 2. 저장된 값들 중 가장 큰 값을 bestScore에 할당한다. 3. alpha와 value 중 더 큰 값을 alpha에 할당한다. 4. 1-3번의 과정을 팩맨이 이동할 수 있는 모든 action에 대해 반복한다. 5. bestScore의 움직임을 최종적으로 반환한다.

maxValue(self, gameState, currentDepth)	Max agent, 즉 팩맨이 현재 state에서 얻을 수 있는 점수들 중 가장 높은 점수를 반환하는 함수이다.
	1. 먼저 isTerminal 함수를 통해 현재 state에 대한 Terminal Test를 진행한다.
	  1. Terminal 상태라면, evaluationFunction 함수를 호출하여 그 값을 반환한다.
	  2. Terminal 상태가 아니라면 2번을 진행한다.
	2. 현재 state에서 팩맨이 이동할 수 있는 action을 취했을 때의 다음 state와 현재 depth인 currentDepth, 첫 번째 유령의 index인 1, alpha, beta를 파라미터로 하여 minValue 함수를 호출하고 그 값을 저장한다.
	3. 저장된 값들 중 가장 큰 값을 value 변수에 할당한다.
	4. alpha와 value 중 더 큰 값을 alpha에 할당한다.
	5. 만약 value의 값이 beta 값보다 크다면 value를 바로 반환한다. (Pruning 단계)
	6. 2-5번의 과정을 팩맨이 이동할 수 있는 모든 action에 대해 반복한다.
	7. value를 반환한다.

minValue(self, gameState, currentDepth, agentIndex)	Min agent, 즉 유령이 현재 state에서 얻을 수 있는 점수들 중 가장 낮은 점수를 반환하는 함수이다.
	1. 먼저 isTerminal 함수를 통해 현재 state에 대한 Terminal Test를 진행한다.
	  1. Terminal 상태라면, evaluationFunction 함수를 호출하여 그 값을 반환한다.
	  2. Terminal 상태가 아니라면 2번을 진행한다.
	2. 현재 state에서 현재 유령이 이동할 수 있는 action을 취했을 때의 다음 state를 구한다.
	3. 만약 현재 유령이 두 번째 유령(agentIndex=2)이라면, 모든 agent가 움직였으므로 다음 state와 다음 depth, alpha, beta를 파라미터로 하여 maxValue 함수를 호출하고 그 값을 저장한다.
	4. 현재 유령이 첫 번째 유령(agentIndex=1)이라면, 다음 state와 현재 depth인 currentDepth, 두 번째 유령의 index인 nextGhost(=2), alpha, beta를 파라미터로 하여 minValue 함수를 호출하고 그 값을 저장한다.
	5. 저장된 값들 중 가장 작은 값을 value 변수에 할당한다.
	6. beta와 value 중 더 작은 값을 beta에 할당한다.
	7. 만약 value의 값이 alpha 값보다 작다면 value를 바로 반환한다. (Pruning 단계)
	8. 2-7번의 과정을 유령이 이동할 수 있는 모든 action에 대해 반복한다.
	9. value를 반환한다.

isTerminal(self, gameState, currentDepth)	gameState와 currentDepth가 주어졌을 때, terminal 상태인지 아닌지를 반환하는 함수이다. 다음과 같은 상황일 때 terminal 상태로 판단하여 True를 반환한다. 1. currentDepth가 미리 지정된 depth와 같을 때 2. 팩맨이 이긴 상황일 때 3. 팩맨이 진 상황일 때
isPacman(self, agentIndex)	agentIndex가 주어졌을 때, 팩맨이라면(agentIndex=0) True를 반환하고 아니라면 False를 반환한다.

ExpectimaxAgent

```
class ExpectimaxAgent(AdversialSearchAgent):
  """
    [문제 03] Expectimax의 Action을 구현하시오. (25점)
    (depth와 evaluation function은 위에서 정의한 self.depth and self.evaluationFunction을 사용할 것.)
  """
  def Action(self, gameState):
    ######################## Write Your Code Here ################################
    move_candidate = gameState.getLegalActions()
    scores = []
    for action in move_candidate:
      successorGameState = gameState.generatePacmanSuccessor(action)
      scores.append(self.minValue(successorGameState, 0, 1))

    bestScore = max(scores)
    Index = [index for index in range(len(scores)) if scores[index] == bestScore]
    get_index = random.choice(Index)

    return move_candidate[get_index]

  def isTerminal(self, gameState, currentDepth):
    return True if currentDepth == self.depth or gameState.isWin() or gameState.isLose() else False

  def isPacman(self, agentIndex):
    return True if agentIndex == 0 else False

  def maxValue(self, gameState, currentDepth):
    if self.isTerminal(gameState, currentDepth): # Terminal Test
      return self.evaluationFunction(gameState)

    move_candidate = gameState.getLegalActions()
    scores = []

    for action in move_candidate:
      successorGameState = gameState.generatePacmanSuccessor(action)
      scores.append(self.minValue(successorGameState, currentDepth, 1))

    return max(scores)

  def minValue(self, gameState, currentDepth, agentIndex):
    if self.isTerminal(gameState, currentDepth): # Terminal Test
      return self.evaluationFunction(gameState)

    move_candidate = gameState.getLegalActions(agentIndex)
    scores = []
    ghostNum = gameState.getNumAgents() - 1

    for action in move_candidate:
      successorGameState = gameState.generateSuccessor(agentIndex, action)

      if agentIndex == ghostNum:
        nextDepth = currentDepth + 1
        scores.append(self.maxValue(successorGameState, nextDepth))
      else:
        nextGhost = agentIndex + 1
        scores.append(self.minValue(successorGameState, currentDepth, nextGhost))

    return float(sum(scores)/len(move_candidate))

    ##############################################################################
```

ExpectimaxAgent	
Action(self, gameState)	팩맨이 주어진 depth 안에서 점수를 최대화하는 움직임을 최종적으로 반환하는 함수이다. 1. 현재 state에서 팩맨이 이동할 수 있는 action을 취했을 때의 다음 state와 현재 depth인 0, 첫 번째 유령의 index인 1을 파라미터로 하여 minValue 함수를 호출하고 그 값을 저장한다. 2. 1번의 과정을 팩맨이 이동할 수 있는 모든 action에 대해 반복한다. 3. minValue 함수를 통해 반환된 값들 중 가장 높은 점수를 내는 움직임을 최종적으로 반환한다.

maxValue(self, gameState, currentDepth)	Max agent, 즉 팩맨이 현재 state에서 얻을 수 있는 점수들 중 가장 높은 점수를 반환하는 함수이다.
	1. 먼저 isTerminal 함수를 통해 현재 state에 대한 Terminal Test를 진행한다.
	  1. Terminal 상태라면, evaluationFunction 함수를 호출하여 그 값을 반환한다.
	  2. Terminal 상태가 아니라면 2번을 진행한다.
	2. 현재 state에서 팩맨이 이동할 수 있는 action을 취했을 때의 다음 state와 현재 depth인 currentDepth, 첫 번째 유령의 index인 1을 파라미터로 하여 minValue 함수를 호출하고 그 값을 저장한다.
	3. 2번의 과정을 팩맨이 이동할 수 있는 모든 action에 대해 반복한다.
	4. minValue 함수를 통해 반환된 값들 중 가장 높은 점수를 반환한다.

minValue(self, gameState, currentDepth, agentIndex)	Min agent, 즉 유령이 현재 state에서 얻을 수 있는 점수들 중 가장 낮은 점수를 반환하는 함수이다.
	1. 먼저 isTerminal 함수를 통해 현재 state에 대한 Terminal Test를 진행한다.
	  1. Terminal 상태라면, evaluationFunction 함수를 호출하여 그 값을 반환한다.
	  2. Terminal 상태가 아니라면 2번을 진행한다.
	2. 현재 state에서 현재 유령이 이동할 수 있는 action을 취했을 때의 다음 state를 구한다.
	3. 만약 현재 유령이 두 번째 유령(agentIndex=2)이라면, 모든 agent가 움직였으므로 다음 state와 다음 depth에 대한 maxValue 함수를 호출하고 그 값을 저장한다.
	4. 현재 유령이 첫 번째 유령(agentIndex=1)이라면, 다음 state와 현재 depth인 currentDepth, 두 번째 유령의 index인 nextGhost(=2)를 파라미터로 하여 minValue 함수를 호출하고 그 값을 저장한다.
	5. 2-4번의 과정을 유령이 이동할 수 있는 모든 action에 대해 반복한다.
	6. 저장된 값들의 합을 유령이 취한 action의 개수로 나눈 값을 반환한다. (Expectimax, 확률을 적용한 값 -> 확률적 행동을 가능하도록 함)

isTerminal(self, gameState, currentDepth)	gameState와 currentDepth가 주어졌을 때, terminal 상태인지 아닌지를 반환하는 함수이다. 다음과 같은 상황일 때 terminal 상태로 판단하여 True를 반환한다. 1. currentDepth가 미리 지정된 depth와 같을 때 2. 팩맨이 이긴 상황일 때 3. 팩맨이 진 상황일 때
isPacman(self, agentIndex)	agentIndex가 주어졌을 때, 팩맨이라면(agentIndex=0) True를 반환하고 아니라면 False를 반환한다.

2. 실행 화면

Minimax Agent 명령어의 승률 출력 화면

```
Win Rate: 65% (659/1000)
('Total Time:', 79.53952527046204)
('Average Time:', 0.07953952527046204)
========================================
```

time_check.py에서 출력된 실행시간 캡처 화면 (smallmap, mediummap, minimaxmap)

Minimax	AlphaBeta
Win Rate: 10% (32/300) ('Total Time:', 49.23086333274841) ('Average Time:', 0.16410287777582805) ========== ----- END MiniMax (depth=2) For Small Map	Win Rate: 12% (36/300) ('Total Time:', 40.85451030731201) ('Average Time:', 0.13618170102437338) ========== ----- END AlphaBeta (depth=2) For Small Map
Win Rate: 13% (39/300) ('Total Time:', 117.25644421577454) ('Average Time:', 0.39085481405258177) ========== ----- END MiniMax (depth=2) For Medium Map	Win Rate: 14% (42/300) ('Total Time:', 106.25668358802795) ('Average Time:', 0.35418894529342654) ========== ----- END AlphaBeta (depth=2) For Medium Map
Win Rate: 41% (410/1000) ('Total Time:', 51.93077778816223) ('Average Time:', 0.051930777788162234) ========== ----- END MiniMax (depth=4) For Minimax Map	Win Rate: 38% (386/1000) ('Total Time:', 41.231202363967896) ('Average Time:', 0.04123120236396789) ========== ----- END AlphaBeta (depth=4) For Minimax Map

Expectimax Agent 명령어의 승률 및 Score 출력 화면

```
----------Game Results----------
('Average Score:', 46.02)
('Score Results:', '-502, 532, 532, -502, -502, -502, 532, -502, -502, 532, -502, 53
2, -502, 532, -502, -502, 532, 532, 532, 532, 532, -502, -502, -502, 532, 532, 532,
532, 532, 532, -502, 532, 532, 532, -502, -502, -502, 532, 532, -502, 532, -502, 532
, -502, -502, 532, -502, -502, -502, 532, -502, -502, -502, -502, 532, -502, -502, 5
32, -502, -502, -502, 532, -502, 532, 532, 532, 532, 532, 532, 532, 532, 532, -502,
532, -502, -502, -502, -502, 532, 532, 532, -502, 532, 532, 532, 532, 532, -502, -50
2, -502, 532, 532, -502, 532, -502, 532, -502, 532, -502, 532')
('Record:', 'Lose, Win, Win, Lose, Lose, Lose, Win, Lose, Lose, Win, Lose, Win, Lose
, Win, Lose, Lose, Win, Win, Win, Win, Win, Lose, Lose, Lose, Win, Win, Win, Win, Wi
n, Win, Lose, Win, Win, Win, Lose, Lose, Lose, Win, Win, Lose, Win, Lose, Win, Lose,
 Lose, Win, Lose, Lose, Lose, Win, Lose, Lose, Lose, Lose, Win, Lose, Lose, Win, Los
e, Lose, Lose, Win, Lose, Win, Win, Win, Win, Win, Win, Win, Win, Win, Lose, Win, Lo
se, Lose, Lose, Lose, Win, Win, Win, Lose, Win, Win, Win, Win, Win, Lose, Lose, Lose
, Win, Win, Lose, Win, Lose, Win, Lose, Win, Lose, Win')

Win Rate: 53% (53/100)
('Total Time:', 0.4632425308227539)
('Average Time:', 0.004632425308227539)
=======================================
```