인공지능 (A.I)

[Four in a row (Connect Four)]

**Group:** 난 너가 어디 둘지 다 안단다

**Member:** 2012210070 최지웅 컴퓨터학과

2012210060 정치훈 컴퓨터학과

2012210076 박종한 컴퓨터통신공학부

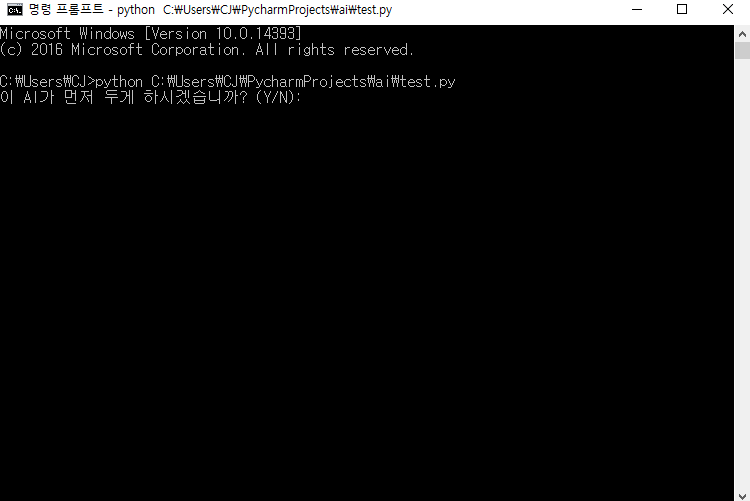
1. 개발환경

저희는 이번 Connect Four을 프로그래밍 할 때 Python을 사용했습니다. Python 은 좀더 유연한 개발환경을 보유하고 있고 보다 빠른 속도의 개발과 다양한 방면의 접근으로 Connect Four의 개발을 진행 할 수 있기에 Python을 선택 하였습니다.

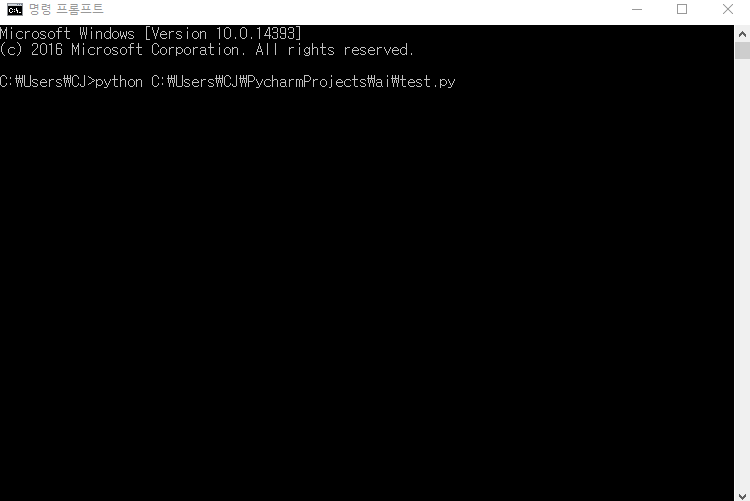
1. 실행방법

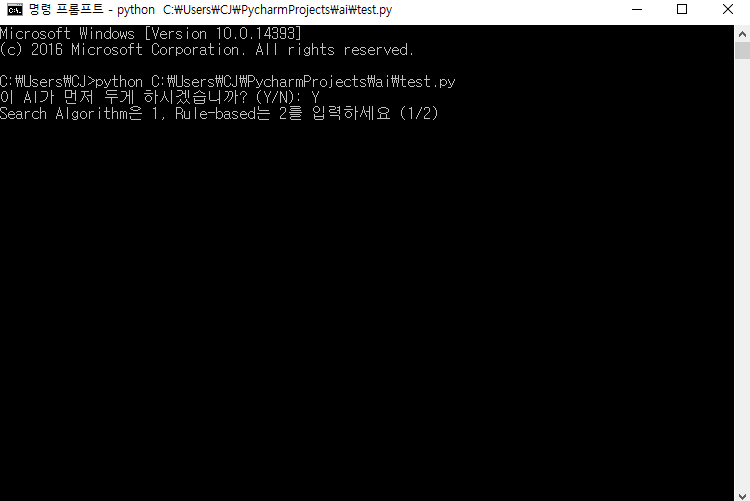


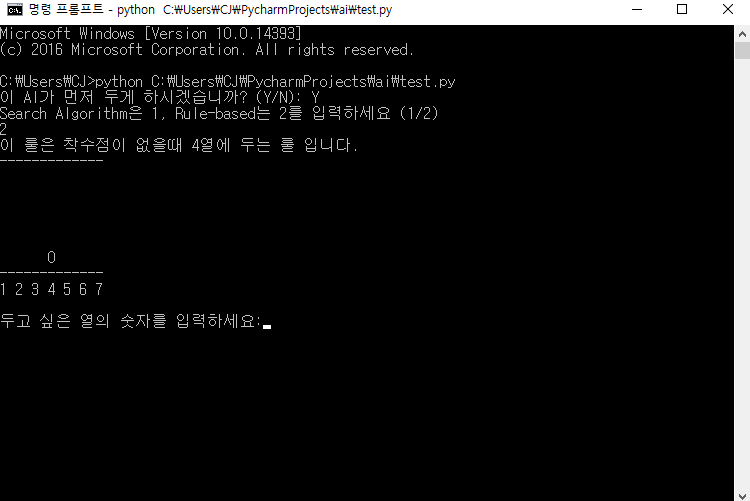
* Cmd 창을 켠다



* Python을 친 후 python 파일을 Cmd 창으로 드래그 한다







* 실행화면ing

1. 수업시간에 학습한 Search Algorithm 종류
2. Mini-max Algorithm (최소최대 알고리즘)

Mini-max는 예상되는 최대의 손실을 최소화 시키기 위해 사용하는 의사결정이론(Decision Theory)의 한 방법입니다. MAX와 MIN이 게임을 한다고 할 때, 트리에서 다음 단계의 상태로 이동할 때 가장 바람직한 이동이 될 수 있도록 합니다. 이때 평가함수 는 MAX에게 가장 유리한 양의 값으로 놓여지게 하여 MAX는 끝단 노드의 평가 값이 가장 큰 것을 선택하고, 그 선행 노드인 MIN은 MAX에 올려진 값 중에서 가장 작은 값을 선택하게 됩니다. 이러한 과정을 반복해서 수행함으로써 정해지는 시작 노드의 평가 값은 노드의 위치에 직접적으로 평가 함수를 적용한 것보다 후계 노드의 위치에 대한 신뢰에 있어 더 가치를 갖게 됩니다.

1. Genetic Algorithm (유전 알고리즘)

유전 알고리즘은 특정한 문제를 풀기 위한 알고리즘이라기 보다는 문제를 풀기 위한 접근방법에 가까우며, 문제를 유전 알고리즘에서 사용할 수 있는 형식으로 바꾸어 표현할 수 있는 모든 문제에 대해서 적용할 수 있습니다. 일반적으로 문제가 계산 불가능할 정도로 지나치게 복잡할 경우 유전 알고리즘을 통하여, 실제 최적 해를 구하지는 못하더라도 최적 해에 가까운 답을 얻기 위한 방안으로써 접근할 수 있습니다. 이 경우 해당 문제를 푸는 데 최적화되어 있는 알고리즘보다 좋은 성능을 보여주지는 못하지만, 대부분 받아들일 수 있는 수준의 해를 보여줄 수 있습니다.

1. A\* 알고리즘

A\* 알고리즘은 주어진 출발 꼭지점에서부터 목표 꼭지점까지 가는 최단 결로를 찾아내는 그래프/트리 탐색 알고리즘 중 하나이다. 이 알고리즘은 각 꼭지점 x에 대해 그 꼭지점을 통과하는 최상의 경로를 추정하는 순위 값인 “Heuristic 추정 값” h(x)을 매기는 방법을 쓴다. 이 알고리즘은 이 Heuristic 추정 값의 순서로 꼭지점을 방문한다. 그러므로 A\* 알고리즘을 너비 우선 탐색의 한 예로 분류할 수 있다.

1. Alpha-Beta Pruning

Alpha-Beta pruning 은 두 명이 참여하는 게임을 위한 최소최대(Mini-max) 알고리즘에 의해 평가되는 노드들의 수를 감소시키기 위한 기술이다. 즉 게임에서 상대방은 절대 도달하지 못하도록 나에게만 유리하게 탐색트리의 일부분을 잘라내는 것입니다. Alpha-Beta pruning을 가진 minimax 는 pruning 을 하지 않는 Mini-max와 같은 결과를 보이지만 훨씬 더 효율적입니다. **보통 효율적인 분기요소 (**branching factor) 를 square root까지 감소시키고, 같은 시간 내에 탐색할 수 있는 가지 (ply) 의 수를 두 배로 만듭니다. 두 값 알파와 베타는, maximizing player 가 보장하는 (assured of) 최소 점수이며 마찬가지로 minimizing player 가 보장하는 최대 점수를 나타냅니다. 처음에 알파는 마이너스 무한대 (minus infinity) 베타는 플러스 무한대 입니다. 그 과정이 반복됨에 따라 "window" 는 더 작게 됩니다. 베타가 알파보다 작게 되면, 그것은 현재의 위치가 두 플레이어에 의한 가장 좋은 플레이의 결과일 수 없다는 것을 의미하고, 따라서 더 이상의 탐색은 할 필요가 없게 됩니다

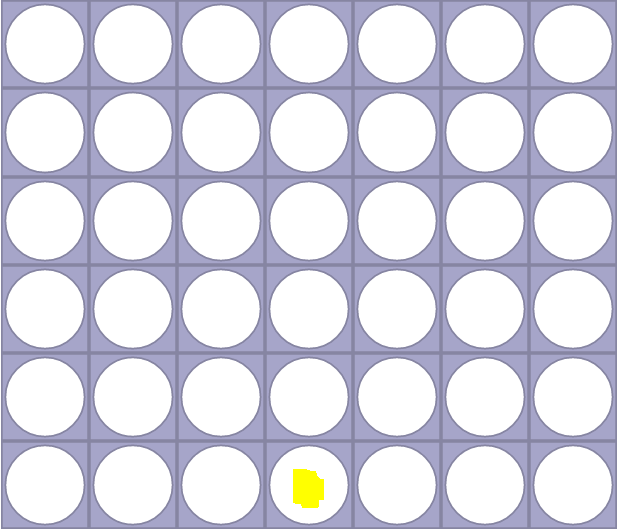
1. 알고리즘 선택 그 외 시행착오

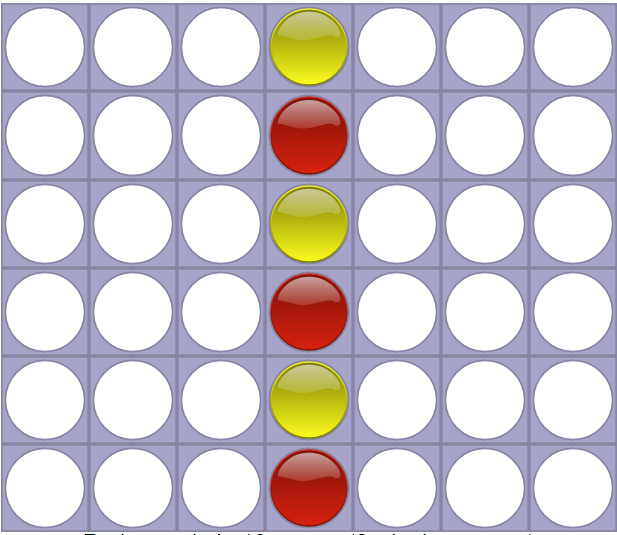
저희는 최종적으로 Mini-max 알고리즘을 선택했습니다. Mini-Max 알고리즘은 기본적으로 Two-player 게임에 사용되는 알고리즘으로 AI가 가장 최적의 해를 찾기 좋은 알고리즘이라 생각 되어 선택하게 되었습니다. 각 팀원이 한 개씩 algorithm을 맡고 리서치와 기존에 저희가 짠 코드를 기반하여 알고리즘을 대입 해본 결과 Mini-Max의 효율이 가장 좋아 Mini-Max 알고리즘을 기반으로 코드를 작성 하였습니다.

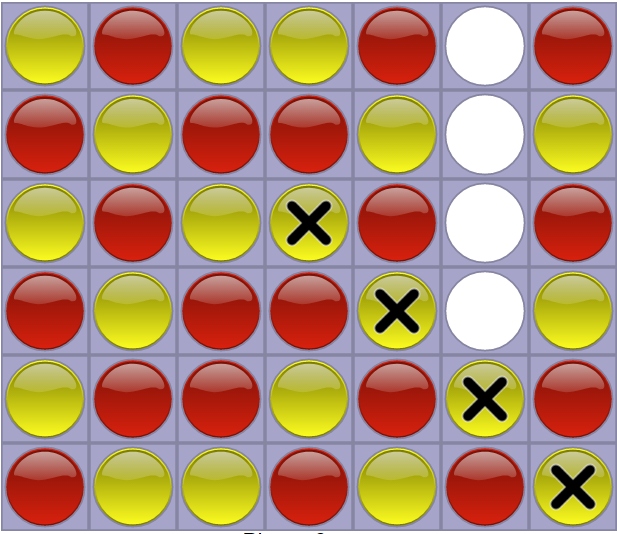
Search Algorithm에서 가장 중요한 Heuristic 함수는 4열을 착수 하는 것이라 판단되어 Mini-Max알고리즘에 4열에 착수를 완료하게끔 Heuristic을 짰습니다. 그 이유는 우선 4열에 착수를 하면 가로에서 4개를 먹히는 일이 발생하지 않게 되고 세로 혹은 대각선에서 승패가 갈리도록 하기 위해 이런 식으로 Heuristic을 설정하게 된 이유이고, Mini-Max의 depth의 값의 설정을 높일수록 가중치의 값이 선명히 구분되어 Search algorithm이 훨씬 더 효율적으로 최적의 해를 찾아 갈 수 있었고, depth의 값을 최대로 높이기 위해 각 팀원들이 각자 여러 번 시도한 끝에 5<x<9 가 가장 최상의 탐색을 할 수 있는 것을 발견하여 depth의 설정 값이 가장 높은 8로 설정하였습니다. 하지만 탐색하는 시간이 현저히 느림을 발견하였고, Alpha-beta pruning의 적용을 통하여 탐색 시간을 개선 하였습니다.

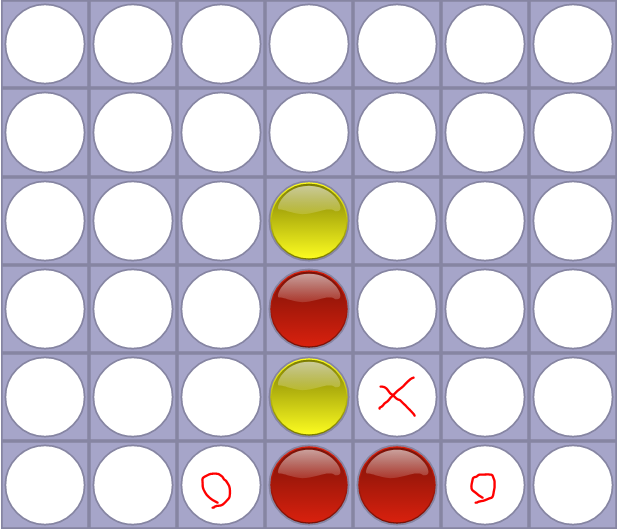
Rule에 대해서 별다른 시행착오를 겪은 건 없었지만 각자의 아이디어가 워낙 많다 보니 모든 Rule을 전부 적용시키기엔 너무나 많은 경우의 수가 나와서 3명의 아이디어 중 중복되는 아이디어만 적용을 시키도록 결정하였습니다.

1. Rule

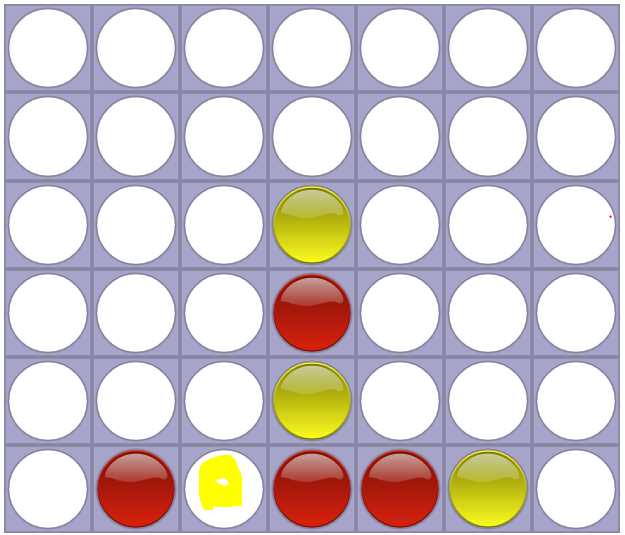
저희가 도입한 룰들은 상대가 우리의 허점을 노리는 것을 방어하는데 가장 취중을 많이 두었습니다. 우선 처음 수를 둘 때 이길 확률이 가장 높은 위치(1, 4)에 위치에 두어야 하는데 탐색 알고리즘을 사용했을 경우 (우리가 선공) (1, 4)의 위치에 수를 두지만 만약 상대가 처음에 룰을 적용하라고 했을 시를 대비하여 만약 보드에 바둑알이 놓여있지 않은 (g값 = null) 경우 (1, 4)에 착수하는 룰을 만들었습니다. 이러면 선공을 했을 때 높은 승률을 먼저 쟁취 할 수 있기에 이런 룰을 도입 했습니다.

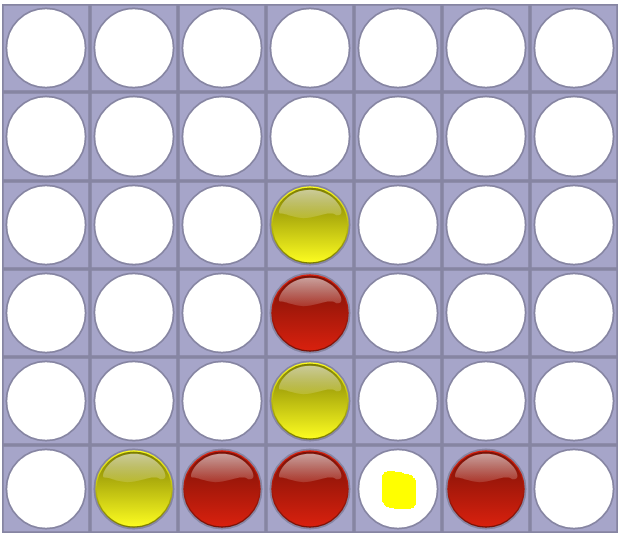
만약 상대가 선공으로 (1, 4)에 착수 했을 경우 룰을 이용하라는 요청을 받았을 시에 상대가 착수한 돌 위에 착수 하는 룰을 만들었는데 그 이유는 저희 팀원들이 각자 Perfect slover와 대전했을 때 선공이 착수한 그 돌 위에 착수하는 것이 가장 안전한 방어책임을 예측하여 그런 룰을 적용 하였고 만약 상대가 6행 마지막에 돌을 놓았을 시에 그 위에 착수 할 수 있는 칸이 없으므로 Random 하게 비어있는 칸에 돌을 놓도록 적용하였습니다. 옆의 그림과 같은 상황일 시에는 우리가 선공할 땐 이미 (1, 4)를 착수한 상태이고 1행 어디에 놓아도 4개를 만들 수 있는 여건이 되므로 Random 하게 착수하게 적용한 이유입니다.

보시는 그림과 같이 이것은 Rule 만 적용하여 Perfect solver와 대전을 한 그림인데 상대가 착수한 위치 위에 착수한 것만으로 최소 4수를 남겨두고 지는 경기를 치르는 굉장히 방어적 룰이라고 볼 수 있습니다.

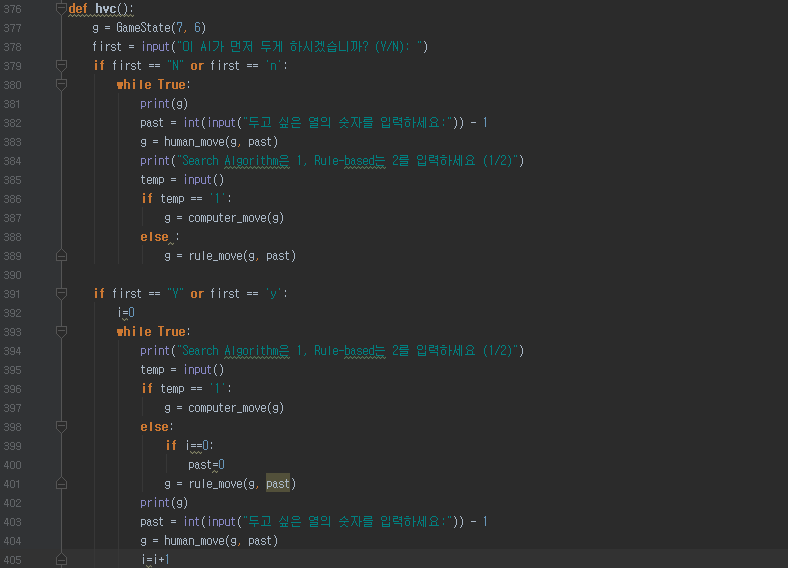


상대가 선공을 두었을 때 (1, 4)에 착수를 했다고 가정하면 일반적으로 Heuristic은 4열을 채울것이며 만약 상대가 (1, 5)에 착수를 두었을 때 저희가 기존에 짜여 있던 룰만을 기반으로 하였을 때(만약 상대가 Rule사용권장시), (2, 5)에 착수를 할 것이고 그럼 상대는 곧바로 (1, 3) 혹은 (1, 6)에 착수하여 어이없는 패배를 당할 것을 예측 하여 1열에 상대 돌이 2개 있을 때 그 옆에 돌을 착수하는 룰을 도입하여 패배를 예방하는 룰을 후에 추가하였습니다.

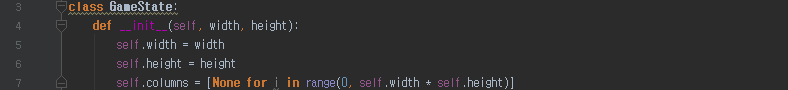


마지막 Rule은 위의 Rule의 연장선상으로 (1, 3) 혹은 (1, 6)에 우리의 돌을 착수 했다고 가정하면 또 다른 공격으로 상대가 (1, 2) 돌을 착수했다고 다시 가정해보자 그러면 상대가 다시 Rule적용을 요청했을 시에 상대의 2개돌과 1개돌 사이에 우리의 돌을 착수하는 Rule이 존재하지 않기 때문에 이것도 마찬가지로 패배로 이어질 수 있다고 생각하였습니다. 그리하여 만약 상대 2개의 돌과 1개의 돌 이 옆의 그림과 같은 포지션으로 자리 잡았을 때 우리의 돌을 그 사이에 착수하는 Rule을 추가하여 위의 Rule의 방어 대책을 좀더 강화 시켰습니다.

1. 코드 설명



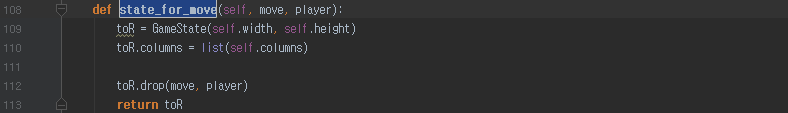
* 함수 Gamestate에 인자값 (7, 6)을 전달하여 호출한 다음 g를 초기화 합니다.
* first변수는 누가 먼저 둘 건지에 대한 문자열 y또는 n을 저장, 따라서 first값에 의해 human이 먼저 두는지 computer가 먼저 두는지 정해집니다.
* past변수는 이전 착수 열 (번호) 값을 저장, AI가 먼저 둘 경우이면서 Rule base인 경우에 이전 착수 열 값이 존재하지 않으므로 사전의 i 값을 0으로 초기화시켜 오류를 방지, past 값은 과거 착수 점을 저장하는 역할 뿐만 아니라 두고 싶은 열에 번호를 저장하는 역할 또한 함
* human move는 현재 테이블 상태인 g와 두고 싶은 열의 번호를 인자값으로 받아 열 번호의 돌을 착수하고 업데이트된 테이블을 반환해주는 함수
* computer move는 현재 테이블 상태를 인자값으로 받아 mini-max알고리즘과 alpha-beta pruning 을 통해 최적화된 수를 놓아 업데이트된 테이블을 반환해주는 함수.
* Rule move는 현재 테이블 상태 g와 과거 데이터 착수점 데이터 past를 인자값으로 받아 if then rule에 의거하여 돌을 착수한 테이블 상태를 반환하는 함수.

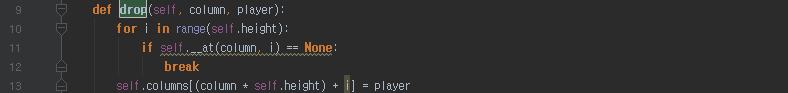
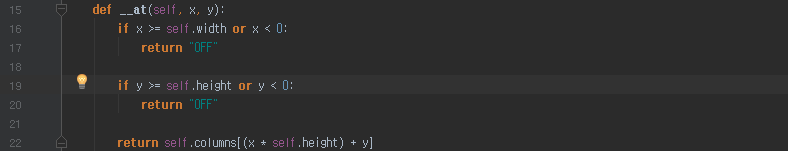


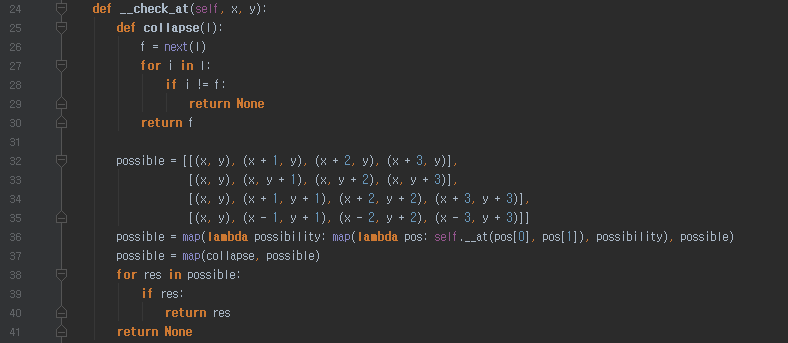
* 함수 Gamestate의 호출에 의해 (7x6)행렬 생성



* Human move 내에서 현재 테이블 상태와 돌을 착수 하고 싶은 열의 번호와 그 열에 들어갈 문자를 인자값으로 받는 state\_for\_move 함수를 호출하여 현재 테이블 상태를 업데이트된 정보로 다시 초기화 한다.
* Check\_win 함수는 업데이트된 테이블 g 를 가지고 게임이 끝나는지 안끝나는지 판별해주는 함수.



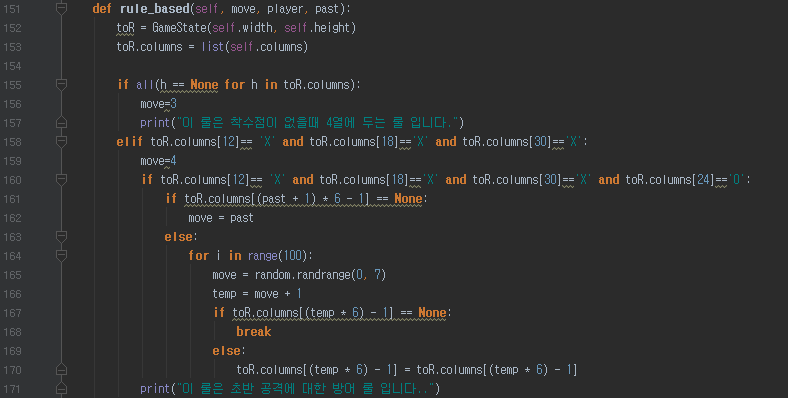
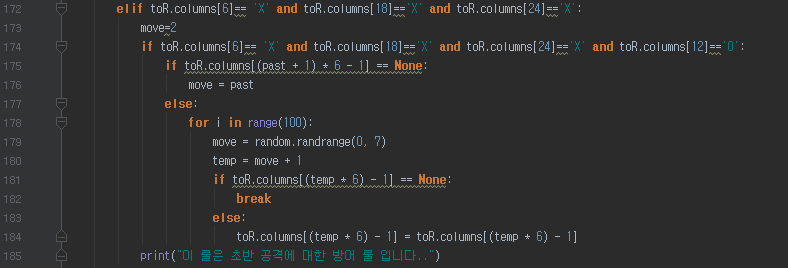
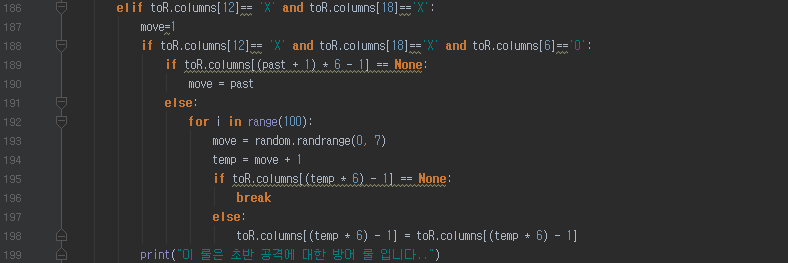
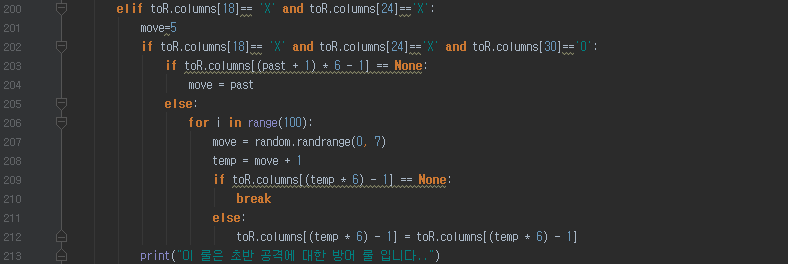
* State\_for\_move는 돌을 착수하고 싶은 열 번호값과 그 열에 착수하고 싶은 문자열을 인자값으로 받는 drop 함수를 호출.
* Drop 함수는 그 열에 인자값으로 받은 문자를 초기화 해주는 함수로써 반복문을 통하여 만약에 해당 행렬의 값이 none이면 초기화 해주고 그렇지 않으면 그 위의 열을 초기화 하려고 시도한다.

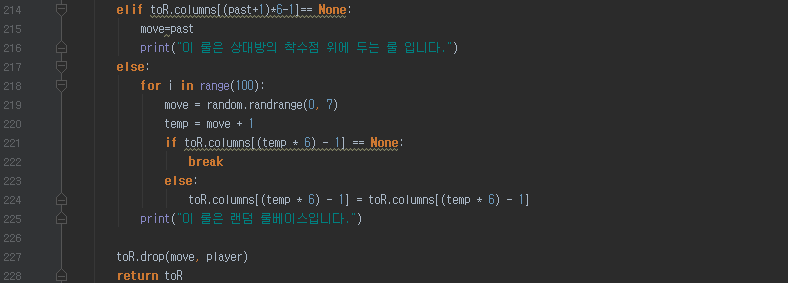


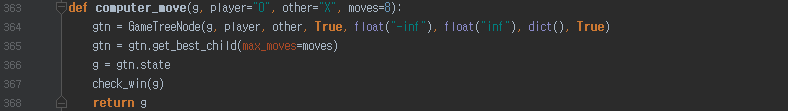
* Check 함수는 drop 함수를 통해 초기화된 테이블 g의 값을 가지고 가로, 세로, 양쪽 대각선 4개의 값이 동일한 문자로 이루어져 있는지 판단하여 만약 그렇다면 게임을 끝낸다. 하지만 이때 none값은 해당되지 않는다



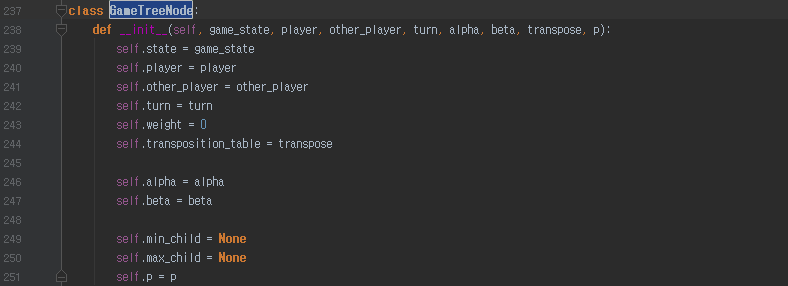
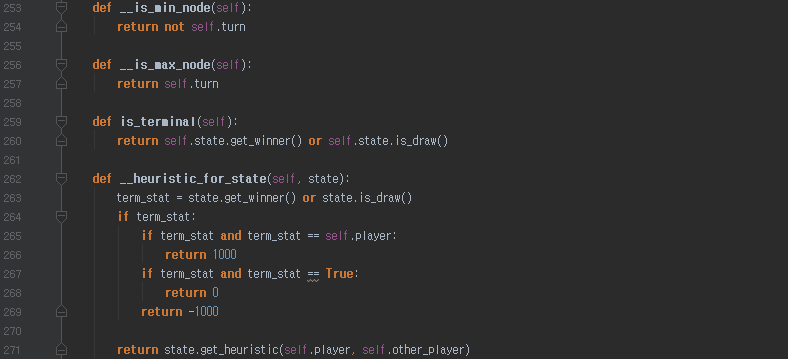
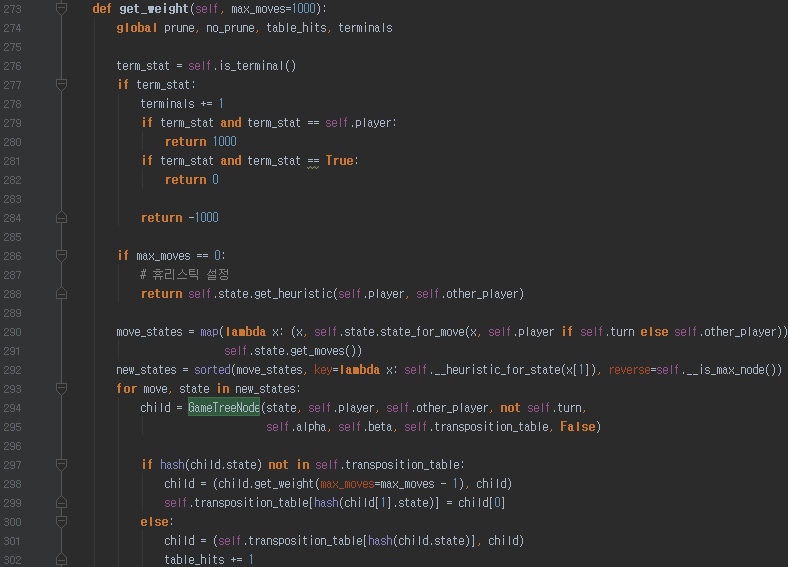
* Rule\_move는 현재 테이블 상태 g와 과거 착수점 열 번호를 인자값으로 받아 g값을 다시 업데이트 받는 함수로써 현재 테이블 상태와 move값과 문자”o” 과거 착수점 열 번호를 포함하고 있는 past 값을 인자로 받는 rule based 함수로 호출.

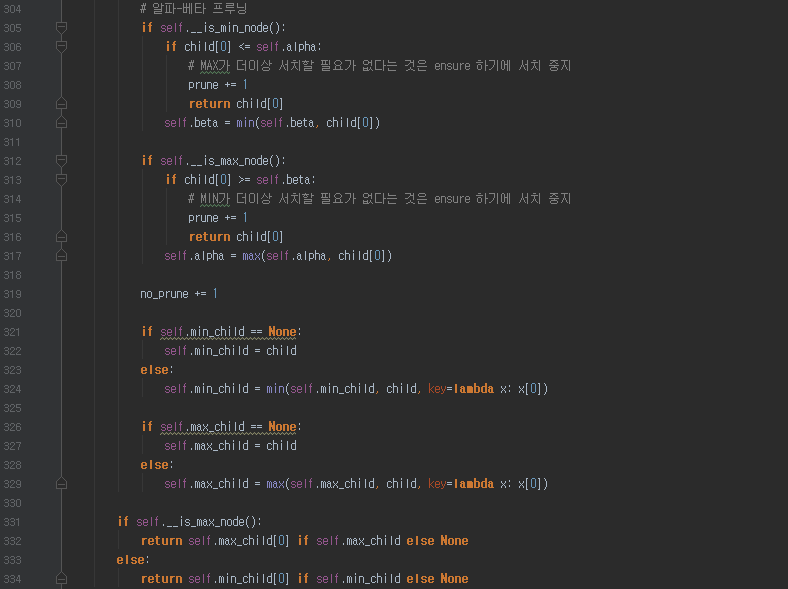
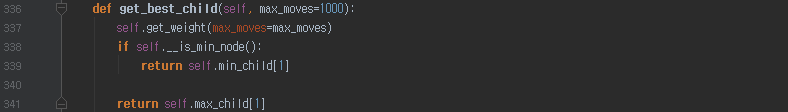


* 첫 번째 if 문은 table값이 모두 none인 경우, 즉 착수점이 아직 없고 rule based 가 첫 수 인 경우 4번열에 착수하는 rule.
* 첫 번째 와 두 번째 else if 문은 게임 극 초반 적의 연속된 두 개의 돌과 그로부터 두 칸 떨어진 적의 돌을 판단하여 그 사이에 나의 돌을 착수하는 rule
* 세 번째, 네 번째 else if 구문은 연속된 적의 두 개의 돌을 판단하여 한쪽의 돌을 착수하는 rule
* 마지막 else if 는 상대방의 마지막 착수 지점을 기점으로 그 위에 돌을 착수하는 rule
* else 구문은 random 함수를 통해 얻은 값을 반환해주는 rule로써 6번 행의 있을 때 수행하지 않고, 그것을 제외한 다른 열의 둘 수 있도록 for 문을 반복. 만약에 다른 열의 값을 얻게 되면 for문은 그 시점의 break를 통해 빠져나옴.



* computer move는 heuristic이 적용된 mini-max 알고리즘과 이를 최적화 시키는 alpha-beta pruning 을 기반으로 가장 좋은 수를 반환하는 함수.



(Mini-max트리 노드 코딩 외 alpha-beta pruning)

* “o”, ”x” 문자를 저장해주는 player, other\_player, state는 game\_state를 불러오는 변수, 가중치를 저장하는 변수 weight, 계산의 편리함을 위해 행렬을 바꿔서 저장하는 transposition\_table 변수, alpha-beta pruning을 위한 alpha, beta 변수, 틀을 구성하는 min\_child, max\_child, p 변수.
* 287줄, depth = 8을 기점으로 우리가 지정해놓은 heuristic 함수 (위의 설명 참조)
* 305줄 Alpha-beta pruning, depth가 높아짐에 따라 연산량과 complexity가 높아지므로 trade off를 위한 alpha-beta pruning 적용. (이 코드는 alpha-beta pruning 관련 논문 참조)