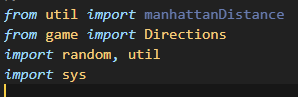
기초인공지능 2차 과제

20171637 박성우

1. 사용한 라이브러리



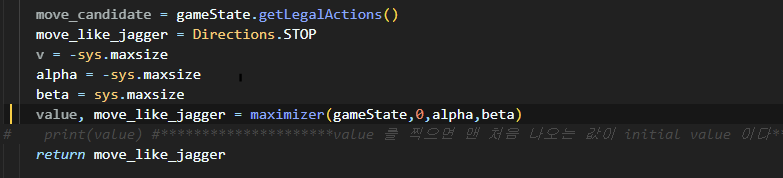
사용한 라이브러리이다. -무한대와 무한대를 나타내기 위해 sys 를 불러 sys.maxsize 를 사용하였다.

2-1 Minimax 알고리즘

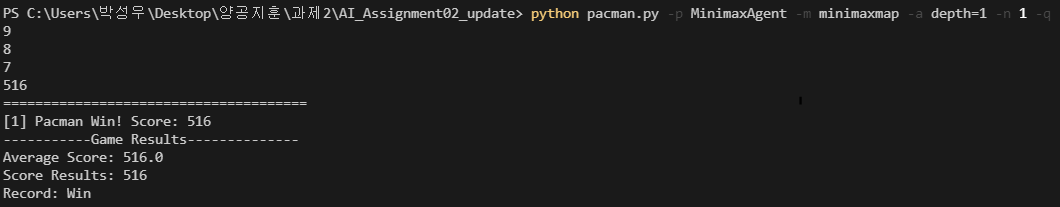
처음에 구현한 알고리즘은 minimax 알고리즘이다. 플레이어는 최대 depth 만큼의 미래를 내다보고 최선의 수를 결정하게 된다. 플레이어가 움직이는 방법을 정의한 것이 maximize 함수이고 유령들이 움직이는 것을 정의한 것이 minimize 함수이다. 플레이어가 먼저 움직이므로 처음에 maximizer 함수를 호출하였다. Maximizer 함수 안에서 플레이어가 가능한 움직이는 경우를 받아오고 각각의 경우에 대하여 유령들을 움직인다. 이때 유령들은 유령들 입장에서 가장 좋은, optimal 한 방법으로 움직인다고 가정한다. 곧 유령들은 플레이어의 점수를 가장 낮게 만드는 방향으로 움직일 것이다. 그럼 플레이어는 각각의 상태에서 유령들을 움직이는 minimizer 함수를 부르고 minimizer 함수가 반환하는 값들 중에 가장 큰 것을 고르고 그 방향으로 action을 선택하면 된다. 고로 maximizer 함수에서 minimizer를 부르고 minimizer 함수에서 maximizer 함수를 부르면서 재귀적으로 들어가게 되는데 설명서에서도 나와있듯 모든 유령이 움직여야 depth가 1 증가한다. 따라서 minimizer 함수에서는 전체 agent 수를 받아와 마지막 유령까지 다 움직일때만 이제 플레이어를 움직이라고 maximizer 함수를 부르고 그 외에는 minimizer 를 다시 불러 다음 유령을 움직여준다. Depth 가 정해진 depth 가 되거나 다음 수에서 플레이어가 승리하거나 패배하면 멈추고 그때의 점수를 반환한다.

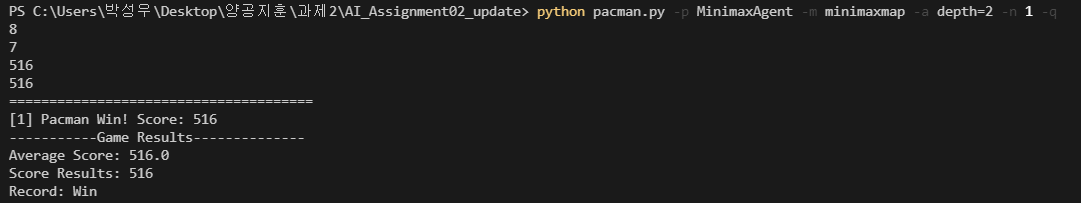
python pacman.py -p MinimaxAgent -m minimaxmap -a depth=4 -n 1000 -q 명령어를 입력하면 50~70% 사이의 승률이 적절하게 잘 나온다.

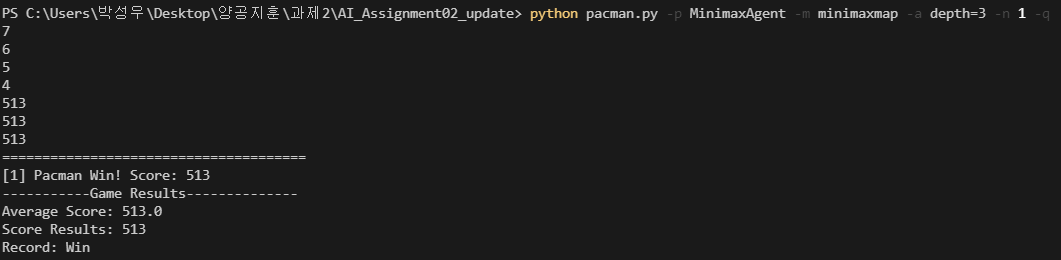
Initial value를 찍어보려면, 주석 처리되어있는 print(value)를 풀어보면 된다.

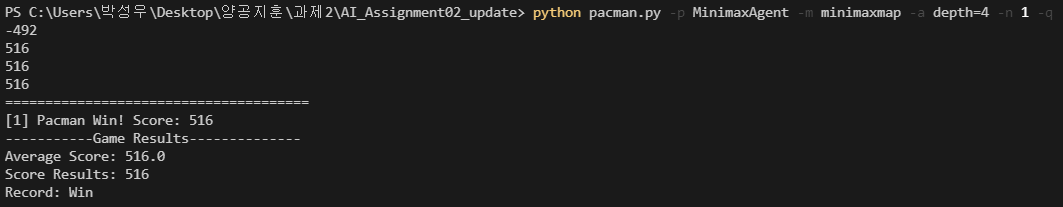


혹시 몰라 initial value 도 캡쳐하였다.









Alphabeta agent 에서도 동일하게 나온다. Print 문을 주석을 풀고 확인가능 하다.



2-2 Alpha-Beta pruning 알고리즘

Alpha-Beta pruning 알고리즘은 기본적으로 minimax 알고리즘과 동일하다. 하지만 다음 수를 결정하는데 있어서 가능한 모든 수를 확인하는 것은 시간이 너무 오래 걸리기에 구지 확인할 필요가 없는 경우에는 멈추고 다음으로 넘어가게 하는 알고리즘이다. Maximizer 함수에서는 minimizer 함수를 불렀을 때 value가 원래의 것보다 더 크면 바꾸어 주고 alpha 값을 value와 비교해 value가 더 크면 alpha를 갱신한다. 그리고 value 가 만약 beta 보다 더 크면 더 이상 할 필요없이 바로 그 value 를 리턴하면서 멈춘다. 반대로 minimizer 함수에서는 value가 원래의 value 보다 작으면 value를 바꾸어 주고 beta 값을 value 와 비교해서 더 작으면 beta를 갱신한다. 그리고 value가 alpha 보다 작으면 value를 리턴하면서 멈춘다. 이렇게 중간에 정지하고 리턴하는 부분만 추가하였다.

python time\_check.py 명령어를 입력하면 alpha-beta pruning depth=3 알고리즘과 minimax depth=2 알고리즘을 비교하게 된다. 여기서 한가지 의문점은 depth 가 동일한 상태에서 비교해야 정확하게 비교가 되지 않나 싶다. 그러나 alpha-beta pruning이 depth=3 일 때도 minimax의 depth=2 보다 빠른 것을 보면 확실히 alpha-beta 알고리즘이 더 효율적이라는 것을 알 수 있다.

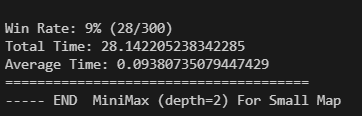
Smallmap 과 Mediummap 에 대해서 2개씩 사진을 첨부하였다.

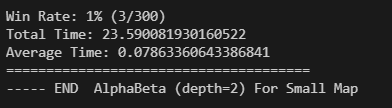
**갑자기 time\_check.py 가 변경되었다. 추가로 사진을 첨부한다. 새로운 Time\_check.py 의 ghost 는 packman 을 쫒아 다니는 것 같아 승률이**

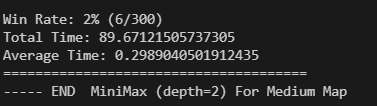
**python pacman.py -p MinimaxAgent -m minimaxmap -a depth=4 -n 1000 -q**

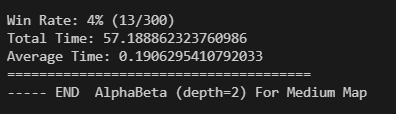
**와 같이 직접 명령을 쳐서 하는 것 보다 승률이 많이 줄어든다.**

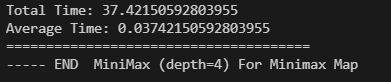
**사진 추가**

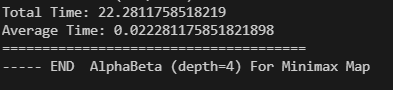




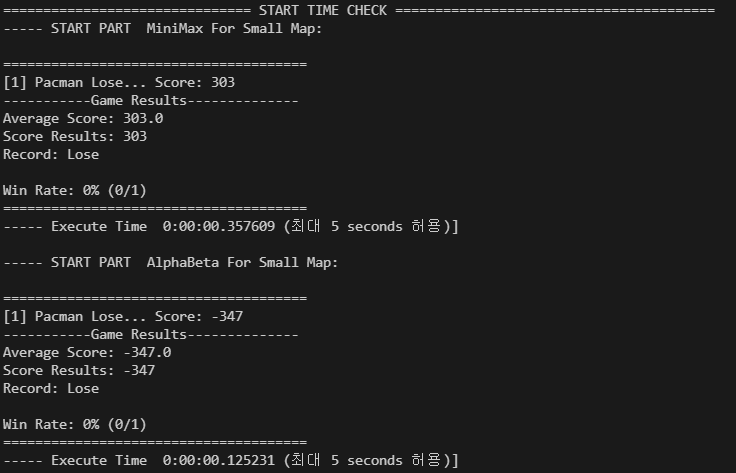


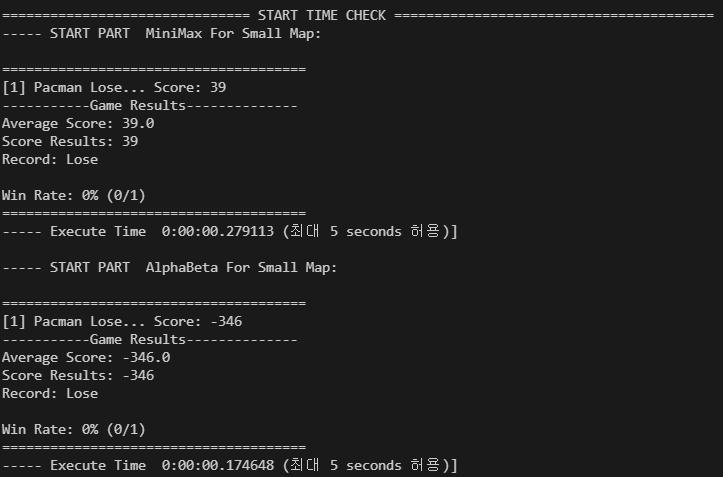




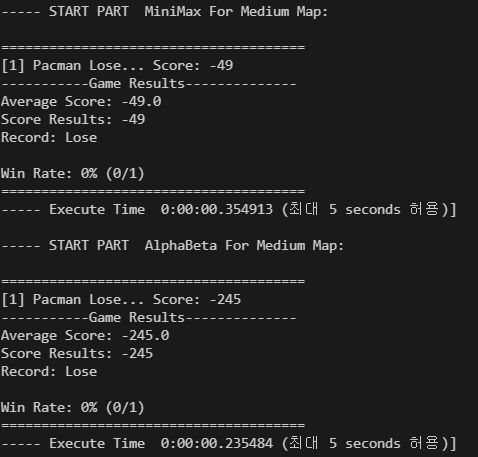


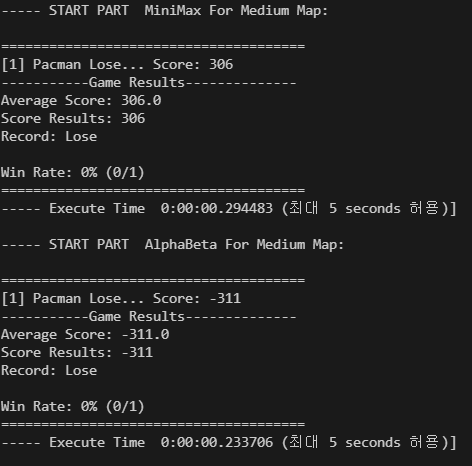
Smallmap





Mediummap





2-3 Expectimax 알고리즘

Expectimax 알고리즘은 유령이 반드시 optimal 하게 움직이지 않는 경우도 고려하는 알고리즘이다. 그래서 player 와 ghost 층 중간에 랜덤 노드를 넣어서 player 차례에서 미래를 내다볼 때 ghost 들이 미리 정해진 확률에 의해서 좀 더 랜덤성있게 움직이는 미래를 보게된다. 반대도 마찬가지이다. 그러나 이번 과제에서는 고스트들이 움직일 수 있는 모든 경우에 대한 가중치를 특정한 방향에 더 많이 주거나 적게 주기 어려워 전부 똑같다. 따라서 사실상 랜덤 노드를 만들지 않고 고스트들에 대해서만 리턴값을 평균으로 변경해주기만 하면 되지만 expectimax 알고리즘에 충실하게 따르기 위해 expectimizer 라는 제 3 agent 를 넣었고 player는 minimizer 대신에 expectimizer 를 부르고 유령들 중 마지막 유령에서 maximizer 대신 expectimizer 를 부른다.

python pacman.py -p ExpectimaxAgent -m stuckmap -a depth=3 -n 100 -q 명령어를 입력하면 expectimax 알고리즘의 점수와 승률이 출력된다. 많이 돌려본 결과 대략적으로 50% 전후의 승률과 이길 경우 532점, 질 경우 -502점이 고정적으로 잘 나온다.

