인공지능 과제 보고서 : 오목 인공지능

12161570 박성연

1. 구현 환경 : Visual Code 1.34.0

   구현 언어 : Python 3.7.3

2. 사용한 모듈, 기본 변수 및 상수, 클래스

   1) 모듈

```
import pygame, sys, random
from pygame.locals import *
import numpy as np
from copy import deepcopy
from enum import Enum
```

   (1) Pygame : pygame 모듈은 파이썬 언어를 통해서 게임을 만들 수 있도록 지원해주는 모듈이다. Prompt 창에 'pip install pygame' 명령어를 사용하여 설치할 수 있다.

```
(base) C:₩>pip install pygame
Requirement already satisfied: pygame in c:₩programdata₩anaconda3₩lib₩site-packages (1.9.6)
```

   (2) Sys : 명령행 인자를 처리할 때, 인자들을 sys 모듈의 속성에 리스트로 저장되게 한다. 파이썬 표준 라이브러리에 있으므로 따로 다운로드 받을 필요는 없다.

   (3) Numpy : 행렬 즉 배열의 정의와 계산을 위해서 사용하는 모듈이다. 이 모듈은 'python -m pip install (파일 경로)' 명령어를 사용하여 설치할 수 있다.

```
(base) C:₩>pip install numpy
Requirement already satisfied: numpy in c:₩programdata₩anaconda3₩lib₩site-packages (1.16.2)
```

   (4) Copy : 객체나 변수를 복사해낼 수 있는 모듈이다. 파이썬 표준 라이브러리에 있으므로 따로 다운로드 받을 필요는 없다. Copy모듈 에서도 deepcopy를 import했다. Deepcopy를 하고 나면 원래의 객체나 변수를 수정하여도 copy한 객체나 변수는 변하지 않는다.

   (5) Enum : 서로 관련 있는 상수들을 모아서 클래스처럼 명칭의 집합으로 정의할 수 있는 모듈이다. 'conda install -c menpo enum' 명령어를 통해서 설치했다.

2) 변수

```
bg_color = (255,255,255)
black = (0,0,0)
white = (255,255,255)
red = (0,255,0)
blue = (0,0,255)
window_width = 800
window_height = 500
board_width = 500
board_size = 15
grid_size = 30
```

(1) Bg_color : 윈도우창의 전체 BACKGROUND 색깔로, 흰색인 (255,255,255) 로 저장했다.

(2) Red : 돌 위에 쓰일 글자 색깔이다. (0,255,0) 으로 저장했다

(3) Blue : 메뉴에 쓸 글씨의 색깔이다. (.

(4) Window_width : 게임을 실행시켰을 때 열리는 윈도우 창의 너비이다. 800으로 저장했다.

(5) Window_height : 마찬가지로 게임을 실행시켰을 때 열리는 윈도우 창의 높이이다. 500으로 저장했다.

(6) Board_width : 오목 보드판의 너비이다. 윈도우 창의 높이와 같은 500으로 저장했다.

(7) Board_size : 오목 보드판의 크기이다. 오목 보드판의 격자가 15x15이므로 15로 저장했다.

(8) Grid_size : 오목의 흑 돌과 백 돌의 사이즈이다. 30으로 저장했다.

3) 상수

가중치들의 상수를 저장했다.

```
#가중치 상수 정의
G22 = 5
B22 = 15
AB33 = 100
G33 = 600
B33 = 700
G44 = 150000
A33 = 5000
A44 = 200000
```

(1) G22 : 상대 돌이 두개 연결되어 있고, 양 쪽에 아무 돌이 없을 때를 열린 2라고 하는데 이때 빈 양 쪽의 칸에 더해줄 가중치이며, 5라고 저장했다. 이때 G는 GUARD의 G이다.

(2) B22 : 상대 돌이 두개 연결되어 있고, 양 쪽 중 한 곳이라도 나의 돌이 있을 때를 닫힌 2라고 한다. 다른 한 쪽에 빼어줄 가중치이며, 15라고 저장했다. 이때 B는 BLOCKED의 B이다.

(3) AB33 : 인공지능의 돌이 세 개가 연결되어 있고, 양 쪽 중 한 곳이라도 상대 돌이 있을 때를 닫힌 3이라고 한다. 이 때 닫힌 쪽을 제외한 다른 곳에 더해줄 가중치이며, 100으로 저장했다. A는 ATTACK의 A이고 B는 BLOCKED의 B이다.

(4) G33 : 상대의 열린3, 즉 양 쪽 모두 나의 돌이 없는 연결된 세개의 상대 돌이 있을 때, 양 쪽에 더해 줄 가중치이며, 600으로 저장했다.

(5) B33 : 상대의 막힌3, 즉 한 쪽에 나의 돌이 있는 연결된 세개의 상대 돌이 있을 때, 상대가 4로 만들 수 있는 경우에 빼어 줄 가중치이며, 700으로 저장했다.

(6) G44 : 상대의 막힌 4가 있을 때, 다른 한 쪽에 더해 줄 가중치이며, 150000으로 저장했다.

(7) A33 : 나의 열린3이 있을 때, 양 쪽에 더해 줄 가중치이며, 5000으로 저장했다.

(8) A44 : 나의 돌이 네 개가 연결되어 있고, 한 쪽이라도 막히지 않았다면 그 막히지 않은 곳에 더해 줄 가중치이며, 200000으로 저장했다.

```
class BoardState(Enum):
    EMPTY = 0
    BLACK = 1
    WHITE = 2
```

(9) ENUM 저장 :

오목 보드판의 배열에 각 자리마다 현재 상태를 나타내 줄 상수들을 모아둔 것이다. EMPTY는 비어있는 칸이고, BLACK은 흑 돌, WHITE는 백 돌을 의미한다.

```
turn = 1
counter = BoardState.WHITE
computer = BoardState.BLACK
```

(10) Turn은 인공지능의 수가 BLACK인지 WHITE인지를 지정하는 변수로, 먼저 흑 돌이라고 저장을 했다. 선수와 후수를 바꾸는 버튼을 누르면 이 변수를 2로 설정하여, 인공지능의 수가 WHITE가 되도록 한다.

(11) COUNTER는 상대의 상태를 저장한 변수이고, COMPUTER는 인공지능의 상태를 저장한 변수이다. 이 또한 선수와 후수를 바꾸는 버튼을 누르면 서로 바뀌도록 한

다.

4) 클래스

```
class Rule(object):…

def main():…

class AI(object) :…

class Omok(object):…

class Menu(object):…

if __name__ == '__main__':
    main()
```

(1) RULE 클래스 : 현재 상태의 BOARD를 받아서 규칙 허용이 되는 지와 연결된 5개를 찾아서 게임이 끝났는 지 확인하는 클래스이다.

(2) OMOK 클래스 : 오목의 전반적인 것을 구현한 클래스이다.

(3) MENU 클래스 : 오목 메뉴를 나타내는 것을 구현한 클래스이다.

(4) AI 클래스 : 오목을 두는 인공지능을 구현한 클래스이다.

3. 규칙 클래스

```
class Rule(object):…
```

1) 생성자

```
def __init__(self, board):
    self.board = board
```

현재 오목보드판의 상태가 저장되어 있는 리스트를 받아서 RULE 클래스의 BOARD 리스트에 저장한다.

2) IS_INVALID

```
def is_invalid(self, x, y):
    return (x < 0 or x >= board_size or y < 0 or y >= board_size)
```

클릭된 위치가 오목 보드판 위의 돌을 놓을 수 있는 곳인지 확인하는 함수이다. 함수의 인자로 클릭된 위치의 좌표 x, y를 받고, x나 y가 보드의 사이즈를 넘어가면 True를 반환하는 함수이다.

3) SET_STONE

```
def set_stone(self, x, y, stone):
    self.board[y][x] = stone
```

돌을 놓아보는 함수이다.

4) GET_XY

```
def get_xy(self, direction):
    list_dx = [-1, 1, -1, 1, 0, 0, 1, -1]
    list_dy = [0, 0, -1, 1, -1, 1, -1, 1]

    return list_dx[direction], list_dy[direction]
```

방향의 리스트를 반환하는 함수이다. 인수로 몇 번째 방향인지 받고, x리스트와 y리스트에서 해당하는 방향의 x와 y를 반환한다. 0번째 방향은 (-1,0)은 왼쪽, 1번째 방향은 (1,0)은 오른쪽을 뜻하고, 차례로 (-1,-1), (1,1), (0,-1), (0,1), (1,-1), (-1,1)은 왼쪽 위, 오른쪽 아래, 위, 아래, 오른쪽 위, 왼쪽 아래를 뜻한다. 이 함수는 주변을 탐색하는 함수들이 방향을 얻기 위해서 사용한다.

5) GET_STONE_COUNT

```
def get_stone_count(self, x, y, stone, direction):
```

연결된 돌의 개수를 세는 함수이다. 오목판이 클릭이 되어 그 위치에 돌이 놓아졌을 때 그 위치에서부터 가로(좌,우), 세로(상,하), 대각(좌상,우하), 반대각(우상, 좌하)를 확인한다.

```
x1, y1 = x, y
cnt = 1
```

먼저 클릭된 위치의 좌표를 인자로 받아서 x1,y1에 저장을 한다. Count는 현재 클릭된 곳의 돌의 개수 1로 선언을 해준다.

```
for i in range(2):
    dx, dy = self.get_xy(direction * 2 + i)
```

direction은 가로는 0, 대각은 1, 세로는 2, 반 대각은 3으로 인자를 받는다. 가로와 세로, 대각, 반대각 각각 두개의 방향을 반환해서 검사해야하므로, i를 0과 1로 하여 get_xy(direction * 2 + i)를 해서 dx, dy에 저장한다.

```
while True:
    x, y = x + dx, y + dy
    if self.is_invalid(x, y) or self.board[y][x] != stone:
        break
    else:
        cnt += 1
```

클릭된 위치의 x와 y에 dx, dy를 더해줘서 그 위치에 인자로 받은 stone과 같은 stone이 있다면 count에 1씩 더해주고, 오목판 범위에서 벗어나거나 같은 stone이 아닌 경우 다른 방향을 다른 방향을 탐색하도록 break를 해준다.

```
return cnt
```

한 방향으로 연결된 같은 돌의 개수인 count를 반환한다.

6) IS_GAMEOVER

```
def is_gameover(self, x, y, stone):
```

모든 방향에서 연결된 같은 돌의 수를 확인하고, 5가 넘어가는 방향이 있으면 그 때의 돌의 개수를 반환하는 함수이다. 클릭된 위치의 좌표와 그 때의 돌 색깔을 인자로 받는다.

```
for i in range(4):
    cnt = self.get_stone_count(x, y, stone, i)
    if cnt >= 5:
        return cnt
return cnt
```

가로, 세로, 대각, 반대각을 모두 확인할 수 있도록 get_stone_count의 direction에 0,1,2,3을 넣어 연결된 돌의 개수를 얻는다. 그 돌의 개수를 반환한다. 나중에 육목이 된다면 금수의 자리이므로 forbidden_point 함수에서 사용한다.

7) IS_FIVE

```
def is_five(self, x, y, stone):
```

다섯개가 되어 오목이 됐는지 확인하는 함수이다. 인자로 클릭된 위치의 좌표와 돌의 색을 받는다.

```
for i in range(4):
    cnt = self.get_stone_count(x, y, stone, i)
    if cnt == 5:
        return True
return False
```

가로, 세로, 대각, 반대각을 모두 확인하기 위해서 get_stone_count의 direction인자에 0,1,2,3을 넣어서 개수를 확인해보고 그 개수가 5개라면 True를 반환한다. 네 방향을 모두 확인했지만 5개가 안된다면 False를 반환한다.

8) FIND_EMPTY_POINT

```
def find_empty_point(self, x, y, stone, direction):
```

현재 둔 위치 주변에 아무 돌이 없는 위치를 찾는 함수이다. 현재 클릭된 위치의 좌표와 돌의 색깔과 가로, 세로, 대각, 반대각 중에 어떤 방향을 탐색할지를 인자로 받는다.

```
dx, dy = self.get_xy(direction)
```

인자로 받은 DIRECTION을 사용해서 dx,dy를 저장한다.

```
while True:
    x, y = x + dx, y + dy
    if self.is_invalid(x, y) or self.board[y][x] != stone:
        break
if not self.is_invalid(x, y) and self.board[y][x] == empty:
    return x, y
else:
    return None
```

탐색할 위치의 좌표를 X와 Y라고 두고 이 자리가 오목판에서 벗어나거나 같은 현재 둔 돌이 아니라면 BREAK를 통해 반복문을 벗어난다. 만약 그 자리가 오목판에서 벗어나지 않고 비어 있는 자리라면 그 위치의 좌표를 반환하고 그렇지 않으면 아무것도 반환하지 않는다.

9) OPEN_THREE

```
def open_three(self, x, y, stone, direction):
```

함수의 인자로 현재 클릭된 위치의 좌표와 돌의 색깔 그리고 방향을 받는다.

```
for i in range(2):
    coord = self.find_empty_point(x, y, stone, direction * 2 + i)
```

빈 곳을 찾아서 좌표를 저장한다.

```
    if coord:
        dx, dy = coord
        self.set_stone(dx, dy, stone)
        if 1 == self.open_four(dx, dy, stone, direction):
            if not self.forbidden_point(dx, dy, stone):
                self.set_stone(dx, dy, empty)
                return True
        self.set_stone(dx, dy, empty)
return False
```

그 좌표에 SET_STONE함수를 이용해서 돌을 둔다고 가정을 한다. 만약 그 곳에 두었을 때 열린4가 만들어진다면 FORBBIDDEN_POINT함수를 이용해서 금수 자리인지를 판단한다. SET_STONE함수를 이용해서 둔다고 가정한 자리의 돌을 다시 빼고, 금수 자리가 아니라면 True를 반환하고 금수자리면 False를 반환한다.

10) OPEN_FOUR

```
def open_four(self, x, y, stone, direction):
```

열린 4가 생기는지 확인하는 함수이다. 현재 클릭된 위치의 좌표와 돌의 색깔과 탐색해볼 방향을 인자로 받는다.

```
cnt = 0
```

오목을 만들 수 있는 4가 몇갠지 확인하는 CNT를 0으로 선언한다.

```
for i in range(2):
    coord = self.find_empty_point(x, y, stone, direction * 2 + i)
    if coord:
        if self.five(coord[0], coord[1], stone, direction):
            cnt += 1
```

FIND_EMPTY_POINT함수를 이용해서 빈 곳을 찾아 그 위치의 좌표를 저장한다. 그 위치에 돌을 두었을 때 오목이 된다면 CNT에 1을 더해준다.

```
if cnt == 2:
    if 4 == self.get_stone_count(x, y, stone, direction):
        cnt = 1
else: cnt = 0
return cnt
```

만약 CNT가 2라면 열린 4가 두개일 수도 있고, 좌랑 우를 동시에 확인해서 2개가 된 것일 수도 있다. 만약 나란히 4개가 있는 걸 확인한 것이라면 GET_STONE_COUNT의 반환 값이 4라면 CNT를 다시 1로 바꿔준다.

그 후 CNT를 반환해준다.

11) FOUR

```
def four(self, x, y, stone, direction):
```

4가 있을 때 게임이 끝날 수도 있기 때문에 검사를 해주는 함수이다. 현재 클릭된 위치의 좌표와 돌의 색깔과 방향을 인자로 받는다.

```
for i in range(2):
    coord = self.find_empty_point(x, y, stone, direction * 2 + i)
    if coord:
        if self.five(coord[0], coord[1], stone, direction):
            return True
return False
```

그 방향에서 FIND_EMPTY_POINT함수를 사용해서 빈자리를 찾아 좌표를 저장해준다. 그 좌표에 같은 돌을 놓았을 때 오목이 되면 TRUE를 반환하고 아니면 FALSE를 반환한다.

12) FIVE

```
def five(self, x, y, stone, direction):
```

오목이 만들어졌는지 확인하는 함수로 클릭된 위치의 좌표와 돌의 색깔, 방향을 인자로 받는다.

```
if 5 == self.get_stone_count(x, y, stone, direction):
    return True
return False
```

GET_STONE_COUNT함수를 통해 방향마다 연결된 같은 돌의 개수를 얻어서 그 개수가 5개라면 TRUE를 반환하고 그렇지 않으면 FALSE를 반환한다.

13) DOUBLE_THREE

```
def double_three(self, x, y, stone):
```

열린3의 개수가 2개 이상인지 확인하는 함수로, 현재 클릭된 위치의 좌표와 돌의 색깔을 인자로 받는다.

```
cnt = 0
```

열린3의 개수를 셀 변수 CNT를 0으로 선언한다.

```
self.set_stone(x, y, stone)
```

SET_STONE함수를 사용하여 인자로 받은 위치의 좌표에 돌을 둔다고 가정한다.

```
for i in range(4):
    if self.open_three(x, y, stone, i):
        cnt += 1
self.set_stone(x, y, empty)
if cnt >= 2:
    print("double three")
    return True
return False
```

가로, 세로, 대각, 반대각으로 OPEN_THREE함수를 이용해서 열린3의 개수를 센다. 방향마다 열린 3이 있다면 CNT를 1씩 더해준다.

CNT가 두개 이상이면 열린3의 개수가 2개 이므로 33이며 TRUE를 반환한다. 그렇지않으면 FALSE를 반환한다.

이 함수는 나중에 FORBIDDEN_POINT함수에 금수로 사용될 것이다.

14) DOUBLE_FOUR

```
def double_four(self, x, y, stone):
```

44를 검사하는 함수이다. 클릭된 위치의 좌표와 돌의 색깔을 인자로 받는다.

```
cnt = 0
```

사목의 개수를 세어줄 변수 CNT를 0으로 저장한다.

```
self.set_stone(x, y, stone)
```

SET_STONE함수를 사용해서 클릭된 위치에 돌을 둔다고 가정한다.

```
for i in range(4):
    if self.open_four(x, y, stone, i) == 2:
        cnt += 2
    elif self.four(x, y, stone, i):
        cnt += 1
```

가로 세로 대각 반대각을 모두 OPEN_FOUR함수를 사용해서 열린 4가 가능한 개수를 얻는다. 두 방향을로 가능하여 2를 얻는다면 CNT에 2를 더해준다. 2를 얻지 않는다면 FOUR함수를 사용해서 사목을 확인한다면 CNT에 1을 더해준다.

```
self.set_stone(x, y, empty)
```

SET_STONE을 사용하여 가정한 위치의 돌을 다시 드러낸다.

```
if cnt >= 2:
    print("double four")
    return True
return False
```

만약 CNT가 2이상이라면 44이므로 TRUE를 반환하고 아니면 FALSE를 반환한다.

이 함수도 금 수인지 확인하는 데에 쓰인다.

15) FORBIDDEN_POINT

```
def forbidden_point(self, x, y, stone):
```

금 수자리인지 확인하는 함수이다. 클릭된 위치의 좌표와 돌의 색깔을 인자로 받는다.

```
if self.is_five(x, y, stone):
    return False
```

클릭된 자리에 돌을 두었을 때, IS_FIVE함수를 이용해서 오목이 되는지 확인하고, 이때 오목이 된다면 FALSE를 반환한다. 그 자리가 금 수자리여도, 오목을 만들면 게임이 끝나기 때문에 금 수로 취급하지 않는다.

```
elif 5 < self.is_gameover(x, y, stone):
    print("overline")
    return True
```

만약 클릭된 자리에 돌을 두었을 때, 육목 이상이 된다면 금 수가 되므로 TRUE를 반환한다.

```
elif self.double_three(x, y, stone) or self.double_four(x, y, stone):
    return True
```

또한 DOUBLE_THREE함수를 사용해서 33인지, DOUBLE_FOUR함수를 사용해서 44인지를 알고 만약 맞다면 금수라고 판단하여 TRUE를 반환한다.

```
return False
```

이 모든 조건에 걸리지 않는다면 FALSE를 반환한다.

16) GET_FORBIDDEN_POINTS

```
def get_forbidden_points(self, stone):
```

금수 자리의 좌표를 반환하는 함수이다. 돌의 색깔을 인자로 받는다.

```
coords = []
for y in range(len(self.board)):
    for x in range(len(self.board[0])):
        if self.board[y][x]:
            continue
        if self.forbidden_point(x, y, stone):
            coords.append((x, y))
return coords
```

좌표를 저장할 COORDS를 선언한다.

RULE 클래스가 정의될 때의 BOARD의 길이만큼 BOARD를 모두 확인했을 때 FORBIDDEN_POINT함수를 사용해서 금수인지 확인하고, 금수이면 COORDS 배열에 좌표를 추가한다. 모든 탐색이 끝나면 COORDS를 반환한다.

4. 메뉴 클래스

```
class Menu(object):
```

전체적인 윈도우 창과 메뉴버튼을 구현한 클래스

1) 생성자

```
def __init__(self, surface):
    self.font = pygame.font.Font('freesansbold.ttf', 20)
    self.surface = surface
    self.draw_menu()
```

PYGAME의 FONT를 사용해서 'FREESANSBOLD.TTF'를 20포인트로 FONT를 설정한다

2) DRAW_MENU

```
def draw_menu(self):
```

메뉴창을 그리는 함수이다.

```
def draw_menu(self):
    top, left = window_height - 30, window_width - 200
    self.new_rect = self.make_text(self.font, 'New Game', blue, None, top - 60, left)
    self.quit_rect = self.make_text(self.font, 'Quit Game', blue, None, top - 30, left)
    self.undo_rect = self.make_text(self.font, 'Undo', blue, None, top - 150, left)
    self.turn_rect = self.make_text(self.font, 'White', blue, None, top - 120, left)
    self.doublethree_rect = self.make_text(self.font, '33Ban', blue, None, top- 90, left)
```

글을 적는 위치의 높이를 TOP = WINDOW_HEIGHT – 30, 왼쪽을 LEFT = WINDOW_WIDTH - 200으로 설정한다.

(1) QUIT GAME : 파란색으로 TOP – 30, LEFT 위치에 적는다. QUIT GAME은 게임을 끝내고 나가는 메뉴이다.

(2) NEW GAME : 파란색으로 TOP – 60, LEFT 위치에 적는다. NEW GAME은 보드를 다 비우고 새로운 게임을 시작하는 메뉴이다.

(3) 33 BAN :파란색으로 TOP – 90, LEFT 위치에 적는다. 33 BAN은 열린3을 두개 이상 두지 못하게 하는 메뉴이다.

(4) WHITE : 파란색으로 TOP – 120, LEFT 위치에 적는다. WHITE는 선수와 후수를 바꾸는 메뉴이다.

(5) UNDO : 파란색으로 TOP – 150, LEFT 위치에 적는다. UNDO는 한 수를 무를 수 있는 메뉴이다.

3) SHOW_MSG

```
def show_msg(self, msg_id):
```

누가 이겼는지 메시지를 띄우는 함수이다. 몇 번째 메시지를 띄울지 INDEX를 인자로 받는다.

```
msg = {
    empty : '
    black_stone: 'Black win!!!',
    white_stone: 'White win!!!',
}
center_x = window_width - (window_width - board_width) // 2
self.make_text(self.font, msg[msg_id], blue, bg_color, 30, center_x, 1)
```

메시지는 BLACK이 이겼을 때와 WHITE가 이겼을 때와 게임 중일 때 EMPTY로 둔다.

메뉴 부분의 가운데 부분을 CENTER_X라고 두고 윈도우창에서 보드크기를 뺀 부분에서의 가운데라고 한다.

MSG_ID 번째의 MSG를 파란색으로 가운데에 적는다.

4) MAKE_TEXT

```
def make_text(self, font, text, color, bgcolor, top, left, position = 0):
```

텍스트를 쓰는 함수이다. 폰트와 텍스트, 폰트 색깔, 배경 색깔, 위치를 인자로 받는다.

```
surf = font.render(text, False, color, bgcolor)
```

FONT의 RENDER함수를 이용하여 인자로 받은 텍스트 객체를 생성한다. 이때 RENDER가 받는 인자는 텍스트와 ANTI-ALIASING과 폰트색깔 배경색깔 이렇게 네 가지이다. 이때 ANTI-ALIASING은 TRUE이면 부드러운 느낌을 내고, FALSE면 딱딱한 느낌을 낸다.

```
rect = surf.get_rect()
```

GET-RECT함수를 사용해서 텍스트가 있는 위치를 가져와 rect 변수에 저장한다.

```
if position:
    rect.center = (left, top)
else:
    rect.topleft = (left, top)
```

만약 인자 POSITION이 1이면 RECT의 가운데는 인자의 LEFT와 TOP이 되고, 0이면 RECT의 TOPLEFT가 인자의 LEFT와 TOP이 된다.

```
self.surface.blit(surf, rect)
```

SURFACE의 BLIT을 사용하여 텍스트 객체를 출력한다.

5) CHECK_RECT

```
def check_rect(self, pos, omok):
```

메뉴를 클릭했을 때 함수를 호출하는 함수이다. 인자로는 클릭된 위치와 오목클래스를 받는다.

```
elif self.undo_rect.collidepoint(pos):
    omok.undo()
elif self.turn_rect.collidepoint(pos):
    omok.turn_change()
elif self.doublethree_rect.collidepoint(pos):
    omok.redo()
elif self.quit_rect.collidepoint(pos):
    self.terminate()
```

각 텍스트 객체를 클릭했을 때 위치에 맞는 오목의 함수를 호출하여 메뉴에 맞는 일을 하게한다.

5. 오목 클래스

1) 생성자

```
def __init__(self, surface):
self.board = [[BoardState.EMPTY for i in range(board_size)] for j in range(board_size)]
self.weightboard = [[0 for i in range(board_size)] for j in range(board_size)]
```

현재 오목 보드판에 어떤 돌이 둬져 있는지 상태를 입력할 BOARD 배열과 그 상태에 따른 각 자리마다의 WEIGHT를 입력할 WEIGHTBOARD 배열을 선언했다.

```
self.menu = Menu(surface)
self.rule = Rule(self.board)
```

메뉴 클래스와 규칙 클래스를 가져와서 쓸 예정이기 때문에 생성자에서 선언해줬다.

2) INIT_GAME

```
def init_game(self):
```

게임을 초기화하는 함수이다.

```
self.turn  = black_stone
self.draw_board()
self.menu.show_msg(empty)
self.init_board()
self.coords = []
self.redos = []
self.id = 1
self.is_show = True
self.is_gameover = False
self.is_forbidden = False
```

첫 수는 흑 돌로 초기화한다. DRAW_BOARD함수를 사용하여 보드를 새로 그리고, 메뉴 창 상단에 누가 이겼는지에 대한 말을 지운다. BOARD 배열을 초기화한다. 돌이 둬진 좌표들을 저장하는 COORDS 배열을 초기화하고, UNDO할 때 쓸 배열 REDOS를 초기화 한다. 돌을 몇 개 뒀는지 세는 ID도 초기화 한다.

3) SET_IMAGE_FONT

```
def set_image_font(self):
```

필요한 이미지와 폰트를 불러오는 함수이다.

```
white_img = pygame.image.load('image/white.png')
self.white_img = pygame.transform.scale(white_img, (grid_size, grid_size))
black_img = pygame.image.load('image/black.png')
self.black_img = pygame.transform.scale(black_img, (grid_size, grid_size))
```

IMAGE 폴더에 있는 흑 돌, 백 돌을 가져온다. 이때 PYGAME 모듈의 IAMGE.LOAD를 사용해서 불러오고, TRANSFORM.SCALE을 이용하여 사이즈를 처음에 설정했던 바둑돌의 사이즈 GRID_SIZE로 변형하여 저장한다.

```
self.board_img = pygame.image.load('image/board.png')
self.font = pygame.font.Font("freesansbold.ttf", 14)
```

마찬가지로 오목보드판과 폰트를 불러와서 저장한다. 오목보트판을 불러올 땐 마찬가지로 PYGAME의 IMAGE.LOAD함수를 사용하고, 폰트를 불러올 때는 MENU에서와 같이 PYGAME의 FONT.FONT함수를 사용한다.

4) INIT_BOARD

```
def init_board(self):
```

현재 오목 보드판에 어떤 돌들이 어디에 둬져 있는 지 상태를 알려주는 BOARD 배열을 초기화하는 함수이다.

```
for y in range(board_size):
    for x in range(board_size):
        self.board[y][x] = 0
```

for문을 사용해서 BOARD 배열을 모두 0으로 초기화 시켜준다.

5) DRAW_BOARD

```
def draw_board(self):
    self.surface.blit(self.board_img, (0, 0))
```

보드판을 그리는 함수이다.

SET_IMAGE_FONT 함수에서 불러온 보드판의 이미지를 BLIT함수를 사용해서 윈도우 창의 (0,0)에 출력한다.

6) DRAW_IMAGE

```
def draw_image(self, img_index, x, y):
    img = [self.black_img, self.white_img, self.last_b_img, self.last_w_img]
    self.surface.blit(img[img_index], (x, y))
```

돌을 그리는 함수이다. 인자로 IMAGE_INDEX와 클릭되어 돌을 둘 좌표를 받는다. IMG 배열에는 SET_IMAGE_FONT함수에서 불러온 흑 돌과 백 돌, 현재 상태에서 마지막으로 그려진 돌의 위치에 그려줄 LAST BALCK IMAGE와 LAST WHITE IMAGE가 있다. IMAGE_INDEX 인자로 결정된 돌을 인자로 받은 좌표에 BLIT함수를 사용해서 출력한다.

7) UNDO

```
def undo(self):
```

수를 무를 수 있는 함수이다.

```
self.id -= 1
self.draw_board()
coord = self.coords.pop()
self.redos.append(coord)
x, y = self.get_point(coord)
self.board[y][x] = empty
```

여태 둬진 돌의 개수를 새는 변수 ID에 -1을 해주고, 여태 돌이 둬진 좌표들을 저장하는 COORDS 배열에서 좌표를 하나 POP함수를 통해 빼낸 뒤 REDO에 저장한다. BOARD의 REDO좌표를 EMPTY로 바꾼다.

8) CHANG_STATE

```
def change_state(self) :
    if self.turn == BoardState.BLACK: self.turn = BoardState.WHITE
    else : self.turn = BoardState.BLACK
```

차례를 바꾸는 함수이다. 현재 흑 돌 차례였다면 백 돌 차례로 바꿔주고 반대도 해준다.

9) TURN_CHANGE

```
def turn_change(self):
    if computer == BoardState.BLACK :
        computer = BoardState.WHITE
        counter = BoardState.BLACK
    else :
        computer = BoardState.WHITE
        counter = BoardState.BLACK
```

인공지능과 사람의 선수 후수를 바꿔주는 함수이다. 인공지능이 선수 였다면 후수로 바꿔주고 사람을 선수로 바꿔준다. 반대도 해준다.

10) SET_COORDS

```
def set_coords(self):
```

보드판의 각 자리마다의 픽셀 위치를 정해주는 함수이다.

```
for y in range(board_size):
    for x in range(board_size):
        self.pixel_coords.append((x * grid_size + 25, y * grid_size + 25))
```

이중 FOR문으로 15X15 배열을 모두 방문하여 APPEND함수를 사용하여 위치를 추가해준다. 보드판의 가장자리 25를 더하고 그 위치마다의 GRID_SIZE를 곱하여 X * GRID_SIZE + 25를 추가한다. Y도 마찬가지이다.

11) GET_COORD

```
def get_coord(self, pos):
```

```
for coord in self.pixel_coords:
    x, y = coord
    rect = pygame.Rect(x, y, grid_size, grid_size)
    if rect.collidepoint(pos):
        return coord
return None
```

PIXEL_COORDS 들의 좌표를 구해서 그 위치 공간의 객체를 RECT에 저장한다. 인자로 받은 위치가 그 객체 안에 있으면 그 위치를 반환하고, 아니면 아무것도 반환하지 않는다.

12) GET_POINT

```
def get_point(self, coord):
```

인자로 COORD를 입력 받으면 BOARD배열의 어느 위치에 있는 지를 반환하는 함수

이다.

```
x, y = coord
x = (x - 25) // grid_size
y = (y - 25) // grid_size
return x, y
```

GET_COORD에서와 반대로 COORD의 X나 Y에서 25를 뺀 뒤 GRID_SIZE로 나눠주고 반환한다.

13) GET_BOARD

```
def get_board(self) :
    return self.board
```

BOARD 리스트를 반환하는 함수이다,

14) GET_WEIGHTBOARD

```
def get_weightboard(self) :
    return self.weightboard
```

WEIGHTBOARD 리스트를 반환하는 함수이다.

15) CHECK_BOARD

```
def check_board(self, pos):
```

돌을 두는 함수이다.

```
coord = self.get_coord(pos)
```

클릭된 위치를 인자로 받아서 COORD에 저장을 한다.

```
x, y = self.get_point(coord)
if self.board[y][x] != empty:
    return True
else:
    return self.draw_stone(x, y, coord)
```

GET_POINT함수를 사용해서 그 COORD위치가 BOARD배열에서의 어떤 XY인지 얻는다. 그 자리가 비어 있다면 DRAW_STONE함수를 이용해서 돌을 그린다.

16) CHECK_FORBIDDEN

```
def check_forbidden(self):
```

클릭된 위치가 금 수 위치인지 확인하는 함수이다.

```
if self.turn == black_stone:
    coords = self.rule.get_forbidden_points(self.turn)
    while coords:
        x, y = coords.pop()
        x, y = x * grid_size + 25, y * grid_size + 25
        self.draw_image(4, x, y)
    self.is_forbidden = True
```

RULE클래스의 GET_FORBIDDEN_POINT함수를 사용하여 금수자리의 COORDS를 얻는다. 그 위치에 LAST IMAGE를 DRAW IMAGE 함수를 사용해서 그린다. IS_FORBIDDEN을 TRUE로 바꿔준다.

17) DRAW_STONE

```
def draw_stone(self, coord, stone, increase):
```

돌을 그리는 함수이다. 인자로 돌을 그릴 위치와 돌의 색깔, 그리고 돌의 개수를 세는 ID를 얼마나 올릴지를 받는다.

```
if self.is_forbidden:
    self.draw_board()
x, y = self.get_point(coord)
self.board[y][x] = stone
```

IS_FORBIDDEN변수를 사용해서 금 수인지 아닌지 확인한다.

GET_POINT함수를 사용해서 COORD의 배열에서의 위치를 XY에 저장한다. 그 위치의 BOARD배열의 상태를 바꿔준다.

```
self.id += increase
```
ID를 인자로 받은 INCREASE만큼 올려준다.

18) CHECK_GAMEOVER

```
def check_gameover(self, coord):
```
게임 끝났는 지 확인하는 함수이다. 클릭된 위치를 인자로 받는다.

```
x, y = self.get_point(coord)
```
GET_POINT함수를 사용해서 인자로 받은 COORD의 배열에서의 위치를 얻어낸다.

```
if self.id == board_size * board_size:
    self.menu.show_msg(tie)
    return True
```
현재 둔 돌의 개수들이 ,BOARD_SIZE * BOARD_SIZE로 전체 BOARD를 채웠다면 무승부를 MENU의 SHOW_MSG함수를 사용해서 띄운다.

```
elif 5 <= self.rule.is_gameover(x, y, self.turn):
    self.menu.show_msg(self.turn)
    return True
```
그게 아니라면 RULE 클래스에서 IS_GAMEOVER함수를 사용해서 오목이 되었다면 그 돌이 이겼다는 메시지를 MENU의 SHOW_MSG함수를 사용해서 띄운다.

```
return False
```
게임이 끝났다면 TRUE를 반환하고 아니면 FALSE를 반환한다.

6. 인공지능 클래스

```
class AI(object) :
```
오목을 두는 인공지능 클래스이다. 현재 상태에 따라 WEIGHTBOARD의 가중치를 정하고 ALPHA_BETA PRUNING을 통해서 어떤 자리를 선택할지 정하는 클래스이다.

1) 생성자

```
def __init__(self, Omok, currentState, depth) :
    self.__omok = Omok
    self.__currentState = currentState
    self.__depth = depth
    self.__currentI = -1
    self.__currentJ = -1
```
OMOK 클래스와 현재 상태 CURRENTSTATE, 얼마나 깊이 탐색을 할 지 DEPTH와 현재 위치를 받는다. 또 CURRENTI와 CURRENTJ를 -1로 초기화한다.

2) SET_BOARD

```
def set_board(self,i,j,state,) :
    self.__omok.set_board(i,j,state)
```
OMOK클래스의 SET_BOARD함수를 사용해서 돌을 둬보는 함수이다.

3) G_44

```
def G_44(self, board_list, weightboard_list):
    board = np.reshape(board_list, (15,15))
    weightboard = np.reshape(weightboard_list, (15,15))
    two = counter
    zero = 0
```

닫힌 4의 다른 한쪽에 가중치 150000을 더해주는 함수이다. BOARD와 WEIGHTBOARD 리스트를 인자로 받고, NUMPY의 RESHAPE함수를 사용해서 15X15배열로 만들어 BOARD배열에 저장한다. 마찬가지로 WEIGHTBOARD도 해준다. TWO와 ZERO라는 변수를 상대와 EMPTY를 의미하는 0으로 초기화해준다.

```
if board[j][i-2] == computer and board[j][i-1] == counter and board[j][i] == counter and board[j][i+1] == counter and board[j][i+2] == counter and board[j][i+3] == 0:
    weightboard[j][i+3] += G44
```

연결된 네 개의 상대 돌이 있고, 양 쪽 중 한쪽에 인공지능의 돌이 있는 상태가 있다면, 양쪽 중 안 닫힌 한쪽의 WEIGHTBOARD에 가중치 G44를 더해준다.

```
for i in range(2,11) :
    for j in range(0,14) :
```

모든 위치를 확인해 주기위해서 이중 FOR문을 사용했고, 가로와 세로를 모두 확인해줬다.

```
#대각 022221
if board[j-2][i-2] == 0 and board[j-1][i-1] == counter and board[j][i] == counter and board[j+1][i+1] == counter and board[j+2][i+2] == counter and board[j+3][i+3] == computer :
    weightboard[j-2][i-2] += G44
#반대각 122220
if board[j-2][i+2] == computer and board[j-1][i+1] == counter and board[j][i] == counter and board[j+1][i-1] == counter and board[j+2][i-2] == counter and board[j+3][i-3] == 0 :
    weightboard[j+3][i-3] += G44
```

```
for i in range(3,11) :
    for j in range(2,11) :
```

대각 반대각을 확인할 때는 확인할 수 있는 범위가 다르기 때문에 다른 FOR문으로 뺐다.

```
#22202 or 20222 가로
if board[j][i-2] == counter and board[j][i-1] == two and board[j][i] == counter and board[j][i+1] == zero and board[j][i+2] == counter :
    if k == 0 : weightboard[j][i+1] += G44
    elif k == 1 : weightboard[j][i-1] += G44
```

연속된 세개의 상대 돌이 있고 한 칸이 비어 있고 그 다음 칸에 또 상대 돌이 있으면 그 빈 칸의 가중치에 G44를 더해준다.

```
for k in range(2) :
    for i in range(2,12) :
        for j in range(0,14) :
```

22202일 수도 있고 20222일 수도 있기 때문에, FOR문을 하나 더 넣어 K가 0일 때는 전자의 경우 이고, K가 1일 때는 후자의 경우라고 생각해서 가장 밖의 FOR문이 끝났을 땐, TWO와 ZERO의 값을 바꿔준다.

```
temp = two
two = zero
zero = temp
```

```
if board[j][i-2] == counter and board[j][i-1] == counter and board[j][i] == 0 and board[j][i+1] == counter and board[j][i+2] == counter :
    weightboard[j][i] += G44
```

연결된 두 개의 상대 돌이 있고 한 칸이 비어 있고 그 다음 칸에 또 연결된 두 개의 상대 돌이 있다면, 비어 있는 칸의 가중치에 G44를 더해준다.

이 두 FOR 문도 대각 반대각일 때 FOR문의 범위가 달라지기 때문에 따로 빼서 한번 더 탐색했다.

4) G_33

```
def G_33(self, board_list, weightboard_list) :
    board = np.reshape(board_list, (15,15))
    weightboard = np.reshape(weightboard_list, (15,15))
    find33 = 0
    two = counter
    zero = 0
```

열린 3을 방어하는 함수이다. G_44함수와 마찬가지로 BOARD와 WEIGHTBOARD 리스

트를 받아서 RESHAPE함수로 15X15배열을 만들어준다. 열린 3의 개수를 세어 33인지 확인할 수 있는 FIND33을 0으로 초기화 시켜 선언해준다. G_44때와 마찬가지로 TWO와 ZERO를 상대 돌과 EMPTY로 초기화 해준다.

```
if board[j][i-2] == 0 and board[j][i-1] == counter and board[j][i] == counter and board[j][i+1] == counter and board[j][i+2] == 0 :
    weightboard[j][i-2] += G33
    weightboard[j][i+2] += G33
    find33 = find33 + 1
```

연결된 세 개의 백 돌이 있고 양 쪽이 비어 있으면, 양 쪽의 가중치에 G33을 더해준다. 가로와 세로 모두 해주고, 대각과 반대각은 탐색 범위가 달라서 다른 FOR문으로 뺐다.

```
if board[j][i-2] == 0 and board[j][i-1] == counter and board[j][i] ==two and board[j][i+1] == zero and board[j][i+2] == counter and board[j][i+3] == 0 :
    if k == 0 : weightboard[j][i+1] += G33
    elif k == 1 : weightboard[j][i] += G33
    find33 = find33 + 1
```

또한 연결된 두 개의 상대 돌이 있고 한 칸이 비어 있고 그 다음 칸에 상대 돌이 하나 더 있다면 비어 있는 칸의 가중치에 G33을 더해준다.

```
temp = two
two = zero
zero = temp
```

이 때 2202의 경우와 2022의 경우가 있기 때문에 TWO와 ZERO의 변수를 사용해서 두 경우를 모두 탐색해줬다. G_44함수와 마찬가지로 가장 밖의 FOR문 마지막에는 TWO와 ZERO의 값을 바꿔줬다. 마찬가지로 대각 반대각은 탐색 범위가 달라서 다른 FOR문으로 뺐다.

```
return find33
```

탐색을 해줄 때 열린3의 개수를 세는 FIND33에 1씩 더해줬고, FIND33을 반환한다.

5) A_33

```
def A_33(self,board_list, weightboard_list) :
    board = np.reshape(board_list, (15,15))
    weightboard = np.reshape(weightboard_list, (15,15))
    zero = 0
    two = computer
    find33 = 0
    g33 = A33 + 5000
```

열린 3이 있을 때 4 공격을 하도록 가중치를 더해주는 함수이다. BOARD와 WEIGHTBOARD 리스트를 받아서 RESHAPE함수를 사용해 15X15배열을 만들어 준다. 인공지능의 돌과 EMPTY 상태로 TWO와 ZERO를 초기화해주고, 열린 3의 개수를 세어 33인지 확인할 수 있게 FIND33을 0으로 초기화 해준다. 가중치는 g33으로 A33에 5000을 더한 값을 사용한다.

```
if board[j][i-2] == 0 and board[j][i-1] == computer and board[j][i] == computer and board[j][i+1] == computer and board[j][i+2] == 0 :
    weightboard[j][i-2] += g33
    weightboard[j][i+2] += g33
    find33 = find33 + 1
```

연결된 세 개의 인공지능의 돌이 있고 양 쪽이 비어 있으면, 양 쪽의 가중치에 g33을 더해준다. 가로와 세로를 모두 해주고 대각 반대각은 FOR문의 범위가 다르기 때문에 다른 FOR문으로 뺐다.

```
if board[j][i-2] == 0 and board[j][i-1] == computer and board[j][i] == two and board[j][i+1] == zero and board[j][i+2] == computer and board[j][i+3] == 0
    if k == 0 : weightboard[j][i+1] += g33
    elif k == 1 : weightboard[j][i] += g33
    find33 = find33 + 1
```

또한 연결된 두 개의 인공지능 돌이 있고 한 칸이 비어 있고 그 다음 칸에 인공지능 돌이 하나 더 있다면, 비어 있는 칸의 가중치에 g33을 더해준다.

```
temp = two
two = zero
zero = temp
```

이 때 1011일 경우와 1101일 경우가 있기 때문에 TWO와 ZERO의 변수를 사용해서 가장 바깥의 FOR문 마지막에는 TWO와 ZERO의 변수를 바꿨다.

금수인지 33을 확인하기 위해서 매번 3을 찾을 때마다 FIND33에 1을 더해줬다.

6) B_G33

```
def B_G33(self,find33, board_list, weightboard_list) :
    board = np.reshape(board_list, (15,15))
    weightboard = np.reshape(weightboard_list, (15,15))
    zero = 0
    zero2 = 0
    one = computer
    two = counter
```

상대의 닫힌 3이 있는 경우에 가중치를 정하는 함수이다. BOARD와 WEIGHTBOARD 리스트를 받아서 RESHAPE 함수를 사용해서 15X15배열로 만들어 준다. EMPTY 상태의 ZERO 두개와 COMPUTER와 COUNTER 상태의 변수를 선언한다.

```
if board[j][i-2] == one and board[j][i-1] == counter and board[j][i] == counter and board[j][i+1] == counter and board[j][i+2] == zero :
    if k ==0 :
        weightboard[j][i+2] += G33
        if find33 >= 1 : weightboard[j][i+2] -= B33
    elif k == 1 :
        weightboard[j][i-2] += G33
        if find33 >= 1 : weightboard[j][i-2] -= B33
```

상대의 닫힌 3이 있는 경우 12220 이거나 02221인 경우에 0인 자리에 가중치 G33을 더해준다. 만약 열린 3이 있는 경우 B33을 뺀다.

```
if board[j][i-2] == one and board[j][i-1] == counter and board[j][i] == two and board[j][i+1] == zero and board[j][i+2] == counter and board[j][i+3] == zero2 :
    if l ==0 :
        weightboard[j][i+3] += G33
        if find33 >= 1 : weightboard[j][i+3] -= B33
    elif l == 1 :
        weightboard[j][i-2] += G33
        if find33 >= 1 : weightboard[j][i-2] -= B33
```

12202거나 12022인 경우, 또 20221이거나 22021인 경우 빈 자리의 가중치에 G33을 더해준다 마찬가지로 ZERO변수와 ONE, TWO 변수를 FOR문을 사용해서 바꿔가면서 모든 경우를 탐색해준다. 마찬가지로 열린 3이 있는 경우에는 닫힌 3의 빈 부분의 가중치에 B33을 뺀다.

가로 세로와 다르게 대각 반대각은 탐색범위가 다르기 때문에 FOR문을 따로 만들어서 대각 반대각은 따로 탐색을 또 했다.

7) B_A33

```
def B_A33(self, board_list, weightboard_list) :
    board = np.reshape(board_list, (15,15))
    weightboard = np.reshape(weightboard_list, (15,15))
    zero = 0
    two = counter
    one = computer
    zero2 = 0
```

인공지능의 닫힌 3이 있을 때, 다른 한쪽의 4로 공격을 하는 함수이다. BOARD와 WEIGHTBOARD 리스트를 받아서 RESHAPE함수를 사용하여 15X15배열을 만들어준다. EMPTY상태의 변수 두개와 COMPUTER, COUNTER의 상태의 ONE, TWO 변수를 선언해준다.

```
if board[j][i-2] == two and board[j][i-1] == computer and board[j][i] == computer and board[j][i+1] == computer and board[j][i+2] == zero :
    if k == 0 : weightboard[j][i+2] += AB33
    elif k == 1 : weightboard[j][i-2] += AB33
```

21110이거나 01112인 경우 0인 위치의 가중치에 AB33을 더해준다. 가로와 세로 모두 해준다. 대각과 반대각은 탐색 범위가 다르기 때문에 다른 FOR문으로 뺀다.

```
if board[j][i-2] == two and board[j][i-1] == computer and board[j][i] == one and board[j][i+1] == zero and board[j][i+2] == computer and board[j][i+3] == zero2 :
    if l == 0 :
        #211010
        if k == 0: weightboard[j][i+1] += AB33
        #210110
        elif k == 1 : weightboard[j][i+3] += AB33
    if l == 1 :
        #011012
        if k == 0 : weightboard[j][i-2] += AB33
        #010112
        if k == 1 : weightboard[j][i] += AB33
```

또 다른 닫힌 3으로는 210110, 211010, 010112, 011012가 있다. 이 경우도 다 해주기 위해서 ZERO 두개와 ONE, TWO 변수를 사용했다. 마찬가지로 가로와 세로 모두 탐색해주고, 대각 반대각은 탐색 범위가 다르기 때문에 다른 FOR문으로 뺀다.

8) A_44

```
def A_44(self, board_list, weightboard_list) :
    board = np.reshape(board_list, (15,15))
    weightboard = np.reshape(weightboard_list, (15,15))
    zero = 0
    two = counter
    one = computer
```

인공지능의 사목 있을 때 오목을 만드는 함수이다. BOARD와 WEIGHTBOARD 리스트를 받아서 RESHAPE함수를 사용하여 15X15 배열로 만들어 저장한다. EMPTY의 상태인 ZERO와 상대와 인공지능의 상태인 TWO, ONE 변수를 만들어준다.

```
if board[j][i-2] == two and board[j][i-1] == computer and board[j][i] == computer and board[j][i+1] == computer and board[j][i+2] == computer and board[j][i+3] == zero:
    if k == 0 : weightboard[j][i+3] += A44
    elif k == 1 : weightboard[j][i-2] += A44
```

```
temp = two
two = zero
zero = temp
```

211110의 경우와 011112의 경우 0인 위치의 WEIGHTBOARD에 가중치 A44를 더해준다. 이 때 두 경우를 모두 탐색하기 위해서 TWO와 ZERO의 값을 바꿔주는 과정도 거친다.

가로와 세로를 해주고 대각과 반대각의 경우 탐색 범위를 다르게 해줘야 하기 때문에, 다른 FOR문으로 뺀다.

```
if board[j][i-2] == 0 and board[j][i-1] == computer and board[j][i] == computer and board[j][i+1] == computer and board[j][i+2] == computer and board[j][i+3] == zero:
    weightboard[j][i+3] += A44
    weightboard[j][i-2] += A44
```

열린 4의 경우의 가로 세로 대각 반대각도 마찬가지로 탐색해준다. 사목의 경우는 연속된 네 개의 돌이 있는 경우도 있지만, 11101,11011, 10111의 경우도 있기 때문에 이 경우들의 가로 세로 대각 반대각도 모두 검사해서 가중치에 A44를 더해준다.

9) G_22

```
def G_22(self, board_list, weightboard_list) :
    board = np.reshape(board_list, (15,15))
    weightboard = np.reshape(weightboard_list, (15,15))
    find22 = 0
```

상대의 돌이 두개가 연결되어 있고 양 쪽에 아무 돌이 없는 열린 2의 상황에서 양 쪽에 가중치를 더해주는 함수이다. BOARD와 WEIGHTBOARD의 리스트를 받아서 RESHAPE함수를 사용하여 15X15배열을 저장한다. 열린 2의 개수를 셀 FIND22 변수를 0으로 초기화 한다.

```
if board[j][i-1] == 0 and board[j][i] == counter and board[j][i+1] == counter and board[j][i+2] == 0 :
    weightboard[j][i-1] += G22
    weightboard[j][i+2] += G22
    find22 = find22 + 1
```

열린 2가 있는 경우 빈 양 쪽의 WEIGHTBOARD에 G22를 더해준다. 열린 2의 개수를 세기 위해서 FIND22에 1을 더해준다. 열린 2의 개수는 나중에 닫힌 2보다 열린 2의 우선순위를 두는 데에 사용된다.

```
for i in range (1, 12) :
    for j in range (2, 12) :
```

```
for i in range(1,12) :
    for j in range(0,14) :
```

가로와 세로를 모두 탐색하여 가중치를 더해주고, 대각과 반대각은 탐색 범위가 다르기 때문에 다른 FOR문으로 탐색한다.

10) B_22

```
def B_22(self,find22,board_list, weightboard_list) :
    board = np.reshape(board_list, (15,15))
    weightboard = np.reshape(weightboard_list, (15,15))
    zero = 0
    one = computer
```

닫힌 2가 있을 때의 가중치를 결정하는 함수이다. 이 때 열린 2가 있는 경우 가중치를 달리 해주기 위해서 G_22함수에서 얻은 FIND22를 인자로 받는다. BOARD와 WEIGHTBOARD 리스트를 받아서 RESHAPE함수를 이용하여 15X15 배열을 만든다. 비슷한 경우를 같이 비교하기 위해서 EMPTY 상태와 인공지능 상태의 변수 ZERO와 ONE을 선언한다.

```
if board[j][i-1] == one and board[j][i] == counter and board[j][i+1] == counter and  board[j][i+2] == zero :
    if k == 0 :
        weightboard[j][i+2] += G22
        if find22 >= 1 : weightboard[j][i+2] -= B22
    elif k == 1 :
        weightboard[j][i-1] += G22
        if find22 >= 1 : weightboard[j][i-1] -= B22
```

2110의 경우와 0112의 경우를 ONE과 ZERO의 값을 바꿔서 모두 검사해주고 빈 곳에 G22를 더해준다. 만약 열린2가 있으면 B22를 뺀다.

가로와 세로를 같이 검사하고, 대각과 반대각은 탐색 범위가 다르기 때문에 다른 FOR문으로 뺀다.

11) BLACKWEIGHT, WHITEWEIGHT

```
def blackweight(self, board_list, weightboard_list) :
    board = np.reshape(board_list, (15,15))
    weightboard = np.reshape(weightboard_list, (15,15))
```

인공지능의 돌이 있는 경우에 주변 여덟 곳 중 비어 있는 곳에는 가중치 1을 더해주는 함수이다. WHITEWEIGHT함수는 상대의 돌이 있는 경우 주변 여덟 곳 중 비어 있는 곳에 가중치 -1을 더해주는 함수이다.

```
board = np.reshape(board_list, (15,15))
weightboard = np.reshape(weightboard_list, (15,15))
for i in range(board_size) :
    for j in range(board_size) :
        if board[j][i] == BoardState.BLACK :
            for k in range(-1,1) :
                for l in range(-1,1) :
                    if board[l][k] == BoardState.EMPTY : weightboard[l][k] += 1
```

FOR문으로 BOARD 배열의 상태를 모두 확인하고, 인공지능의 돌이 있을 때 그 주변 여덟 곳의 상태 중 EMPTY 상태인 곳의 WEIGHTBOARD에 1을 더해준다. WHITEWEIGHT도 마찬가지로 상대 돌의 주변의 WEIGHTBOARD에 -1을 더해준다.

12) EVALUATE

```
def evaluate(self) :
```

WEIGHTBOARD를 반환하는 배열이다.

```
for i in range(board_size) :
    for j in range(board_size) :
        state_vector.append(self.__omok.get_board()[i][j])
for i in range(board_size) :
    for j in range(board_size) :
        weight_vector.append(self.__omok.get_weightboard()[i][j])
```

현재 보드판의 상태를 받은 STATE_VECTOR와 가중치 배열을 받은 WEIGHT_VECTOR를 선언해 주고 OMOK에서 그 BOARD와 WEIGHTBOARD를 GET_BOARD와 GET_WEIGHTBOARD함수를 통해서 받는다.

```
find22 = self.G_22(state_vector,weight_vector)
(state_vector, weight_vector) = self.B_22(find22,state_vector, weight_vector)
find33 = self.G_33(state_vector, weight_vector)
(state_vector, weight_vector) = self.B_G33(find33,state_vector, weight_vector)
(state_vector, weight_vector) = self.G_44(state_vector, weight_vector)
(state_vector, weight_vector) = self.A_33(state_vector, weight_vector)
(state_vector, weight_vector) = self.B_A33(state_vector, weight_vector)
(state_vector, weight_vector) = self.A_44(state_vector, weight_vector)
(state_vector, weight_vector) = self.blackweight(state_vector, weight_vector)
(state_vector, weight_vector) = self.whiteweight(state_vector, weight_vector)
```

위의 함수들을 모두 사용하여 WEIGHT_VECTOR 배열의 값들을 모두 업데이트 시켜준다.

```
for i in range(board_size):
    for j in range(board_size) :
        evaluate_array.append([weight_vector[j][i],j,i])
```

WEIGHT_VECTOR를 모두 업데이트 한 뒤 EVALUATE_ARRAY에 가중치 값과 그 위치의 좌표를 함께 APPEND 해준다. 그리고 EVALUATE_ARRAY를 반환한다.

13) ALPHA_BETA

```
def alpha_beta(self, depth, nodeIndex, ai, tree, alpha = 10000000, beta = -1000000) :
```

ALPHA BETA PRUNING을 하는 함수이다. 인자로 얼마나 탐색을 할지 깊이 DEPTH, 몇 번째를 탐색하는지 NODEINDEX, 인공지능 클래스, EVALUATE_ARRAY를 받는다. ALPHA와 BETA는 최대 최소값으로 받는다.

```
nextState = counter
nextPlay = AI(deepcopy(self.__omok),nextState, self.__depth + 1)
nextPlay.set_board(tree[nodeIndex][1], tree[nodeIndex][2],self.__currentState)
(val,y,x) = self.alpha_beta(depth+1, nodeIndex*2+1, nextPlay, self.evaluate(), alpha, beta)
```

DEPTH가 짝수인 경우에는 MAX를 찾는다. NEXTSTATE에는 상대방으로 하고, NEXTPLAY라는 AI를 선언했다. NEXTPLAY 상태로 ALPHA BETA를 다시하여 DEPTH에 1을 더하고, 새로운 EVALUATE_ARRAY를 인자로 넣어 새로운 값을 반환한다.

```
if val < best :
    best = val
    (ai.__currentI, ai.__currentJ) = (y,x)

if best < beta :
    beta = best
    (ai.__currentI, ai.__currentJ) = (y,x)
if beta<=alpha : break
i+=1
```

그 값이 ALPHA값보다 작으면 ALPHA값을 다시 갱신해주고, 그 때의 위치의 좌표도 저장해준다. 이 값이 BETA보다 작으면 BREAK를 한다.

```
return best_I,best_J
```

홀수인 경우에는 MIN을 찾는다. 마찬가지로 NEXTPLAY를 선언하여 새로운 ALPHA BETA를 하여 재귀함수로 ALPHA BETA를 하고, 인자로 받은 DEPTH만큼 확인했다면, 그 때의 좌표를 반환한다.

14) FIRST

```
def first(self) :
    self.__omok.check_board((7,7))
    return True
```

처음에는 정 가운데인 7,7에 검은 돌을 두고 시작하는 함수이다. OMOK 클래스의 CHECK_BOARD함수를 사용해서 구현했다.

15) ONE_STEP

```
def one_step(self) :
```

인공지능이 돌을 두는 한 단계를 구현한 함수이다.

```
for i in range(board_size) :
    for j in range(board_size) :
        if self.__omok.get_board()[i][j] != BoardState.EMPTY : continue
node = AI(self.__omok, self.__currentState, self.__depth)
self.alpha_beta(3,0,node,node.evaluate())
(i,j) = (node.__currentI,node.__currentJ)
if not i is None and not j is None :
    self.__omok.check_board((i,j))
    return True
return False
```

OMOK 클래스의 GET_BOARD함수를 사용해서 현재 상태의 배열을 받는다. 그 배열을 사용해서 ALPHA_BETA를 깊이 3으로 진행하고, 그 때의 좌표를 받는다. OMOK 클래스의 CHECK_BOARD함수를 사용해서 그 위치에 돌을 그린다.

7. 메인 함수

```
pygame.init()
```
PYGAME의 INIT함수를 사용하여 모든 PYGAME 모듈을 초기화한다.

```
surface = pygame.display.set_mode((window_width, window_height))
```
WINDOW_WIDTH와 WINDOW_HEIGHT의 크기 변수를 PYGAME의 DISPLAY SET_MODE 함수를 사용해서 그 크기만큼의 DISPLAY를 정해 SURFACE를 정의해준다.

```
pygame.display.set_caption("Omok game")
```
PYGAME모듈의 DISPLAY SET_CAPTION함수를 사용해서 "OMOK GAME"이라고 타이틀 텍스트를 정한다.

```
surface.fill(bg_color)
```
SURFACE를 FILL함수를 사용하여 배경 색상 변수인 BG_COLOR로 배경 색상을 바꾼다.

```
omok = Omok(surface)
menu = Menu(surface)
```
OMOK 클래스와 MENU 클래스를 선언한다.

```
ai = AI(omok,BoardState.BLACK,2)
```
AI 클래스를 BOARDSTATE중 BLACK으로 선언한다.

```
enable_ai = True
```
AI가 둘 수 있게 하는 변수를 TRUE로 먼저 선언한다.

```
ai.first()
```
AI의 FIRST함수를 사용해서 가운데에 두고 게임을 시작한다.

```
result = omok.get_board_result()
```
함수들을 모두 거치고 돌을 두는 곳이 갱신된 BOARD를 얻는 OMOK 클래스의 GET_BOARD_RESULT함수를 사용해서 RESULT 변수에 저장한다.

```
for event in pygame.event.get() :
```
EVENT를 받을 때마다 마우스가 클릭이 됐으면 다음 내용을 수행한다.

```
if enable_ai :
    ai.one_step()
    result = omok.get_board_result()
else : omok.change_state()
```
클릭된 위치의 좌표를 받아서 그 상태를 업데이트하고 AI의 차례라면 AI의 ONE_STEP함수를 사용해서 AI가 두게한다.

```
if omok.is_gameover:
    return
```
OMOK 클래스의 IS_GAMEOVER가 TRUE라면 게임을 끝낸다.

8. 느낀점

3, 4학년이 되고 본격적인 컴퓨터공학과 전공수업을 들으면서, 과제들의 난이도가 1, 2학년 때보다 훨씬 어려워진 것을 느꼈습니다. 1, 2학년 때는, 어렵기도 하고 생소하기도 한 과제들을 하면서도 실제 생활에 완전히 쓰인다는 느낌을 받지 못했습니다. 3학년 때부터 해온 과제들과 이번 과제를 통해서, 여태 배워 온 것들이 어떻게 사용되어 실생활에 적용되는 지를 알게 됐습니다. 더불어 우리 과의 필요성도 많이 느끼게 됐습니다.

처음에 오목 과제를 받았을 때, GUI 구현부터 평가함수 설계와 작동까지 구현한다는 거에 많이 막막하기도 했고, 어떻게 하면 될 거라고 바로 머릿 속에 그려지기도 해서 잘 될 거라고 믿기도 했습니다. 본격적으로 구현을 시작했을 때, 파이썬 언어 자체를 3학년 때 과제하면서 처음 써봐서 익숙하지는 않았습니다. GUI개발도 해본 적이 없어서 많이 막막했습니다. 인터넷에서 pygame 모듈을 이용해서 gui를 개발하는 거에 많은 도움을 받았고, 또 공부가 많이 됐습니다. 가중치를 정하고 alpha beta pruning을 구현하는 데에는 많은 힘이 들지는 않았습니다. 2학년 때 들은 자료구조 수업 때도, 알고리즘 설명을 듣고 그대로 구현하는 훈련을 받았던 게 도움이 됐던 것 같습니다.

이번 과제는 다른 전공 사람들에게 우리는 과에서 무엇을 한다는 것을 제대로 설명할 수 있게 하는 과제였다고 생각합니다. 여태까지 배우고 쌓아온 것들도 많이 생각해보게 됐고, 저학년 때는 이런 걸 왜 구현을 하나 가끔 의문이 있긴 했는데, 그런 것들이 다 도움이 되고 나중에는 실생활 적용 프로그램에 이용이 된다는 것을 많이 느꼈습니다.

9. 참고

금 수 : https://blog.naver.com/jko673/220413355511

가중치 참고 : https://ku-hug.tistory.com/2

Gui : https://blog.naver.com/dnpc7848/221503651970