**인공지능 HW1**

고려대학교 컴퓨터학과

2015410113 신채호

**1. 실행 방법**

1). 콘솔에 흑이면 0, 백이면 1 입력

2). 콘솔에 원하는 AI의 시간제한 입력(플레이어는 시간제한 없음)

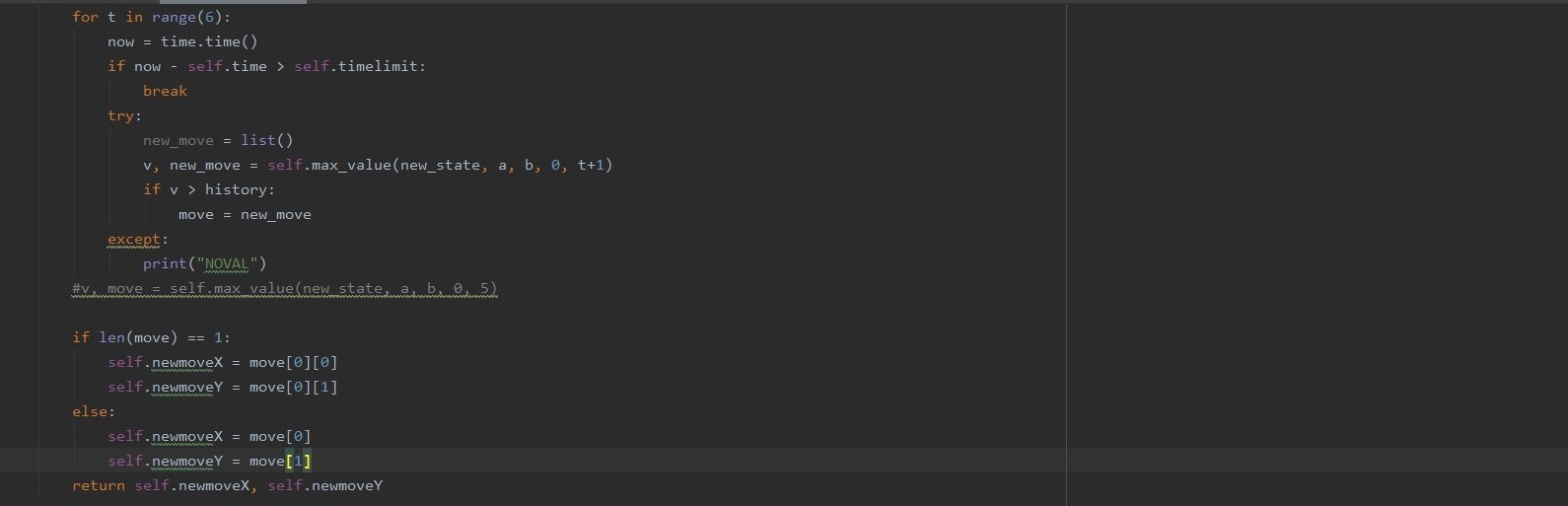
3). 바둑판이 뜨면 게임 시작 (클릭으로 돌 놓기)

**2. 코드 설명**

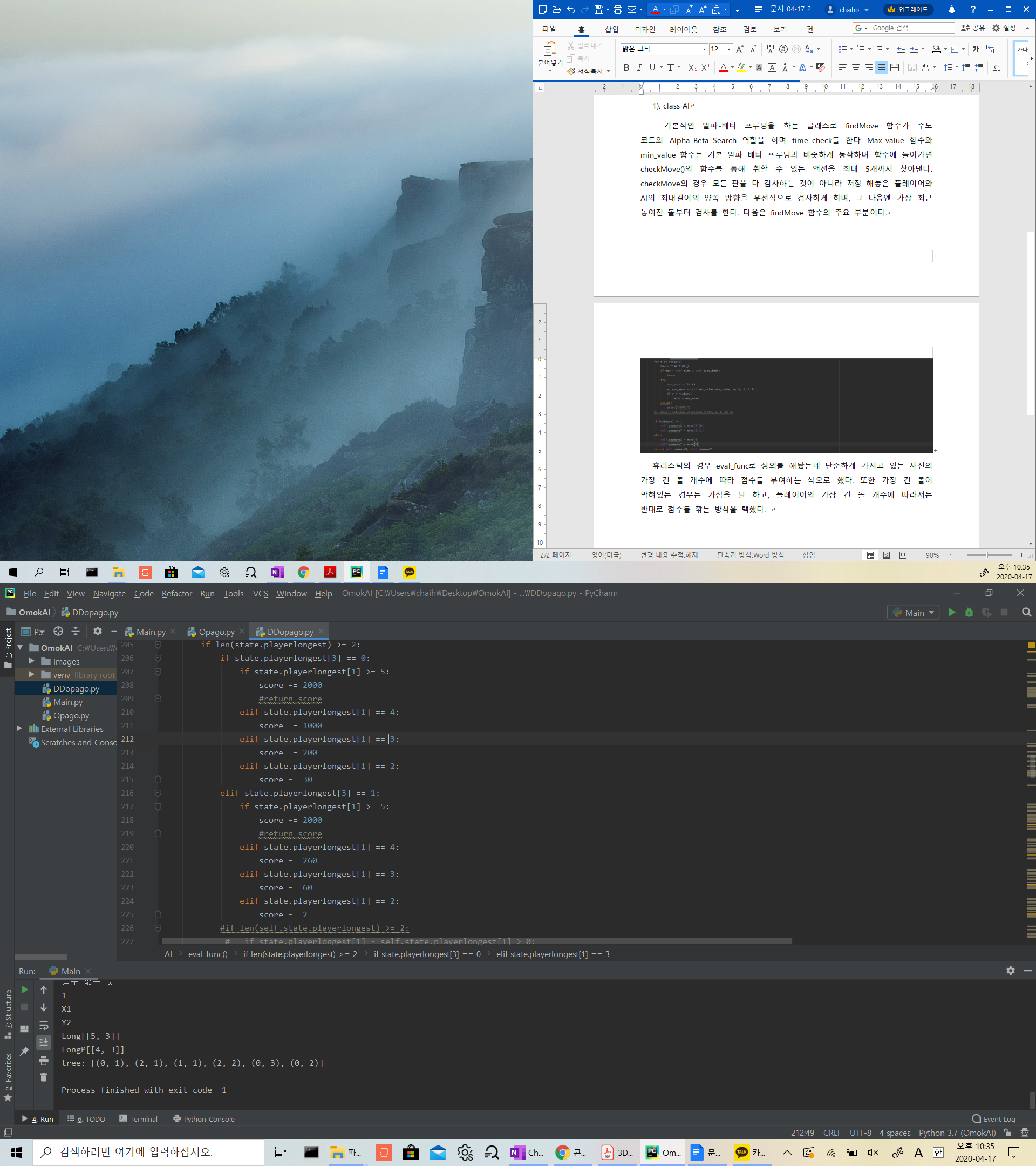
크게 Main.py와 DDopago.py로 구성되고 메인에는 AI 생성, GUI 및 3x3 처리, 오목 완성 처리 등을 한다. DDopago는 State class와 AI class로 구성된다. AI는 흑백 모두의 가장 긴 줄을 [시작점, 개수, 방향, 막힘 여부] 로 저장하고 있고, state의 변화가 있을 때 마다 새로 찾아주며 그걸 기반으로 자기 차례가 오면 최적의 state를 찾는다. AI의 시간제한을 플레이어가 설정할 수 있으며 시간제한이 끝날때까지 iterative하게 depth를 늘리며 (최대 7 depth) 서치를 한다. 서치를 하다가 중간에 시간이 지나게 되면 min\_value, max\_value 모두에 break 문을 걸어 이미 들어온거까지 검사를 하고 빠져나오게 된다.

1). class AI

기본적인 알파-베타 프루닝을 하는 클래스로 findMove 함수가 수도 코드의 Alpha-Beta Search 역할을 하며 time check를 한다. Max\_value 함수와 min\_value 함수는 기본 알파 베타 프루닝과 비슷하게 동작하며 함수에 들어가면 checkMove()의 함수를 통해 취할 수 있는 액션을 최대 5개까지 찾아낸다. checkMove의 경우 모든 판을 다 검사하는 것이 아니라 저장 해놓은 플레이어와 AI의 최대길이의 양쪽 방향을 우선적으로 검사하게 하며, 그 다음엔 가장 최근 놓여진 돌부터 검사를 한다. 다음은 findMove 함수의 주요 부분이다.

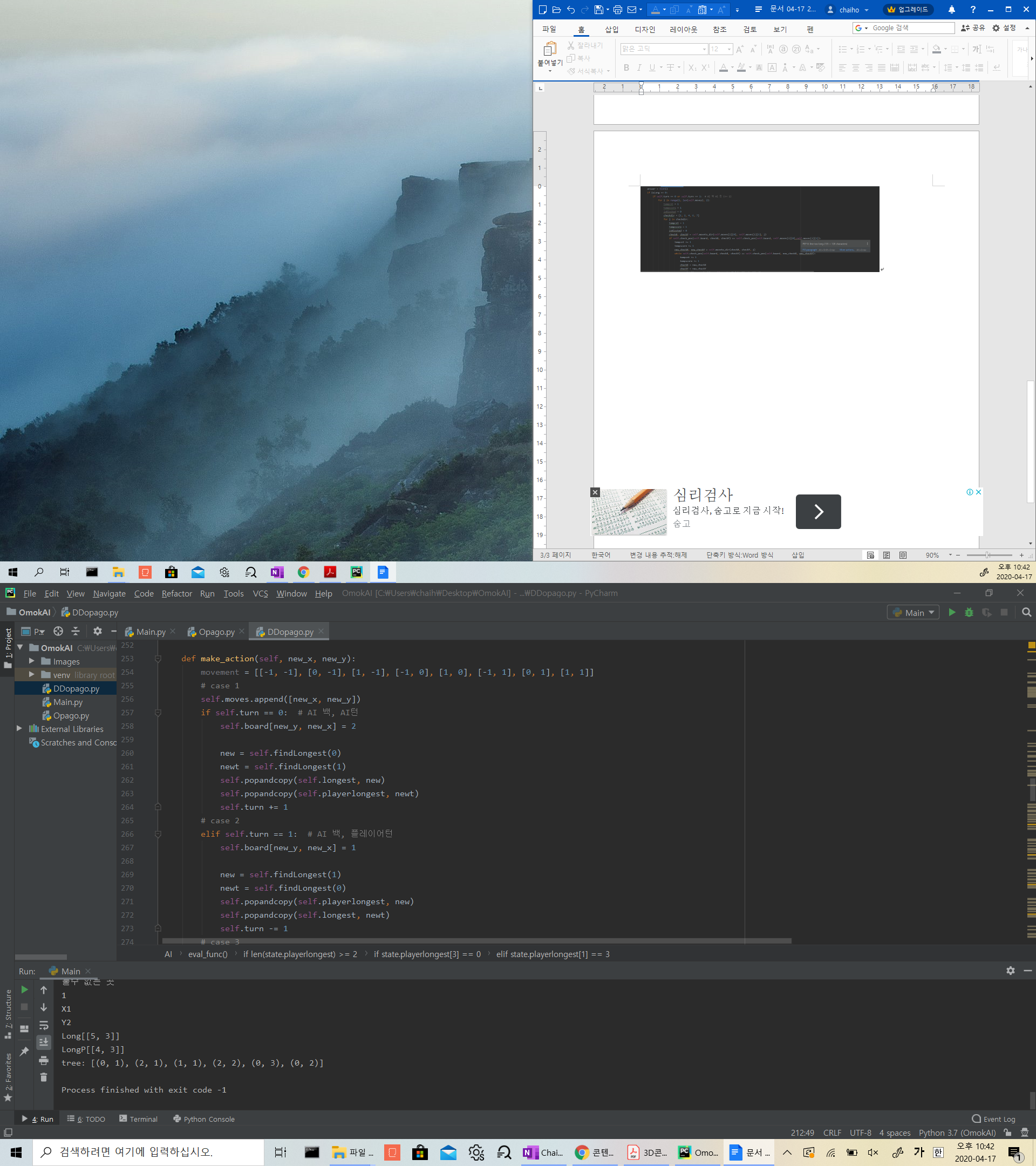


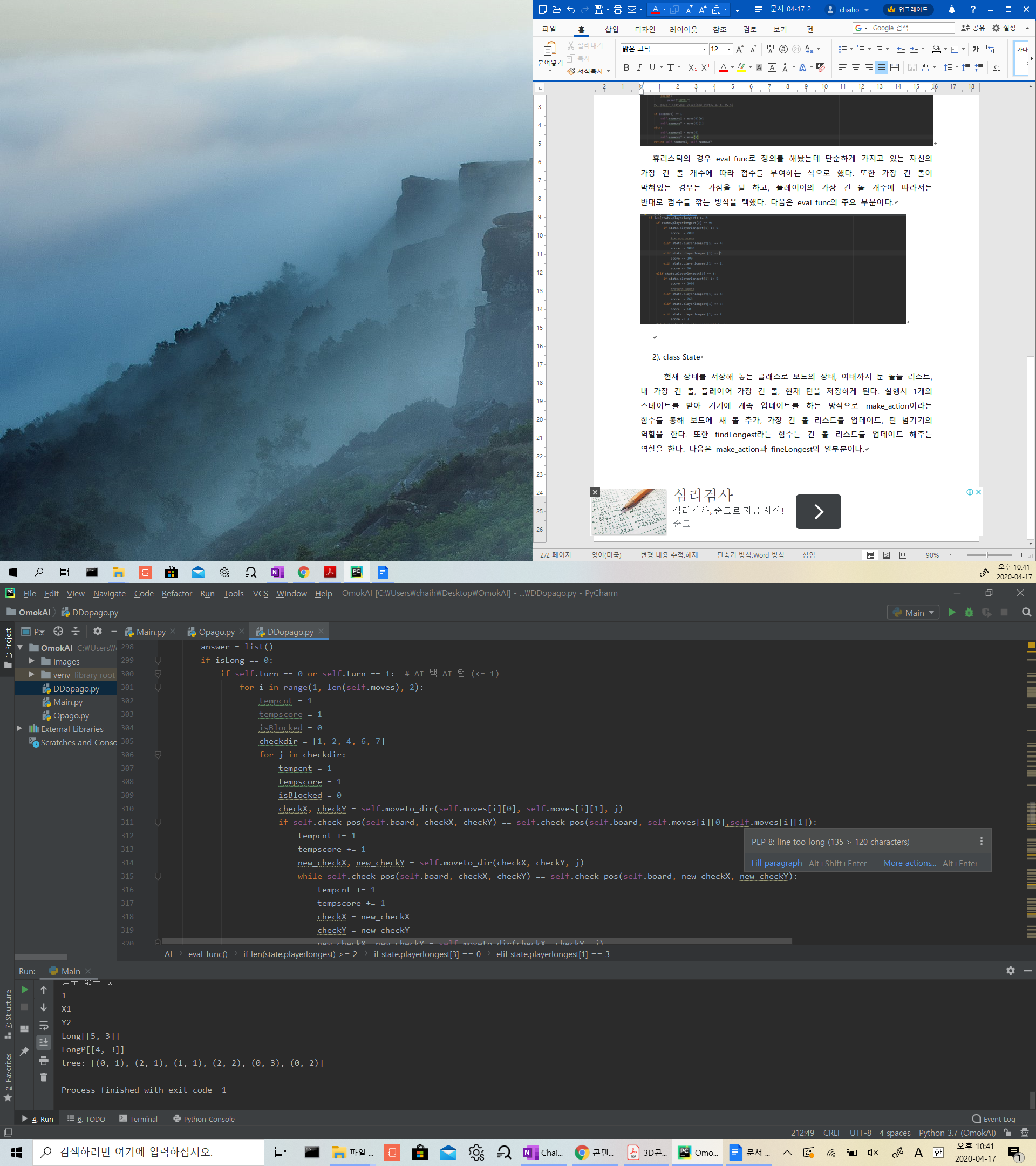
휴리스틱의 경우 eval\_func로 정의를 해놨는데 단순하게 가지고 있는 자신의 가장 긴 돌 개수에 따라 점수를 부여하는 식으로 했다. 또한 가장 긴 돌이 막혀있는 경우는 가점을 덜 하고, 플레이어의 가장 긴 돌 개수에 따라서는 반대로 점수를 깎는 방식을 택했다. 다음은 eval\_func의 주요 부분이다.



2). class State

현재 상태를 저장해 놓는 클래스로 보드의 상태, 여태까지 둔 돌들 리스트, 내 가장 긴 돌, 플레이어 가장 긴 돌, 현재 턴을 저장하게 된다. 실행시 1개의 스테이트를 받아 거기에 계속 업데이트를 하는 방식으로 make\_action이라는 함수를 통해 보드에 새 돌 추가, 가장 긴 돌 리스트들 업데이트, 턴 넘기기의 역할을 한다. 또한 findLongest라는 함수는 긴 돌 리스트를 업데이트 해주는 역할을 한다. 다음은 make\_action과 fineLongest의 일부분이다.





3). Main

GUI 셋업을 하고, 게임 설정 선택, 3x3 체크, 끝난지 여부 체크, 둘 수 있는 자리인지 여부 체크를 한다. GUI의 경우 pygame 모듈을 이용해 이미지를 load 하는 방식을 택했다.

**3. 개선점**

휴리스틱을 너무 심플한 것 같아 여러경우에서 score가 똑같이 나오게 되는 경우가 많다. 가장 긴 돌 리스트를 계속 tracking하며 그 돌들 주위를 먼저 검사하도록 하는데도 휴리스틱이 스코어링을 제대로 해주지 못하여 성능이 많이 좋지 않은 것 같다. 움직였을 때의 가장 긴 노드와 처음의 가장 긴 노드 간의 돌 개수 차이로 가점을 더 부여하는 방식이나 돌의 형태마다 가점을 다르게 하는 방식들을 생각해 봐야겠다.