

Project-2: Multi-Agent Pacman

2015136107 이현진

컴퓨터공학부

2sguswls2s@koreatech.ac.kr

1 Introduction

Multi-Agent Search

이번 과제에서는 팩맨의 목적을 달성하는 프로그램을 작성하는데, 유령에 관한 내용을 디자인 할 것이다. 그 과정에서 minimax와 expectimax search를 구현하고, 설계를 할 것이다.

이번 프로젝트에서 작성해야 할 5가지의 클래스는 다음과 같다.

- ReflexAgent
- Minimax
- Alpha-Beta Pruning
- Expectimax
- Evaluation Function

이번 보고서에서 점수를 확인하는 방법은 다음과 같다.

```
python autograder.py
```

만약 2번 문제만 실행하고 싶다면 다음과 같이 실행한다.

```
python autograder.py -q q2
```

마지막으로 2번 문제 중 small-tree를 실행하고 싶을 때는 다음과 같이 실행한다.

```
python autograder.py -t test_cases/q2/0-small-tree
```

팩맨 파이썬 파일 설명 및 주의사항

Table 1. 팩맨 파이썬 파일 설명

파일명	설명
multiAgents.py	모든 멀티 에이전트 검색을 수행하는 곳.
Pacman.py	팩맨 게임을 실행하는 메인 파일, Pacman GameState 이용 시 사용한다.
Game.py	AgentState, Agent, Direction 등 함수의 원형을 볼 수 있는 곳.
Util.py	검색 알고리즘을 구현하는데 유용한 데이터 구조를 제공함.
graphicsDisplay.py	팩맨의 그래픽 코드
graphicsUtils.py	팩맨의 그래픽 코드 지원
textDisplay.py	아스키 그래픽 코드
ghostAgents.py	유령 관련 코드
keyboardAgents.py	키보드 인터페이스
layout.py	레이아웃 코드
autograder.py	프로젝트 검사 코드
testParser.py	Autograder.py 솔루션 파일 구문 분석
testClasses.py	프로젝트 검사 코드
test_cases/	질문에 대한 테스트 사례가 포함된 디렉터리
searchTestClasses.py	코드 등급 클래스

이번 과제는 multiAgent.py 파일을 채우는 과제이다. multiagent.py 코드 내에 있는 "YOUR CODE HERE" 부분을 스스로 짜면 된다.

주의사항으로는 지난 보고서와 동일하게 multiagent.py 내 클래스의 이름, 함수의 이름을 변경하지 않도록 주의한다.

2 Questions

2.1 Reflex Agent

해결방법

이 문제를 해결하기 위해서는 팩맨이 어떻게 이길 수 있는지, 지는 방법이 무엇인지 고민해보았는데, 이기기 위해서는 모든 음식을 먹어야하고, 지기 위해서는 유령에 닿으면 죽는 것을 생각해 보았다. 그래서 음식의 모든 좌표를 찾고, 맨해튼 거리를 이용하여 거리값을 구한 후, 배열에 넣어주었고, 모든 좌표에 대해서 가중치를 부여하는데, 문제에서 중요한 일이라고 생각되는 것을 점수를 부여할 때, 역수를 취하면 점수를 크게 줄수록 점수가 줄어들어 최악의 점수를 받고 그쪽으로 움직이지 않도록 할 수 있다는 생각을 하였다.

모든 음식에 대해서 음식 거리 중 제일 적은 값을 주고, 역수를 취해줌으로써 가장 우선순위로 먹을 수 있도록 하였고, 유령의 좌표를 찾아 맨해튼 거리를 사용하여 유령의 대략적인 위치를 파악하고, 해당 유령에서 임계값을 주어(여기서는 2를 주었음) 2 미만인 경우 점수를 크게 주고, 역수를 주어 유령이 가까이 있을 경우 먹이를 먹는 것보다 유령을 피하는데 우선적으로 신경 쓸 수 있도록 코드를 작성하였다.

수도코드

ReflexAgent - evaluationFunction

Algorithm evaluationFunction

Input : currentGameState, action

Output : Int

Int score

Turple PacmanPosition

list GhostPosition

list FoodPositions

list FoodDistance

Turple GhostDistance

score = 0

for GhostPosition in range(GhostPosition) **do**

GhostDistance = abs(PacmanPosition – GhostPosition)

If GhostDistance < 2 **then**

Score += -20000

endif

endfor

for FoodPosition in FoodPositions **do**

FoodDistance = abs(PacmanPosition – FoodPosition)

endfor

score += -1 * min(FoodDistance)

return score

수도코드 설명

1. for문

1. 유령의 위치를 모아둔 리스트 GhostPosition 내에 있는 x, y 좌표를 이용하여 팩맨을 기준으로 한 맨해튼 거리 GhostDistance를 구한다.

2. if-then 문

1. 현 GhostDistance 가 2 미만일 경우, 유령이 가까이 있는경우 이므로 점수를 많이 깎아 피할 수 있도록 한다.

2. for문

1. 음식의 위치를 모아둔 리스트 FoodPositions 내에 있는 FoodPosition을 이용하여 팩맨을 기준으로 한 맨해튼 거리를 리스트 FoodDistance에 모아놓는다.

3. FoodDistance 리스트에서 가장 작은 값을 모든 음식의 점수로 결정한다.

4. score 를 반환한다.

Grading code 결과

```

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python autograder.py -q q1 --no-graphics
Starting on 5-3 at 16:16:28

Question q1
=====

Pacman emerges victorious! Score: 1238
Pacman emerges victorious! Score: 1244
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1240
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1237
Pacman emerges victorious! Score: 1238
Pacman emerges victorious! Score: 1247
Pacman emerges victorious! Score: 1238
Pacman emerges victorious! Score: 1242
Average Score: 1240.2
Scores:      1238.0, 1244.0, 1239.0, 1240.0, 1239.0, 1237.0, 1238.0, 1247.0, 1238.0, 1242.0
Win Rate:    10/10 (1.00)
Record:      Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/q1/grade-agent.test (4 of 4 points)
***      1240.2 average score (2 of 2 points)
***      Grading scheme:
***          < 500: 0 points
***          >= 500: 1 points
***          >= 1000: 2 points
***      10 games not timed out (0 of 0 points)
***      Grading scheme:
***          < 10: fail
***          >= 10: 0 points
***      10 wins (2 of 2 points)
***      Grading scheme:
***          < 1: fail
***          >= 1: 0 points
***          >= 5: 1 points
***          >= 10: 2 points

### Question q1: 4/4 ###

Finished at 16:16:30

Provisional grades
=====
Question q1: 4/4
-----
Total: 4/4

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

```

Fig. 1. Reflex Agent Grading code 결과

결과 및 분석

```

PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py -p ReflexAgent -l testClassic
Pacman emerges victorious! Score: 564
Average Score: 564.0
Scores: 564.0
Win Rate: 1/1 (1.00)
Record: Win
PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py --frameTime 0 -p ReflexAgent -k 1
Pacman emerges victorious! Score: 1251
Average Score: 1251.0
Scores: 1251.0
Win Rate: 1/1 (1.00)
Record: Win
PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py --frameTime 0 -p ReflexAgent -k 2
Pacman died! Score: -40
Average Score: -40.0
Scores: -40.0
Win Rate: 0/1 (0.00)
Record: Loss
PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py --frameTime 0 -p ReflexAgent -k 2
Pacman emerges victorious! Score: 1883
Average Score: 1883.0
Scores: 1883.0
Win Rate: 1/1 (1.00)
Record: Win
PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py -p ReflexAgent -l testClassic
Pacman emerges victorious! Score: 562
Average Score: 562.0
Scores: 562.0
Win Rate: 1/1 (1.00)
Record: Win
PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py --frameTime 0 -p ReflexAgent -k 1
Pacman emerges victorious! Score: 790
Average Score: 790.0
Scores: 790.0
Win Rate: 1/1 (1.00)
Record: Win

```

Fig. 2. Reflex Agent 결과

코드를 작성한 후, 수행 해본 결과 하나의 유령의 경우 잘되었지만, 2개의 유령이 생겼을 경우 죽는 일이 발생하였다. 이는 유령이 가까있을 때, 평가 score를 깎는데 있어서 너무 많은 값을 깎아서 생기게 된 문제인데, 유령에 피하는데 너무 많은 값을 주게 된다면, 유령의 개수가 많아졌을 때 유령을 피하는데만 집중하고, 음식을 먹지 않는 사태가 발생한다. 따라서 유령에 대해서 적정한 값을 주어 평가를 잘 할 수 있도록 만들게 하여 문제를 해결하였다.

느낀 점

문제를 해결하면서 배울 수 있었던 점은 팩맨이 움직일 때, 어디로 갔을 때 평가 점수가 제일 큰지 예상하여 이동하는 것이므로 해당 평가점수를 부여하는데 있어서 다양한 경우를 제시해주어야 팩맨이 원활하게 평가하여 문제를 해결한다는 점을 배웠다. 하지만 너무 많은 경우의 수를 봤을 경우 평가 점수의 합리적임이 흐려져 믿지못하는 평가 알고리즘이 될 수 있다는 점을 배웠다.

2.2 Minimax

해결방법

이 문제를 해결하는데 있어서 가장 중요한 미니맥스 알고리즘을 이해 해야 한다. 미니맥스 알고리즘의 가장 중요한 점이 2가지 존재하는데, 각 플레이어들은 합리적으로 움직여야하고, 한 쪽이 이득을 보면 한 쪽은 손해를 봐야한다. 팩맨의 경우에는 이득을 보고, 유령의 경우 손해를 보도록 코드를 구현하는 것이 미니맥스 알고리즘의 목표라고 할 수 있다.

각 팩맨과 유령은 번갈아가면서 이동을 결정하는 것이기 때문에, 팩맨의 경우 가장 점수가 큰 값인 max 부분을 줘야하고, 유령에게는 가장 적은 값인 min 값을 리턴해줌으로써 최종목표(Terminal)에 도달할 수 있도록 해야한다. 여기서 max와 min 값을 결정하는 것은 Utility의 값 또한 중요하다. Utility는 각각의 쉘들의 값이라고 할 수 있다. 터미널에 있는 값도 일종의 Utility라고 할 수 있다. 해당 Utility 값에 따라서 유령은 최소값을 선택함으로써 손해를 보고, 이득을 챙길 수 있는 구간을 팩맨이 결정할 수 있다는 점에서 중요하다고 생각한다.

코드를 구현하기 위해서 PDF 파일에 있는 수도코드를 참조하여 각각 3개의 함수(max-value, min-value, value)를 작성하였고, 액션의 부분에서 최종적으로는 max가 맨 위에 있기 때문에 getAction 함수에는 최종적으로 유령이 움직인 값인 minValue를 실행하도록 하여 값을 받아온 후, max 함수와 똑같이 수행하도록 코드를 작성 하였다.

수도코드

MinimaxAgent - getAction

Algorithm Minimax

Input : GameState

Output : Action

Function Value(gameState, currentDepth, agentIndex)

If state is terminal state or Game is Win or Game is Lose **then**

return state's utility

endif

if agentIndex is Pacman **then**

return max-Value(gameState, currentDepth, agentIndex)

endif

```

if agentIndex is Ghost then
    return min-Value(gameState, currentDepth+1, agentIndex)
endif

```

```

Function max-Value(gameState, currentDepth, agentIndex)
    maxValue = -infinity
    for action in PacmanGameState do
        maxValue = max(maxValue, Value(gameState, currentDepth, nextGhost))
    endfor
    return maxValue

```

```

Function min-Value(gameState, currentDepth, agentIndex)
    minValue = infinity
    for action in GhostGameState do
        if agentIndex is LastGhost then
            minValue = min(minValue, Value(gameState, currentDepth+1, Pacman))
        else
            minValue = min(minValue, Value(gameState, currentDepth, nextGhost))
        endif
    endfor
    return minValue

```

```

maxValue = -infinity
maxAction = STOP

```

```

for action in PacmanGameState do
    compareValue = Value(nextState, depth = 0, FirstGhost)
    if maxValue < compareValue then
        maxValue = compareValue
        maxAction = action
    endif
endfor
return maxAction

```

수도코드 설명

1. Function Value 부분

1. if-then 문

1. 현재 위치가 최종 도착위치이거나, 이기거나, 진 상황이면 해당 구역의 유틸리티를 반환함.

2. if-then 문

1. 현재 팩맨의 상태이면, max-Value 함수를 실행함.

2. 현재 유령의 상태이면, min-Value 함수를 실행함.

2. max-Value 부분

1. max-Value 를 $-\infty$ 로 설정함.

2. for 문

1. 팩맨의 상태에 따라 수행할 수 있는 액션들 수만큼 반복

2. max-Value 는 기존의 max-Value와 모든 유령의 min값과 비교하여 가장 큰 값을 넣음.

3. max-Value를 반환

3. min-Value 부분

1. min-Value를 ∞ 로 설정함.

2. for문

1. 유령의 상태에 따라 수행할 수 있는 액션들 수만큼 반복

2. if-then-else 문

1. 현재 유령이 마지막 유령이면, min-Value 값은 다음 깊이에 있는 팩맨의 값과 비교하여 작은 값으로 변경시킴.

2. 그렇지 않다면, min-Value의 값은 다음 유령의 min 값과 비교하여 작은 값으로 변경됨.

3. min-Value를 반환.

4. max-Value값은 $-\infty$, maxAction의 값은 STOP을 지정해줌.

5. for문

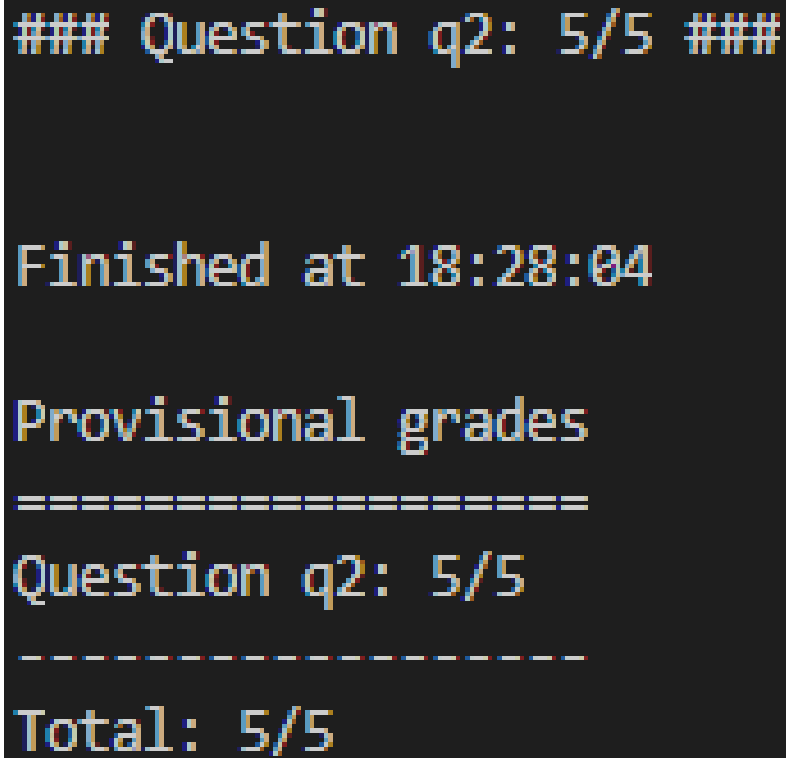
1. 현재 팩맨의 상태에 따른 액션들 수만큼 반복

2. 현재 팩맨의 다음 순서인 유령들에 대한 Value 함수 수행시켜 compareValue를 구함. 이를 통해 유령들의 유틸리티 값을 얻을 수 있음.

3. if – then 문

1. 유령들의 유틸리티 값이 max-Value 보다 크면 max-Value를 유령들 최종 유틸리티 값 compareValue로 바꿈.

2. maxAction 또한 현재 액션(action)으로 변경함

Grading code 결과

```
### Question q2: 5/5 ###  
  
Finished at 18:28:04  
  
Provisional grades  
=====
```

Question	Score
Question q2	5/5

```
-----  
Total: 5/5
```

Fig. 3. Minimax Grading code 결과

결과 및 분석

```

PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:      516.0
Win Rate:    1/1 (1.00)
Record:      win
PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
Pacman died! Score: -501
Average Score: -501.0
Scores:      -501.0
Win Rate:    0/1 (0.00)
Record:      Loss

```

Fig. 4. Minimax 결과

코드를 작성한 후, 수행 해본 결과 위의 코드의 경우는 성공할 때도 있고, 실패하기도 할 수 있는 코드를, 마지막 코드의 경우는 죽을 수 밖에 없는 상황을 재현한 것이다. 이를 통해서 minimax의 경우 팩맨이 이득을 챙길 수 없을 때에는 zerosum에 의해서 팩맨이 손해를 보고, 유령이 이득을 보는 상황이 생겼다. 또한 팩맨과 유령의 이동이 합리적이여야 하기 때문에 순간이동 하는 그러한 장면은 당연히 볼 수 없었고, 결국 이득을 챙길 수 없는 팩맨이 스스로 죽음을 선택한다는 것을 알 수 있었다. 이를 통해 앞에서 얘기한 minimax에서 중요한 2가지 중 zerosum에 대해 다시 한번 상기할 수 있는 결과였다.

느낀 점

이 문제를 풀면서 알 수 있었던 점은 실제로 많이 사용되고 있다고 생각이 들었는데, 흔히 바둑이나 장기 등에서 사용되는 장기도사 등 컴퓨터 프로그램이 생각 났다. 이 프로그램들이 이러한 알고리즘으로 제작 되었을 것이라 생각이 들었고, 이러한 알고리즘을 배워 실제로 응용해볼 수 있었다는 점에서 재미있었다. 그리고 어려웠던 점은 해당 알고리즘을 작성하는데 min-Value에서 마지막 유령인지 검사를 해주어야 했던 점이 존재했는데, 문제 1번처럼 유령이 하나만 존재하는 것이 아니라는 것을 잊고 생각했던 점이 있었다.

2.3 Alpha-Beta Pruning

해결방법

이 문제를 해결하기 위해서는 알파-베타 프루닝 알고리즘을 이해해야 하는데, 앞에 나와있는 프루닝의 뜻이 가지치기이다. 즉, minimax에서는 모든 경우의 터미널과 유틸리티를 검사했다면, 알파-베타 프루닝에서는 알파와 베타를 부여하여 (이때, 알파의 값은 최대값, 베타의 값은 최소값을 의미하는데

최대값은 노드에 들어갈수록 값이 갱신되어 최대한 이득을 볼 수 있는 유틸리티 값을 의미하고, 최소값은 노드에 들어갈수록 값이 갱신되면서 최대한 손해를 보는 유틸리티를 의미함.) 지속적으로 값을 팩맨과 유령의 상태에 값을 전달해주어 검사하지 않아도 되는 부분이 있다면 검사하지 않는 것을 의미한다.

예를 들어서 팩맨의 순서일 때, 왼쪽 노드가 100, 0이고, 오른쪽 노드에는 10, 5가 있을 경우, 왼쪽의 min 값은 0이 나올것이고, 오른쪽에는 5가 선택 될 것이다. 이 때, 팩맨이 결정을 수행할 때, 당연히 오른쪽 노드에 5가 최대값으로 선택이 될 것이고, 왼쪽 노드는 0이 바뀌지 않는 이상 100을 볼 필요가 없기 때문에 가지를 치는 것을 의미한다. 이 때, 알파는 max값이 베타보다 큰지 확인을 하고, 알파를 수정하고, 베타는 min값이 alpha보다 작아야 베타를 수정하여 결정한다.

결론적으로 베타값이 수정되면, 알파값을 수정하기 위해서 현재 max 값이 베타보다 큰지 확인을 해야한다. 크다면, 알파를 변경하지 않고, max 값을 반환하고, 작으면 알파를 변경하여 다음 노드에 알파값을 전달할 때 갱신된 알파값을 보내어 검사해도 되는 노드인지 검사를 수행한다.

수도코드

Alpha-Beta Pruning - getAction

Algorithm Alpha-Beta Pruning

Input : GameState

Output : Action

Function Value(gameState, currentDepth, agentIndex, alpha, beta)

If state is terminal state or Game is Win or Game is Lose **then**

 return state's utility

endif

if agentIndex is Pacman **then**

 return max-Value(gameState, currentDepth, agentIndex, alpha, beta)

endif

if agentIndex is Ghost **then**

 return min-Value(gameState, currentDepth+1, agentIndex, alpha, beta)

endif

Function max-Value(gameState, currentDepth, agentIndex, alpha, beta)

 maxValue = -infinity

for action in PacmanGameState **do**

 maxValue = max(maxValue, Value(gameState, currentDepth, FirstGhost, alpha, beta))

enddo

```

    if maxValue > beta then
        return maxValue
    endif
    alpha = max(alpha, maxValue)
endfor
return maxValue

Function min-Value(gameState, currentDepth, agentIndex, alpha, beta)
    minValue = infinity
    for action in GhostGameState do
        if agentIndex is LastGhost then
            minValue = min(minValue, Value(gameState, currentDepth+1, Pacman, alpha, beta))
            if minValue < alpha then
                return minValue
            endif
            beta = min(beta, minValue)
        else
            minValue = min(minValue, Value(gameState, currentDepth, nextGhost, alpha, beta))
            if minValue < alpha then
                return minValue
            endif
            beta = min(beta, minValue)
        endif
    endfor
    return minValue

maxValue = -infinity
maxAction = STOP
alpha = -infinity
beta = infinity

for action in PacmanGameState do
    compareValue = Value(nextState, depth = 0, FirstGhost, alpha, beta)
    if maxValue < compareValue then
        maxValue = compareValue
        maxAction = action
        alpha = FinalValue
    endif
endfor
return maxAction

```

수도코드 설명

1. Function Value 부분

1. if-then 문

1. 현재 위치가 최종 도착위치이거나, 이기거나, 진 상황이면 해당 구역의 유틸리티를 반환함.

2. if-then 문

1. 현재 팩맨의 상태이면, max-Value 함수를 실행함.
2. 현재 유령의 상태이면, min-Value 함수를 실행함.

2. max-Value 부분

1. max-Value 를 -infinity로 설정함.

2. for 문

1. 팩맨의 상태에 따라 수행할 수 있는 액션들 수만큼 반복
2. max-Value 는 기존의 max-Value와 모든 유령의 min값과 비교하여 가장 큰 값을 넣음.

3. if-then 문

1. max-Value 가 beta보다 크면 바로 max-Value를 반환함.
 4. 알파의 값을 max-Value와 비교하여 둘 중 큰 값으로 알파값을 변환
3. max-Value를 반환

3. min-Value 부분

1. min-Value를 infinity로 설정함.

2. for문

1. 유령의 상태에 따라 수행할 수 있는 액션들 수만큼 반복

2. if-then-else 문

1. 현재 유령이 마지막 유령이면, min-Value 값은 다음 깊이에 있는 팩맨의 값과 비교하여 작은 값으로 변경시킴.

2. if-then 문

1. min-Value 가 alpha보다 작으면 바로 min-Value를 반환함.

3. 베타의 값을 min-Value와 비교하여 둘 중 큰 값으로 베타값을 변환
-

4. 그렇지 않다면, `minValue`의 값은 다음 유형의 `min` 값과 비교하여 작은 값으로 변경됨.

5. if-then 문

1. `minValue` 가 `alpha`보다 작으면 바로 `minValue`를 반환함.

6. 베타의 값을 `minValue`와 비교하여 둘 중 큰 값으로 베타값을 변환

3. `minValue`를 반환.

4. `maxValue`값은 `-infinity`, `maxAction`의 값은 `STOP`, `alpha`는 `-infinity`, `beta`는 `infinity`를 지정해줌.

5. for문

1. 현재 팩맨의 상태에 따른 액션들 수만큼 반복

2. 현재 팩맨의 다음 순서인 유형들에 대한 `Value` 함수 수행시켜 `compareValue`를 구함. 이를 통해 유형들의 유틸리티 값을 얻을 수 있음.

3. if – then 문

1. 유형들의 유틸리티 값이 `maxValue` 보다 크면 `maxValue`를 유형들 최종 유틸리티 값 `compareValue`로 바꿈.

2. `maxAction` 또한 현재 액션(action)으로 변경하고, 알파의 값 또한 `compareValue`로 수정함.

6. `maxAction`을 반환시킴

Grading code 결과

```

### Question q3: 5/5 ###

Finished at 19:07:27

Provisional grades
=====
Question q3: 5/5
-----
Total: 5/5

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

```

Fig. 5. Alpha-Beta Pruning Grading code 결과

결과 및 분석

```

PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
Pacman emerges victorious! Score: 1148
Average Score: 1148.0
Scores: 1148.0
Win Rate: 1/1 (1.00)
Record: Win

```

Fig. 6. Alpha-Beta Pruning 결과

Alpha-Beta Pruning을 구현해본 결과, Adversarial Ghost에 대해서 완벽하게 이기는 방법을 제시하는 것을 확인할 수 있었다. 또한, minimax와 다르게 minimax에서는 모든 노드를 검사했다면, Alpha-Beta Pruning에서는 시간을 조금 더 절약해 해결해주는 느낌이 있었다. 실제로 depth가 3인 Alpha-Beta Pruning의 속도와 depth가 2인 minimax 와 유사한 속도를 확인 할 수 있었다.

느낀 점

Alpha-Beta Pruning를 하면서 알 수 있었던 점은 실제의 minimax에서 가지치기를 함으로써 시간복잡도를 줄이는 알고리즘임을 알 수 있었다. minimax에서 alpha, beta의 추가만으로 이렇게 시간을 단축 시킬 수 있었다는 점에서 매우 흥미로웠던 알고리즘이었다. 하지만, Adversarial Ghost에 대해서는 적절하게 대응하지만, Random Ghost 에서도 시간을 단축시키는데는 한계가 있을수밖에 없다는 점에 아쉬웠다.

2.4 Expectimax

해결방법

이 문제를 해결하기 위해서는 Expectimax 알고리즘을 알아야 하는데, Expectimax 알고리즘은 유틸리티의 값을 온전히 가져오지 않고, 확률의 개념을 넣어서 왼쪽에서 A정도 나오고, 오른쪽에서 B정도 나온다면 A와 B의 사이값을 min으로 지정한다는 것이 있다. 쉽게 생각하면 상대방의 정확한 위치를 minimax로 하는 것이지만, 엄격하게 계산된 움직임이 아닌, 적절한 움직임을 수행하는 것이다. 따라서 이쪽으로 갔을 때 확률적으로 이득이 될 거 같다고 생각을 하고 움직이기 때문에 엄격하지 않다는 점이 있다.

이를 통해서 얻을 수 있는 이점은 위의 minimax나 alpha-beta pruning에서는 엄격한 계산을 요구로 하여 Random으로 움직이는 유령의 움직임에 대해서도 의미있게 보기 때문에 쓸모없는 계산을 수행할 수 있다. 이는 곧 비관주의가 되어 의미없는 움직임을 만들게 할 수 있다. 하지만 Expectimax를 사용하게 된다면 유령이 느슨하게 움직이면 팩맨도 계산을 느슨하게 적용해서 의미없는 움직임을 계산에 두지 않고 목표를 향해 움직임으로써 점수를 쌓을 수 있다는 점이 있다. 하지만 너무 낙천적으로 코드를 짜게 된다면 팩맨은 이길 수 없다는 점을 알고 확률을 통해 계산하여 적절한 행동을 수행한다.

수도코드

Expectimax - getAction

Algorithm Expectimax

Input : GameState

Output : Action

Function Value(gameState, currentDepth, agentIndex)

If state is terminal state or Game is Win or Game is Lose **then**
 return state's utility

endif

if agentIndex is Pacman **then**

return max-Value(gameState, currentDepth, agentIndex)

endif

if agentIndex is Ghost **then**

return Exp-Value(gameState, currentDepth+1, agentIndex)

endif

Function max-Value(gameState, currentDepth, agentIndex)

maxValue = -infinity

```

for action in PacmanGameState do
    maxValue = max(maxValue, Value(gameState, currentDepth, nextGhost))
endfor
return maxValue

Function Exp-Value(gameState, currentDepth, agentIndex)
    ExpValue = 0
    for action in GhostGameState do
        if agentIndex is LastGhost then
            probability = length of Ghost Actions
            ExpValue = ExpValue + Value(gameState, currentDepth+1, Pacman) / proba-
bilty
        else
            probability = length of Ghost Actions
            ExpValue = ExpValue + Value (gameState, currentDepth, nextGhost) / proba-
bilty
        endif
    endfor
    return minValue

    maxValue = -infinity
    maxAction = STOP

    for action in PacmanGameState do
        compareValue = Value(nextState, depth = 0, FirstGhost)
        if maxValue < compareValue then
            maxValue = compareValue
            maxAction = action
        endif
    endfor
    return maxAction

```

수도코드 설명

1. Function Value 부분

1. if-then 문

1. 현재 위치가 최종 도착위치이거나, 이기거나, 진 상황이면 해당 구역의 유틸리티를 반환함.

2. if-then 문

1. 현재 팩맨의 상태이면, max-Value 함수를 실행함.
 2. 현재 유령의 상태이면, min-Value 함수를 실행함.
-

2. max-Value 부분

1. maxValue 를 -infinity로 설정함.

2. for 문

1. 팩맨의 상태에 따라 수행할 수 있는 액션들 수만큼 반복

2. maxValue 는 기존의 maxValue와 모든 유령의 min값과 비교하여 가장 큰 값을 넣음.

3. maxValue를 반환

3. Exp-Value 부분

1. ExpValue를 0으로 설정함.

2. for문

1. 유령의 상태에 따라 수행할 수 있는 액션들 수만큼 반복

2. if-then 문

1. 확률 probability를 구하는데, 활동 가능한 액션의 수에 따라서 해당 유틸리티로 이동할 확률이 변동되기 때문에 활동가능한 액션의 수를 확률로 지정함.

2. 현재 유령이 마지막 유령이면, ExpValue 값은 다음 깊이에 있는 팩맨의 값을 확률로 나눠준 값을 합쳐준다.

3. 그렇지 않다면, ExpValue의 값은 다음 유령의 min값을 확률로 나눠준 값을 합쳐준다.

3. ExpValue를 반환.

4. maxValue값은 -infinity, maxAction의 값은 STOP을 지정해줌.

5. for문

1. 현재 팩맨의 상태에 따른 액션들 수만큼 반복

2. 현재 팩맨의 다음 순서인 유령들에 대한 Value 함수 수행시켜 compareValue를 구함. 이를 통해 유령들의 유틸리티 값을 얻을 수 있음.

3. if – then 문

-
1. 유령들의 유틸리티 값이 `maxValue` 보다 크면 `maxValue`를 유령들 최종 유틸리티 값 `compareValue`로 바꿈.
 2. `maxAction` 또한 현재 액션(action)으로 변경함
 6. `maxAction`을 반환시킴
-

Grading code 결과

```

### Question q4: 5/5 ###

Finished at 19:50:23

Provisional grades
=====
Question q4: 5/5
-----
Total: 5/5

Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

Fig. 7. Expectimax Grading code 결과

결과 및 분석

```

PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
Pacman emerges victorious! Score: 513
Average Score: 513.0
Scores: 513.0
Win Rate: 1/1 (1.00)
Record: Win
PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Average Score: -501.0
Scores: -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0
Win Rate: 0/10 (0.00)
Record: Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss
PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Average Score: -295.2
Scores: -502.0, -502.0, 532.0, 532.0, -502.0, -502.0, -502.0, -502.0, -502.0, -502.0
Win Rate: 2/10 (0.20)
Record: Loss, Loss, Win, Win, Loss, Loss, Loss, Loss, Loss, Loss

```

Fig. 8. Expectimax 전체 결과

```

PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Average Score: 221.8
Scores: 532.0, 532.0, 532.0, 532.0, -502.0, 532.0, -502.0, 532.0, -502.0, 532.0
Win Rate: 7/10 (0.70)
Record: Win, Win, Win, Win, Loss, Win, Loss, Win, Loss, Win

```

Fig. 9. Expectimax 중 random Ghost 결과

Expectimax 알고리즘을 실행해본 결과 alpha-beta pruning에서 수행되는 adversarial 유령상태를 수행할 때는 죽는 경우가 많이 생겼고, Random 유령상태인 경우 실제로 이길 수도 있고, 질 수도 있다는 것을 볼 수 있었다. 또한 점수가 alpha-beta pruning보다 잘 나오는 경우가 생기기도 한다는 점을 관찰 할 수 있었다. 이를 통해 느슨하게 수행했을 때 좋은 결과가 나올 수 있다는 것을 알 수 있었다.

느낀 점

위 과제를 하면서 알 수 있었던 점은 컴퓨터에서도 과학적으로 설명하기 힘든 부분이 있다는 것을 알 수 있었다. 확률적으로 도박같이 움직이는데,

결과가 엄격하게 계산된 것보다 좋게 나올 수 있다는 점이 뭔지는 알겠는데 설명하기는 힘든 부분인 것 같았다. 결과를 보고 매우 흥미롭게 느꼈다.

2.5 Evaluation Function

해결방법

이 문제를 해결하기 위해서 2.1의 Reflex Agent의 기본적인 내용에서 보강하는 방법을 이용하였다. 기본적인 기준에서 조금 더 엄격하게 유틸리티 값을 부여하는 방법을 선택함으로써 이 문제를 해결하였다.

그리고 역수로 부여한 까닭으로는 위의 2.1의 Reflex Agent에서 구현할 때 알 수 있었는데, 역수로 점수를 부여하면 가장 낮은 값을 주는 값에 가장 신경을 쓰는 것을 발견하여 점수에서 역수로 주는 방법을 이용하게 되었다. 또한 두려운 유령과 활성화 유령을 따로 분리한 이유는 파워를 먹고나서도 변하지 않은 유령들을 이전 과제에서 발견하였기 때문에 분리하여 따로따로 계산해주었다. 그리고 거리에 관한 것도 맨해튼 거리를 이용하여 가중치 부여를 다르게 했다는 점도 존재한다.

이를 통해 2.1의 문제에서 보완하여 각각의 유틸리티를 부여한 코드가 완성되었다.

수도코드

Evaluation Function - betterEvaluationFunction

Algorithm Evaluation Function

Input : currentGameState

Output : Action

List GhostList

List scaredGhostlist

List activeGhostlist

List FoodList

List PowerList

List FoodDistanceList

List PowerDistanceList

List ActiveGhostDistanceList

List ScaredGhostDistanceList

Int score = currentGameState.getScore()

if currentGameState is Win then

 return infinity

endif

```

if currentGameState is Lose then
    return -infinity
endif

for ghost in GhostList do
    if ghost is scared then
        scaredGhostlist.append(ghost)
    else
        activeGhostlist.append(ghost)
    endif
endfor

for food in Foodlist do
    foodDistance = abs(currentGameState.getPacmanPosition() – foodPosition())
    FoodDistancelist.append(foodDistance)
endfor

for power in Powerlist do
    powerDistance = abs(currentGameState.getPacmanPosition() – powerPosition())
    PowerDistancelist.append(powerDistance)
endfor

for activeghost in activeGhostlist do
    activeGhostDistance = abs(currentGameState.getPacmanPosition() – ac-
    tiveghost.position())
    ActiveGhostDistancelist.append(activeGhostDistance)
endfor

for scaredghost in ScaredGhostlist do
    scaredGhostDistance = abs(currentGameState.getPacmanPosition() –
    scaredghost.position())
    ScaredGhostDistancelist.append(scaredGhostDistance)
endfor

score += -10 * length of FoodDistancelist
score += -20 * length of PowerDistancelist

if pacman’s nextPosition is Wall then
    score += -infinity
endif

for food in FoodDistancelist do
    if food < 2 then
        score += -1 * food
    elif food < 7 then
        score += -0.5 * food
    else

```

```

        score += -0.3 * food
    endfor

    for power in PowerDistancelist do
        if power < 2 then
            score += -1 * power
        elif power < 7 then
            score += -0.5 * power
        else
            score += -0.3 * power
        endfor

    for ghost in ScaredGhostDistancelist do
        if ghost < 3 then
            score += -20 * ghost
        else
            score += -10 * ghost
        endfor

    for ghost in ActiveGhostDistancelist do
        if ghost < 3 then
            score += -3 * ghost
        elif ghost < 7 then
            score += -2 * ghost
        else
            score += -1 * ghost
        endfor

    return score

```

수도코드 설명

1. if - then부분
 1. 만약 게임이 승리한다면 infinity를 부여
 2. if - then부분
 1. 만약 게임이 진다면 -infinity를 부여
 3. for 부분
 1. 유령 리스트의 수만큼 반복수행
 2. if – then - else 문
 1. 만약 유령상태가 두려워하는 상태이면, 두려워하는 유령 리스트에 넣음
 2. 그게 아니라면 활성화된 유령 리스트에 넣음
-

4. for 부분

1. 음식 리스트의 수만큼 반복수행
2. 맨해튼 거리를 이용하여 팩맨과 음식의 거리를 계산함.
3. 음식거리리스트에 계산된 값을 넣어줌

5. for 부분

1. 파워 리스트의 수만큼 반복수행
2. 맨해튼 거리를 이용하여 팩맨과 파워의 거리를 계산함.
3. 파워거리리스트에 계산된 값을 넣어줌

6. for 부분

1. 활성화된 유령 리스트의 수만큼 반복수행
2. 맨해튼 거리를 이용하여 팩맨과 활성화된 유령의 거리를 계산함.
3. 활성화된 유령 거리리스트에 계산된 값을 넣어줌

7. for 부분

1. 두려운 유령 리스트의 수만큼 반복수행
2. 맨해튼 거리를 이용하여 팩맨과 두려운 유령의 거리를 계산함.
3. 두려운 유령 거리리스트에 계산된 값을 넣어줌

8. 점수에 음식의 개수와 파워의 개수에 따라서 값을 부여함

9. if - then부분

1. 만약 팩맨의 다음 위치가 벽이라면, -infinity를 부여함

10. for 부분

1. if - then - else 문

1. 만약 음식 위치가 3이내이면 점수 적게 부여, 7이면 중간 점수 부여
2. 7이상이면 우선도 크게 부여함.

11. for 부분

1. if - then - else 문

1. 만약 파워 위치가 3이내이면 점수 적게 부여, 7이면 중간 점수 부여
-

2. 7이상이면 점수 적게 부여함.

12. for 부분

1. if - then - else 문

1. 두려운 유령 위치가 3이내이면 점수 적게 부여
2. 거리가 3이 아니어도 상당히 점수 적게 부여함.

13. for 부분

1. if - then - else 문

1. 활성화 유령 위치가 3이내이면 점수 적게 부여
 2. 7이내의 경우도 음식보다 점수적게 부여
 3. 7이후에도 가까운 음식수준의 점수 부여
-

Grading code 결과

```
### Question q5: 6/6 ###

Finished at 21:15:29

Provisional grades
=====
Question q5: 6/6
-----
Total: 6/6
```

Fig. 10. Evaluation Function Grading code 결과

결과 및 분석

```

PS C:\Users\leego\OneDrive\바탕 화면\multiagent\multiagent> python autograder.py -q q5 --no-graphics
Starting on 5-3 at 21:15:18

Question q5
=====

Pacman emerges victorious! Score: 1074
Pacman emerges victorious! Score: 1291
Pacman emerges victorious! Score: 999
Pacman emerges victorious! Score: 1010
Pacman emerges victorious! Score: 1139
Pacman emerges victorious! Score: 1059
Pacman emerges victorious! Score: 1315
Pacman emerges victorious! Score: 1333
Pacman emerges victorious! Score: 1305
Pacman emerges victorious! Score: 1315
Average Score: 1184.0
Scores:      1074.0, 1291.0, 999.0, 1010.0, 1139.0, 1059.0, 1315.0, 1333.0, 1305.0, 1315.0
Win Rate:    10/10 (1.00)
Record:      Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q5\grade-agent.test (6 of 6 points)
***      1184.0 average score (2 of 2 points)
***      Grading scheme:
***          < 500: 0 points
***          >= 500: 1 points
***          >= 1000: 2 points
***      10 games not timed out (1 of 1 points)
***      Grading scheme:
***          < 0: fail
***          >= 0: 0 points
***          >= 10: 1 points
***      10 wins (3 of 3 points)
***      Grading scheme:
***          < 1: fail
***          >= 1: 1 points
***          >= 5: 2 points
***          >= 10: 3 points

```

Fig. 11. Evaluation Function 결과

결과를 실행하면서 볼 수 있었던 점은 가끔씩 중앙에서 움직이는 팩맨과 멈춰서 움직이지 않는 팩맨을 볼 수 있었는데, 그 원인으로는 아마도 점수에 대해 혼동이 생겨서 이러한 문제가 생긴 것 같다.

느낀 점

이번 과제를 하면서 어려웠던 점은 바로 음식의 거리에 따른 각각의 유틸리티 값의 부여와 유령위치, 파워 위치에 따른 유틸리티 또한 계산 해야 한다는 점이 있었다. 역수를 취해서 줌으로써 계산하는데 있어서 많이 헷갈렸던 점이 있었던 것 같다. 너무 어려웠던 문제였다.

3 Conclusion

느낀 점

이번 과제를 모두 하면서 느꼈던 점은 인공지능을 배우는데 있어서 많은 알고리즘이 존재하고, 해당 알고리즘을 이해해야 문제를 해결 할 수 있다는 것을 알 수 있었다. 실제로 위에서 얘기한 것처럼 컴퓨터 프로그램에서도 사용하는 알고리즘만큼 인공지능을 배우는데 있어 중요한 내용이었던 것 같고, 이 문제들을 해결 할 수 있어서 성취감을 나뉘 느낄 수 있었다. 마지막으로 이번과제도 많이 어려웠던 점이 존재하여 공부를 열심히 해야겠다고 다짐하였다.

어려웠던 점

가장 어려웠던 점으로는 각각의 유틸리티를 정하는 것이 가장 어려웠던 점인 것 같다. 유틸리티를 정하는데 있어서 다양한 알고리즘을 사용하였지만, 5번의 직접 유틸리티를 짜는 것은 공정성 있고, 합리적으로 코드를 짜야하기 때문에 매우 어려웠던 점이 있었다. 이번 과제를 통해서 코드의 공정성이 얼마나 중요한지 알 수 있었기도 한다.

제안점

과제와 관련된 다양한 설명을 해주는 영상이나 자료들을 모아서 공지해주었으면 좋겠다.

References

1. CS188, <https://inst.eecs.berkeley.edu/~cs188/fa18/project2.html>
2. AIStudy, <http://www.aistudy.com/heuristic/mini-max.htm>
3. Tistory blog, <https://redcarrot.tistory.com/41>
4. Youtube, <https://www.youtube.com/watch?v=fY-9Kcf9ycI>