기초인공지능 HW 02 보고서

20181264 노영현

1. 각 알고리즘마다 구현한 방법에 대한 설명

아래 알고리즘들은 기본적으로 전부 Minimax 알고리즘에 기반한다. Minimax 알고리즘은 상대방이 항상 최선의 수를 둔다는 가정하에, 가능한 모든 형태를 확인해 현재 둘 수 있는 최선의 수를 선택하는 알고리즘이다. 팩맨을 max라고 설정하고 ghost들을 min이라고 설정한다. 우리의 목표는 팩맨의 움직임을 return하는 것이다. 현재 max노드에서 시작해서 가능한 움직임들을 기반으로 tree를 만든다. 각 노드에는 이전 노드에서 어느 방향으로 이동해서 도달한 것인지, 그리고 해당 노드의 평가 점수가 저장되어있다. max노드는 자식들 중 평가 점수가 가장 큰 노드를 고르고, min노드는 자식들 중 평가 점수가 가장 낮은 노드를 고른다. 트리에서 가장 밑에 있는 노드는 evaluationFunction을 통해서 현재 상황의 점수를 설정해준다. 위 방법으로 가장 위에서 골라지는 노드의 방향이 현재 pacman이 이동할 방향이 된다.

(1) MinimaxAgent

MinimaxAgent 알고리즘을 구현하기 위해서 클래스 내에 3개의 함수를 선언해주었다. maxagent함수, minagent함수, Action 함수를 선언하였다.

maxagent함수는 인자로 gameState와 depth를 받는다. 만약 gameState.isWin()이나 gameState.isLose()가 True값을 가지면 게임이 minimax알고리즘에서 밑으로 더 내려갈 수 없는 것이므로 현재 gameState의 evaluationFunction을 return한다. 또한 인자로 받은 depth가 설정해놓은 self.depth와 같으면 트리에서 원하는 만큼 밑으로 내려간 것이므로 gameState의 evaluationfunction을 return한다. 현재 노드의 평가점수를 -inf로 설정해준 뒤, 현재 노드에서 이동가능한 방향의 노드에 대해서 minagent함수를 돌려준다. 이동 가능한 노드중 minagent함수 반환값이 가장 큰 노드의 이동방향과 해당 반환값을 maxagent함수에서 return한다.

minagent함수는 인자로 gameState와 depth와 더불어 agentidx또한 받아준다. agentidx는 유령이 여러 개인 경우가 있기 때문에 각 유령에 이름을 붙여주는 것이다. agentidx 0은 무조건 pacman을 가르키고, agentidx 1부터 각각의 유령들을 가리킨다. minagent함수 또한 gameState.isWin() 또는 gameState.isLose()가 True값을 가지면 evaluationFunction을 return하고 인자로 받은 depth와 self.deepth가 같으면 evaluationFunction을 return한다. 이후 평가 점수를 inf로 초기화 해준 뒤, agentidx값이 (전체 agentnum-1)인 경우와 아닌 경우로 나눠준다. 전자의 경우, 현재 유령이 마지막 유령이므로 다음 노드가 pacman이므로 다음 노드에 대해서 maxagent함수를 실행시켜야 한다. 후자의 경우 다음 노드또한 ghost이므로 다음 노드에 대해서 minagent함수를 실행시킨다. minagent함수는 이동 가능한 노드의 평가 점수 중 가장 작은 노드를 골라 해당 노드의 이동 방향과 해당 평가 점수를 return한

다.

마지막으로 Action함수에서 maxagent(gameState, 0)을 실행하면 pacman에 대해서 minimax 알고리즘이 돌아가 현재 pacman이 어느 방향으로 이동해야 하는지를 return한다.

(2) AlphaBetaAgent

AlphaBetaAgent의 경우 Minimax알고리즘의 확장판이다. minimax 알고리즘의 경우 가능한 모든 수를 전부 확인하므로 시간적인 측면이나 공간적인 측면에서 좋지 않다. 이 문제를 해결하고자 AlphaBetaAgent 알고리즘을 사용한다.

AlphaBetaAgent알고리즘은 최종 결정에 영향을 미치지 않는 가지들을 잘라내는 알고리즘이다. 예를 들어, max노드 기준에서 max노드는 child들의 평가 점수 중 가장 큰 값을 고르고, max노드의 child인 min노드는 child들의 평가 점수 중 가장 작은 값을 고른다. max노드의 첫 번째 child에서 3을 반환했다고 가정해보자. 만약 두 번째 child의 첫 번째 child가 2라면 뒤에서 어떤 값이 나오든 이 값들은 max노드의 영향을 미치지 않는다. 이런 branch들을 확인하지 않으면 시간과 공간적인 측면에서 이득을 볼 수 있다.

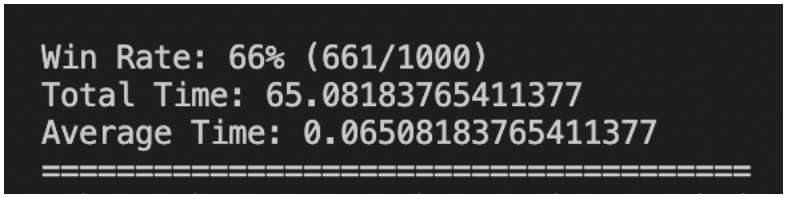
함수의 인자에 alpha, beta를 추가한다. maxagent함수에서 현재 노드에 저장되어 있는 평가점수가 alpha보다 크면 alpha를 현재 점수로 갱신하고, 평가점수가 beta보다 크면 함수를 종료하고 현재의 평가점수를 반환한다. minagent함수에서는 현재 노드에 저장되어 있는 평가점수가 beta보다 작으면 beta를 현재 점수로 갱신하고, 평가점수가 alpha보다 작으면 함수를 종료하고 현재의 평가점수를 반환한다. 정리하자면, alpha는 max에서 산출되는 hightest-value이며, beta는 min에서 산출되는 lowest-value이다.

(3) ExpectimaxAgent

ExpectimaxAgent의 경우 Minimax알고리즘의 가정인 모든 플레이어가 최선의 판단을 한다는 가정이 지켜지지 않는 경우를 대비한 알고리즘이다. minagent함수에서 child의 평가 점수들 중 가장 낮은 값을 고르는게 아닌, child의 평가 점수들의 평균을 반환하는 방식으로 실행된다. child들의 평가 점수들의 합을 child들의 개수로 나눠준 값을 반환한다.

2. 실행 캡처 화면

(1) Minimax Agent 명령어의 승률 출력 화면



(2)time_check.py에서 출력된 실행시간

```
Win Rate: 15% (45/300)
Total Time: 311.9587712287903
Average Time: 1.0398625707626343
=======================================
----- END  MiniMax (depth=3) For Medium Map

Win Rate: 20% (62/300)
Total Time: 148.85965704917908
Average Time: 0.49619885683059695
=======================================
----- END  AlphaBeta (depth=3) For Medium Map

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

Win Rate: 37% (372/1000)
Total Time: 40.004462003707886
Average Time: 0.040004462003707886
=======================================
----- END  MiniMax (depth=4) For Minimax Map

Win Rate: 35% (359/1000)
Total Time: 19.021267652511597
Average Time: 0.019021267652511598
=======================================
----- END  AlphaBeta (depth=4) For Minimax Map
```

(3) Expectimax Agent 명령어의 승률 및 Score 출력 화면

```
----------Game Results----------
Average Score: 15.0
Score Results: -502, 532, -502, 532, -502, -502, 532, 532, 532, -502, 532, 532, -502, -502, 532, -502, 532, 532, -502, 532, -502, 532, -502, 532, -502, -502, 532, -502, 532, -502, 532, -502, 532, -502, 532, 532, -502, -502, -502, 532, 532, 532, 532, 532, -502, 532, 532, 532, -502, 532, 532, -502, 532, -502, -502, -502, -502, -502, 532, 532, 532, 532, 532, 532, 532, 532, 532, -502, -502, 532, -502, -502, -502, -502, -502, -502, 532, 532, 532, 532, -502, -502, -502, 532, 532, -502, -502, -502, -502, -502, -502, -502, 532, 532, 532, 532, -502, -502, -502, 532, 532, -502, -502, -502, -502, -502, -502, 532, -502, -502, -502, 532, 532, 532

Win Rate: 50% (50/100)
Total Time: 0.31125926971435547
Average Time: 0.0031125926971435547
=======================================
```