프로젝트 설계 기술서

자료구조 - 오목 프로그램

과목	자료구조
학과	전자공학과
학번	2011144024
이름	유대성 (월 5-6)
제출일	2018-06-22
담당교수	장지웅 교수님

1. 주제 및 배경이론

오목이란 무엇이며 게임 룰과 승리 조건에 대해 알아보자.

오목이란?

오목은 오목알(바둑알)을 사용하여 가로, 세로, 대각선 경로로 다섯개의 돌을 연달아 놓으면 승리하는 게임이다. 바둑보다는 진입장벽이 낮고 간단한 룰로 많은 사람들이 즐겨하는 보드 게임이다. 오목의 기원은 명확하지 않지만 중국 황허강 유역에서 오목과 유사한 '격오(格五)'라는 명칭이 존재한 것으로 보아 오랜 역사를 지닌 게임으로 이해할 수 있겠다. 이런 오랜 역사와 관심을 가지는 만큼 다양한 룰이 존재하고 최근 이슈가 된 알파고와 같은 인공지능 프로그램인 Yixin 도 탄생하게 되었다.

오목의 룰

게임 장비 : 15 x 15 또는 19 x 19 의 격자 형태의 보드와 흰색과 검은색의 오목알(바둑알)을 사용

공식 게임룰 : 오목은 흑에게 유리한 필승법이 존재한다. 따라서 이를 막기 위한 '렌주룰'과 같은 방법이 사용되고 있다.

- 렌주룰 : 흑의 유리함을 막고자 흑에게 여러가지 금지수를 추가하였다. 3-3, 4-4 와 장목(6 목 이상을 두는 것)을 금지하고 이를 어겼을 시 흑에게 반칙패를 준다.

선수와 포석 : 선수는 체스에서 체크, 장기에서 장군과 같이 상대에게 수비를 강요하는 상황을 말한다. 열린 3 목, 4 목 같은 경우 이를 막지 않으면 다음 턴 혹은 그 다음 턴에 무조건 패배하게 되므로 이를 선수로 칭하게 된다. 포석은 직접적인 공격은 아니지만 이후에 원활한 공격을 위해 두는 수를 의미한다.

주형 : 1 턴. 즉, 흑의 첫 수는 오목판의 정 중앙인 천원에 두어야 한다. 2 턴. 즉, 백의 첫 수는 그 대응수로 가로, 세로, 대각선 한 칸 이내로 두어야 한다.

이번 프로젝트에서 사용하는 룰

이번 프로젝트에서는 기본적으로 렌주룰을 사용하지만 주형은 따르지 않는다. 게임에 대해 제대로 이해하기 위해 아래에 이번 프로젝트의 룰을 적어본다.

1. 프로그램을 완성시키지 못했거나 실행되지 않을 경우 실격패
2. 프로그램이 일련의 버그로 인해 판정을 잘못 수행할 경우엔 무승부
3. 이미 돌이 있는 위치에 수를 놓을 경우 패배
4. 흑은 3-3, 4-4, 장목을 금지수로 하며 이를 두었을 시 반칙패
5. 백이 장목을 두면 승리
6. 0.5 초의 제한시간이 있으며 이를 어겼을 시 패배

2. 프로그램 구상

문제점을 인식하고 승리하는 프로그램을 고안해보자.

1 턴 : 흑의 1 수

흑의 1 수는 천원에 두도록 한다.

- 흑의 위치를 외곽에 두어 상대의 실수를 유발하는 방법은 정정당당하지 못하다고 판단하여 공식 룰을 따르도록 했다.

2 턴 : 백의 1 수

흑의 첫수를 파악하여 그에 상응하는 위치에 돌을 두도록 한다.

- 흑색 플레이어가 천원이 아닌 곳에 착수할 수 있으므로 다음의 룰을 따른다
  1. 외곽에서 3 칸 이하 거리에 착수한 경우 : 착수 위치로부터 안쪽으로 직선으로 3 칸 떨어진 위치에 착수한다.
     - second_side_marking_2011144024() 함수 사용
  2. 천원으로부터 2 칸 이하 거리에 착수한 경우 : Yixin 프로그램의 인공지능이 두는 위치를 직접 입력하여 넣음
     - second_center_marking_2011144024() 함수 사용
  3. 그 외의 위치에 두었을 경우 설계한 포인트 업데이트 알고리즘을 따르도록 한다.

3 턴 : 흑의 2 수

백의 첫수를 파악하여 그에 상응하는 위치에 돌을 두도록 한다.

- 백색 플레이어는 무조건 게임판의 균형을 깨뜨리는 위치에 착수할 수 밖에 없으므로 그 상태에 맞게 공격하도록 한다.
  1. 천원으로부터 1 칸 거리에 착수한 경우 : Yixin 프로그램의 인공지능이 두는 위치를 직접 입력하여 넣음
     - third_one_length_marking_2011144024() 함수 사용
  2. 천원으로부터 2 칸 거리에 착수한 경우 : Yixin 프로그램의 인공지능이 두는 위치를 직접 입력하여 넣음
     - third_two_length_marking_2011144024() 함수 사용
  3. 그 외의 위치에 두었을 경우 설계한 포인트 업데이트 알고리즘을 따르도록 한다.

4 턴 이후

이후에는 포인트 업데이트 알고리즘 을 따라서 착수한다. 각각의 상태는 오목판의 각 점(착수가 가능한 위치)의 상태 값을 업데이트하고 변경된 상태 값 중 가장 큰 값에 돌을 두게 한다. 이 상태 값을 업데이트 하는 함수가 바로 이 프로그램의 핵심이 될 것 이다.

3. 주요 변수 및 사용자 정의 데이터

프로그램 전반적으로 사용되는 주요 변수와 사용자 정의형을 소개한다.

```cs
#define BOARD_SIZE_2011144024 19
#define BOARD_MID_2011144024 9
#define BLACK -1
#define WHITE 1
```

BOARD_SIZE_2011144024 : 게임 보드의 가로, 세로 크기를 정의하는 사용자 정의 값
BOARD_MID_2011144024 : 게임 보드의 가로, 세로의 중앙을 정의하는 사용자 정의 값
BLACK, WHITE : 게임 돌의 흑백을 정의하는 값

```cs
typedef struct GAME_BOARD_2011144024 {
    int color;
    int xMax;
    int yMax;
    int uMax;
    int dMax;
    int point;
} GAME_BOARD_2011144024;

GAME_BOARD_2011144024 game_board_2011144024[BOARD_SIZE_2011144024*BOARD_SIZE_2011144024] = { 0 };
```

GAME_BOARD_2011144024 : 게임 판 내의 점 하나를 정의하는 사용자 정의 형
game_board_2011144024 : 전체 게임 판을 정의하는 1 차원 배열

```cs
char mode_2011144024 = '\n';        // A, B, C ,D
int last_stone_x_2011144024 = -99;
int last_stone_y_2011144024 = -99;
int player_black_turn_2011144024 = 0;
int player_white_turn_2011144024 = 0;
```

mode_2011144024 : 천원이 아닌 점에 착수하였을 때 바둑판의 대칭성이 깨지는 순간 업데이트 된다.

- E, W, S, N : 천원을 중심으로 동서남북을 의미
- A, B, C, D : 천원을 중심으로 서북, 동북, 동남, 서남 쪽을 의미

4. 연계 함수

심판 프로그램을 사용하기 위해 필수적으로 사용되야 하는 함수를 정의

```cs
void WhiteAttack_2011144024(int *x, int *y)
void WhiteDefence_2011144024(int x, int y)
void AttackBlack_2011144024(int *x, int *y)
void DefenceBlack_2011144024(int x, int y)
```

WhiteAttack_2011144024 : 백 돌이 착수했을 때 심판 프로그램에 착수한 돌의 위치를 알려주는 함수
WhiteDefence_2011144024 : 흑 돌이 착수한 값을 심판 프로그램으로부터 안내 받는 흰색 플레이어의 함수
AttackBlack_2011144024 : 흑 돌이 착수했을 때 심판 프로그램에 착수한 돌의 위치를 알려주는 함수
DefenceBlack_2011144024 : 백 돌이 착수한 값을 심판 프로그램으로부터 안내 받는 흑색 플레이어의 함수

5. 포인트 업데이트 함수

각 점은 점수를 가지며 이 점수를 업데이트 한 후에 가장 점수가 높은 점에 착수하게 된다.

흑색 플레이어

```cs
void hot_sopt_point_making_2011144024(int inner_x, int inner_y, int type)
void near_by_other_stone_blocking(int inner_x, int inner_y, int type)
void escape_foul_black_turn_2011144024(int inner_x, int inner_y)
int center_point_update_2011144024(int inner_x, int inner_y)
void near_by_point_update_2011144024(int inner_x, int inner_y, int type)
```

hot_sopt_point_making_2011144024 : 3 목, 4 목이 되는 주요 위치의 값을 업데이트 하는 함수

near_by_other_stone_blocking : 주변에 막고있는 돌이 있는 지 체크하여 그 값을 업데이트 하는 함수

escape_foul_black_turn_2011144024 : 흑돌이 금지수에 두지 못하도록 값을 업데이트 하는 함수

center_point_update_2011144024 : 되도록 천원에 가깝게 두도록 값을 업데이트 하는 함수

near_by_point_update_2011144024 : 오목에서 주요한 형태인 반 삼각, L 형, 호구형을 만들 수 있도록 그 값을 업데이트 하는 함수

백색 플레이어

```cs
void hot_sopt_point_making_2011144024(int inner_x, int inner_y, int type)
void near_by_other_stone_blocking(int inner_x, int inner_y, int type)
void long_linear_white_turn_2011144024(int inner_x, int inner_y)
int center_point_update_2011144024(int inner_x, int inner_y)
void near_by_point_update_2011144024(int inner_x, int inner_y, int type)
```

hot_sopt_point_making_2011144024 : 3 목, 4 목이 되는 주요 위치의 값을 업데이트 하는 함수

near_by_other_stone_blocking : 주변에 막고있는 돌이 있는 지 체크하여 그 값을 업데이트 하는 함수

long_linear_white_turn_2011144024 : 백색 플레이어가 6 목 이상으로 승리할 수 있도록 그 값을 업데이트 하는 함수

center_point_update_2011144024 : 되도록 천원에 가깝게 두도록 값을 업데이트 하는 함수

near_by_point_update_2011144024 : 오목에서 주요한 형태인 반 삼각, L 형, 호구형을 만들 수 있도록 그 값을 업데이트 하는 함수

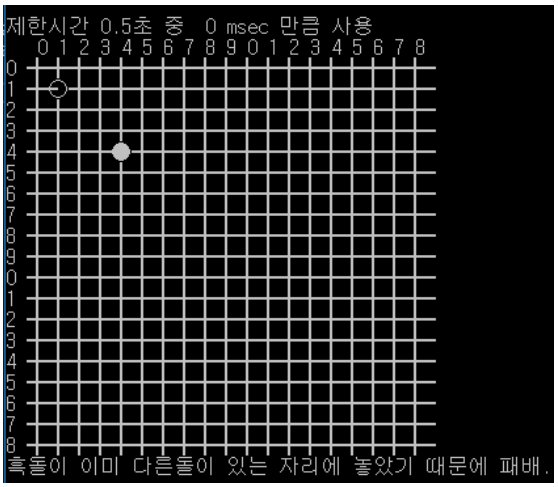
착수 우선 순위(청색 : 공격, 적색 : 수비)

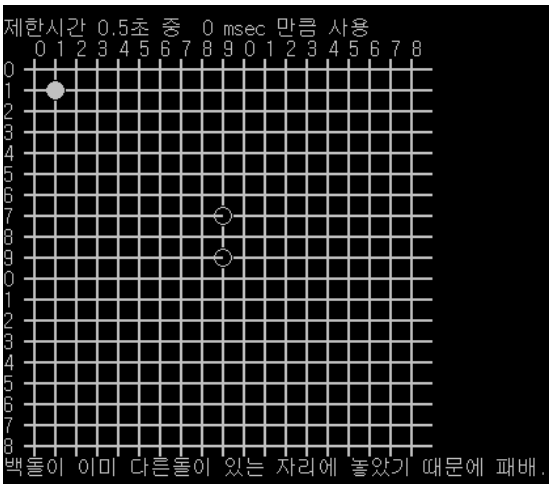
상태	순위	상태	순위	상태	순위
5 목이 되는 위치	1	3 목이 되는 위치	7	L 형이 되는 수	13
5 목이 되는 위치	2	3 목이 되는 위치	8	L 형이 되는 수	14
4 목이 되는 위치	3	한 칸 띈 3 목이 되는 위치	9	반 삼각을 만들 수 있는 위치	15
4 목이 되는 위치	4	한 칸 띈 3 목이 되는 위치	10	반 삼각을 만들 수 있는 위치	16
한 칸 띈 4 목이 되는 위치	5	반 삼각, 호구형이 되는 수	11	L 형을 만들 수 있는 위치	17
한 칸 띈 4 목이 되는 위치	6	반 삼각, 호구형이 되는 수	12	L 형을 만들 수 있는 위치	18
근처에 같은 색 돌이 있을 경우	+	근처에 다른 색 돌이 있을 경우	-	천원에 가까운 위치	+

6. 업데이트 목록

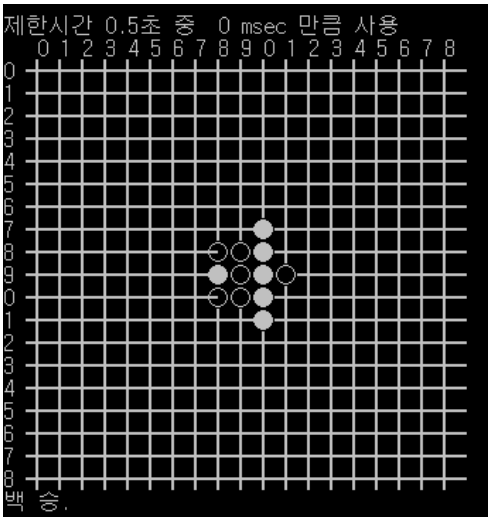
완벽한 프로그램을 만들기 위해 거쳐갔던 이전 프로그램들. 또 그들과 완성된 프로그램과의 배틀 결과를 소개합니다.

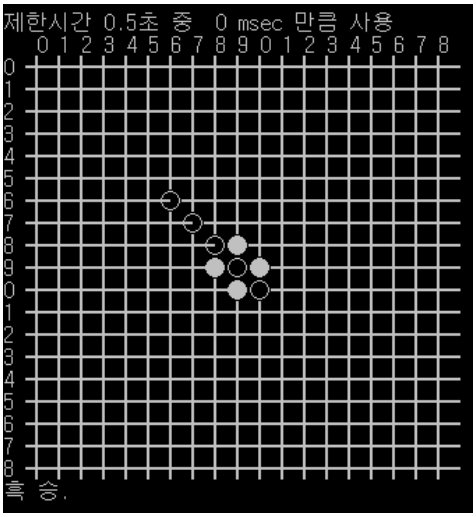
2011144024_유대성_rev01.h (흑 - 착수 에러 / 백 - 착수 에러)



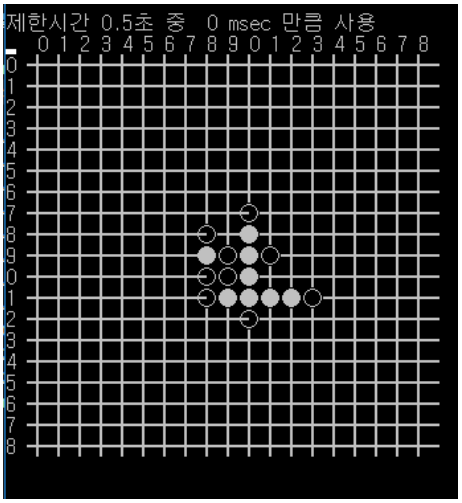


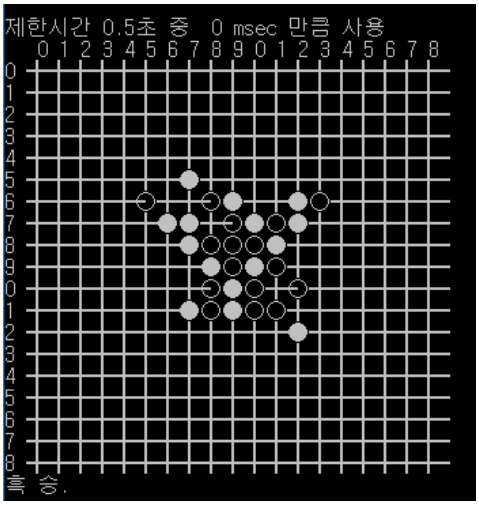
2011144024_유대성_rev02.h (흑 - 패배/ 백 - 패배)



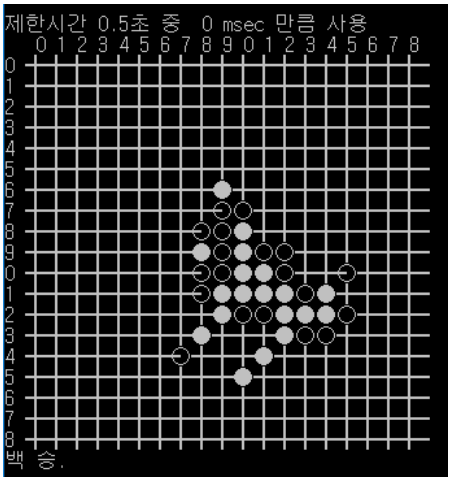


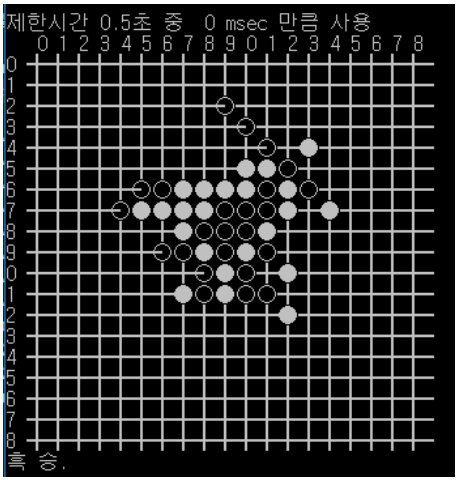
2011144024_유대성_rev03.h (흑 - 이상 종료/ 백 - 패배)



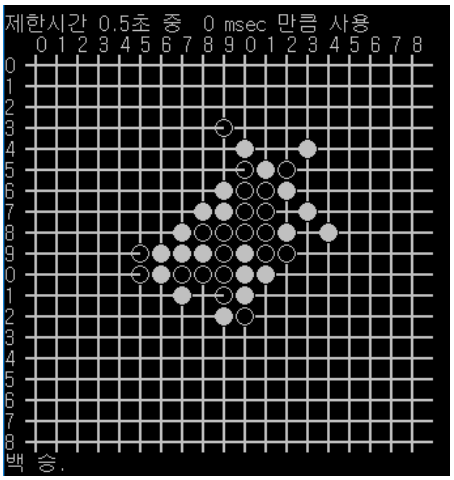


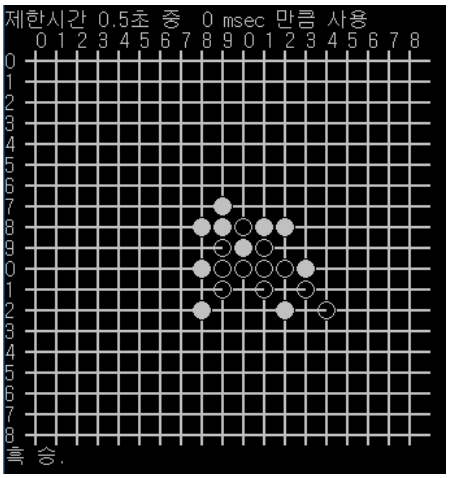
2011144024_유대성_rev04.h (흑 - 패배/ 백 – 패배)



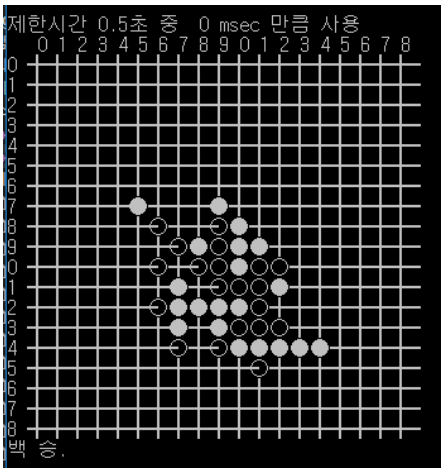


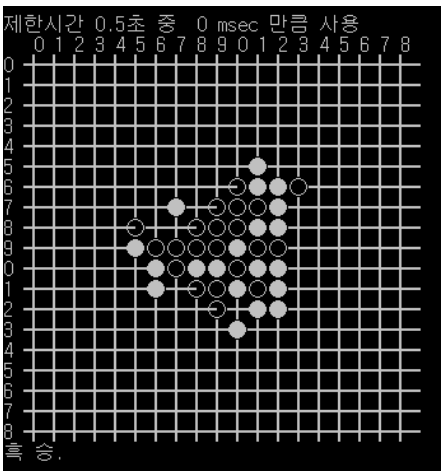
2011144024_유대성_rev05.h (흑 - 패배/ 백 – 패배)



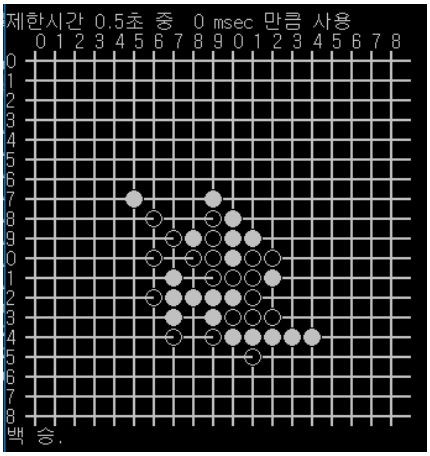


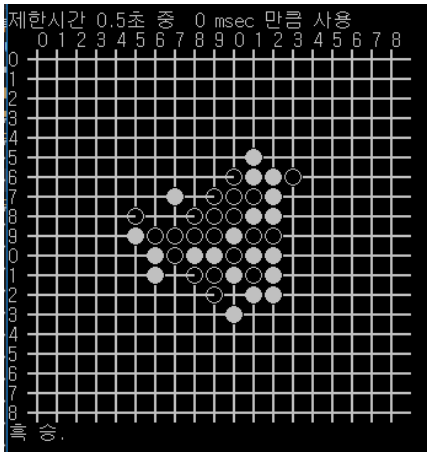
2011144024_유대성_rev06.h (흑 - 패배/ 백 – 패배)



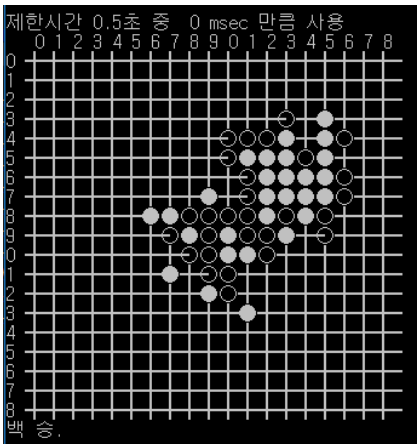


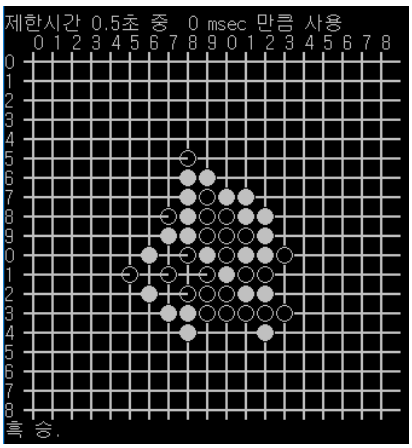
2011144024_유대성_rev07.h (흑 - 패배/ 백 – 패배)





2013180010_박정훈.h (흑 - 패배/ 백 – 패배)





7. 핵심 함수 및 프로그래밍 기법 소개

핵심 함수를 소개하고 프로그래밍 기법을 소개한다.

주요 함수들은 대부분 재귀함수를 통해 구현하였으며 한 칸 띈 상태를 인식시키키 위해 아래와 같은 E(Empty) - 변수를 추가하여 사용하였습니다.

```cs
int PX = 0, PY = 0, PU = 0, PD = 0, MX = 0, MY = 0, MU = 0, MD = 0;
    xEnd_check_2011144024(inner_x, inner_y, type, 1, 0, &PX);
    xEnd_check_2011144024(inner_x, inner_y, type, -1, 0, &MX);
    yEnd_check_2011144024(inner_x, inner_y, type, 0, 1, &PY);
    yEnd_check_2011144024(inner_x, inner_y, type, 0, -1, &MY);
    uEnd_check_2011144024(inner_x, inner_y, type, 1, -1, &PU);
    uEnd_check_2011144024(inner_x, inner_y, type, -1, 1, &MU);
    dEnd_check_2011144024(inner_x, inner_y, type, 1, 1, &PD);
    dEnd_check_2011144024(inner_x, inner_y, type, -1, -1, &MD);
int E_PX = 0, E_PY = 0, E_PU = 0, E_PD = 0, E_MX = 0, E_MY = 0, E_MU = 0, E_MD = 0;
    if (PX == 0) xEnd_check_2011144024(inner_x + 1, inner_y, BLACK, 1, 0, &E_PX);
    if (MX == 0) xEnd_check_2011144024(inner_x - 1, inner_y, BLACK, -1, 0, &E_MX);
    if (PY == 0) yEnd_check_2011144024(inner_x, inner_y + 1, BLACK, 0, 1, &E_PY);
    if (MY == 0) yEnd_check_2011144024(inner_x, inner_y - 1, BLACK, 0, -1, &E_MY);
    if (PU == 0) uEnd_check_2011144024(inner_x + 1, inner_y - 1, BLACK, 1, -1, &E_PU);
    if (MU == 0) uEnd_check_2011144024(inner_x - 1, inner_y + 1, BLACK, -1, 1, &E_MU);
    if (PD == 0) dEnd_check_2011144024(inner_x + 1, inner_y + 1, BLACK, 1, 1, &E_PD);
    if (MD == 0) dEnd_check_2011144024(inner_x - 1, inner_y - 1, BLACK, -1, -1, &E_MD);
```

함수 내부에 함수를 선언하는 방식으로 체이닝을 사용하였습니다.

```cs
point_update_black_2011144024();
```

```cs
void point_update_black_2011144024() {
    for (int i = 0; i < BOARD_SIZE_2011144024; i++) {
        for (int j = 0; j < BOARD_SIZE_2011144024; j++) {
            if (game_board_2011144024[BOARD_SIZE_2011144024 * i + j].color == 0) {
                hot_sopt_point_making_2011144024(j, i, BLACK);
                near_by_other_stone_blocking(j, i, WHITE);
                escape_foul_black_turn_2011144024(j, i);
            }
            else {
                game_board_2011144024[BOARD_SIZE_2011144024 * j + i].point += center_point_update_2011144024(j, i);
                near_by_point_update_2011144024(j, i, BLACK);
            }
        }
    }
}
```



```cs
void near_by_point_update_2011144024(int inner_x, int inner_y, int type) {
    if (inner_x > 2 && inner_x < BOARD_SIZE_2011144024 - 3 && inner_y > 2 && inner_y < BOARD_SIZE_2011144024 - 3) {
        if (game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].color == type) near_by_same_color_point_update_2011144024
(inner_x, inner_y, type);
        else if (game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].color == -
1 * type) near_by_other_color_point_update_2011144024(inner_x, inner_y, -1 * type);
    }
}
```



```cs
void near_by_same_color_point_update_2011144024(int inner_x, int inner_y, int type) {
    raise_point_use_XY_2011144024(inner_x, inner_y, -1, 0, 3, type);
    raise_point_use_XY_2011144024(inner_x, inner_y, 0, -1, 3, type);
    raise_point_use_XY_2011144024(inner_x, inner_y, 1, 0, 3, type);
    raise_point_use_XY_2011144024(inner_x, inner_y, 0, 1, 3, type);
```

게임모드 설정 함수

상대의 첫수를 파악하여 그에 맞게 다음 수를 사용하기 위해 변수를 하나 사용하고 이 변수는 상대의 첫 수 이후에 단 한 번만 업데이트 됨.

```
void catch_mode_2011144024(int inner_x, int inner_y) {        // East West South North
    if (inner_x < BOARD_MID_2011144024 && inner_y < BOARD_MID_2011144024) mode_2011144024 = 'A';
    else if (inner_x > BOARD_MID_2011144024 && inner_y < BOARD_MID_2011144024) mode_2011144024 = 'B';
    else if (inner_x > BOARD_MID_2011144024 && inner_y > BOARD_MID_2011144024) mode_2011144024 = 'C';
    else if (inner_x < BOARD_MID_2011144024 && inner_y > BOARD_MID_2011144024) mode_2011144024 = 'D';
    else if (inner_y == BOARD_MID_2011144024 && inner_x > BOARD_MID_2011144024) mode_2011144024 = 'E';
    else if (inner_y == BOARD_MID_2011144024 && inner_x < BOARD_MID_2011144024) mode_2011144024 = 'W';
    else if (inner_y > BOARD_MID_2011144024 && inner_x == BOARD_MID_2011144024) mode_2011144024 = 'S';
    else if (inner_y < BOARD_MID_2011144024 && inner_x == BOARD_MID_2011144024) mode_2011144024 = 'N';
    else mode_2011144024 = '\n';
}
```


흰색 플레이어의 공격함수

흰색플레이어의 공격 턴에 오목판 상태를 체크하여 각 점에 점수를 부과한다. 이 점수가 가장 높은 점에 착수하게 된다.

```
void point_update_white_2011144024() {
    for (int i = 0; i < BOARD_SIZE_2011144024; i++) {
        for (int j = 0; j < BOARD_SIZE_2011144024; j++) {
            if (game_board_2011144024[BOARD_SIZE_2011144024 * i + j].color == 0) {
                hot_sopt_point_making_2011144024(j, i, WHITE);
                near_by_other_stone_blocking(j, i, BLACK);
                long_linear_white_turn_2011144024(j, i);
            }
            else {
                game_board_2011144024[BOARD_SIZE_2011144024 * j + i].point += center_point_update_2011144024(j, i);
                near_by_point_update_2011144024(j, i, WHITE);
            }
        }
    }
}
```


흑색 플레이어의 공격함수

흑색플레이어의 공격 턴에 오목판 상태를 체크하여 각 점에 점수를 부과한다. 이 점수가 가장 높은 점에 착수하게 된다.

```
void point_update_black_2011144024() {
    for (int i = 0; i < BOARD_SIZE_2011144024; i++) {
        for (int j = 0; j < BOARD_SIZE_2011144024; j++) {
            if (game_board_2011144024[BOARD_SIZE_2011144024 * i + j].color == 0) {
                hot_sopt_point_making_2011144024(j, i, BLACK);
                near_by_other_stone_blocking(j, i, WHITE);
                escape_foul_black_turn_2011144024(j, i);
            }
            else {
                game_board_2011144024[BOARD_SIZE_2011144024 * j + i].point += center_point_update_2011144024(j, i);
                near_by_point_update_2011144024(j, i, BLACK);
            }
        }
    }
}
```

대응 수 함수

상대의 착수에 맞는 대응 수로 반격하는 함수

```cs
void second_side_marking_sub_2011144024(int inner_x, int inner_y, int *target_x, int *target_y, int add_x, int add_y)
void second_side_marking_2011144024(int inner_x, int inner_y, int *target_x, int *target_y)
void second_mid_marking_2011144024(int inner_x, int inner_y, int *target_x, int *target_y)
void second_center_marking_2011144024(int inner_x, int inner_y, int *target_x, int *target_y)
void third_one_length_marking_2011144024(int *inner_x, int *inner_y)
void third_two_length_marking_2011144024(int inner_x, int inner_y, int *target_x, int *target_y)
```

연속된 돌의 개수를 업데이트

게임판 내의 돌 마다 가로, 세로, 대각선으로 연속된 돌의 개수를 업데이트 하는 함수, 매우 중요한 역할을 한다.

```cs
void max_update_2011144024() {
    int PX, PY, PU, PD, MX, MY, MU, MD;
    for (int i = 0; i < BOARD_SIZE_2011144024; i++) {
        for (int j = 0; j < BOARD_SIZE_2011144024; j++) {
            if (game_board_2011144024[BOARD_SIZE_2011144024 * i + j].color != 0) {
                PX = 0; PY = 0; PU = 0; PD = 0; MX = 0; MY = 0; MU = 0; MD = 0;
                xEnd_check_2011144024(j, i, game_board_2011144024[BOARD_SIZE_2011144024 * i + j].color, 1, 0, &PX);
                xEnd_check_2011144024(j, i, game_board_2011144024[BOARD_SIZE_2011144024 * i + j].color, -1, 0, &MX);
                game_board_2011144024[BOARD_SIZE_2011144024 * i + j].xMax = 1 + PX + MX;
                yEnd_check_2011144024(j, i, game_board_2011144024[BOARD_SIZE_2011144024 * i + j].color, 0, 1, &PY);
                yEnd_check_2011144024(j, i, game_board_2011144024[BOARD_SIZE_2011144024 * i + j].color, 0, -1, &MY);
                game_board_2011144024[BOARD_SIZE_2011144024 * i + j].yMax = 1 + PY + MY;
                uEnd_check_2011144024(j, i, game_board_2011144024[BOARD_SIZE_2011144024 * i + j].color, 1, -1, &PU);
                uEnd_check_2011144024(j, i, game_board_2011144024[BOARD_SIZE_2011144024 * i + j].color, -1, 1, &MU);
                game_board_2011144024[BOARD_SIZE_2011144024 * i + j].uMax = 1 + PU + MU;
                dEnd_check_2011144024(j, i, game_board_2011144024[BOARD_SIZE_2011144024 * i + j].color, 1, 1, &PD);
                dEnd_check_2011144024(j, i, game_board_2011144024[BOARD_SIZE_2011144024 * i + j].color, -1, -1, &MD);
                game_board_2011144024[BOARD_SIZE_2011144024 * i + j].dMax = 1 + PD + MD;
            }
        }
    }
}
```


연속된 돌 개수 카운트

연속된 돌의 개수를 count 변수에 저장하는 함수

```cs
void xEnd_check_2011144024(int inner_x, int inner_y, int type, int move_x, int move_y, int *count)
void yEnd_check_2011144024(int inner_x, int inner_y, int type, int move_x, int move_y, int *count)
void uEnd_check_2011144024(int inner_x, int inner_y, int type, int move_x, int move_y, int *count)
void dEnd_check_2011144024(int inner_x, int inner_y, int type, int move_x, int move_y, int *count)
```

연속된 돌을 체크

연속된 돌을 체크하여 공격, 수비가 가능하게 하는 핵심 함수. 본인에 턴에는 수비보다는 공격이 중요하므로(똑같이 3 목인 경우 4 목을 두어 수비를 유도하는 쪽이 이후의 상황을 유리하게 만들 수 있다) 공격 수에 가중치를 둠.

```cs
void hot_sopt_point_making_2011144024(int inner_x, int inner_y, int type) {
    hot_sopt_point_attack_making_2011144024(inner_x, inner_y, type);
    hot_sopt_point_shield_making_2011144024(inner_x, inner_y, -1 * type);
}
```

연속된 돌의 끝이 열렸는 지 막혀있는 지 체크

끝이 상대 돌로 막혀있으면 1, 끝이 비어있으면 0 을 리턴

```cs
int xBlock_check_2011144024(int inner_x, int inner_y, int type, int move_x, int move_y)
int yBlock_check_2011144024(int inner_x, int inner_y, int type, int move_x, int move_y)
int uBlock_check_2011144024(int inner_x, int inner_y, int type, int move_x, int move_y)
int dBlock_check_2011144024(int inner_x, int inner_y, int type, int move_x, int move_y)
```

중앙과 가까운 위치에 가산점 부여

천원에서 가까운 만큼 가산점을 부여해 중앙 위주로 착수하게 유도함

```cs
int center_point_update_2011144024(int inner_x, int inner_y) {
    int result_x = BOARD_MID_2011144024 - inner_x;
    int result_y = BOARD_MID_2011144024 - inner_y;
    result_x = (result_x > 0) ? result_x : -1 * result_x;
    result_y = (result_y > 0) ? result_y : -1 * result_y;

    if (result_x > result_y) {
        return BOARD_MID_2011144024 - result_x;
    }
    else {
        return BOARD_MID_2011144024 - result_y;
    }
}
```

Colored by Color Scripter

모양 만들기

오목에는 3 목, 4 목 말고도 빈삼각, L 자, 호구형과 같은 주요 공격수들이 존재한다. 이 공격수를 인식시켜 효율적인 공격이 가능하게 한다.

```cs
void new_shape_making_2011144024(int inner_x, int inner_y, int x_shift, int y_shift, int type)
```

연계 공격이 가능한 위치 체크

오목판에 이미 돌이 착수되어 있으면 그 돌을 사용한 연계공격을 할 수 있다. 현재 놓인 돌의 위치로부터 가로 세로 3 칸 거리의 모든 점에 각각의 가산점을 부여한다. (L 자형, 빈삼각, 호구형을 만들 수 있는 위치에 가산점을 부여하였다)

```cs
void near_by_point_update_2011144024(int inner_x, int inner_y, int type) {
    if (inner_x > 2 && inner_x < BOARD_SIZE_2011144024 - 3 && inner_y > 2 && inner_y < BOARD_SIZE_2011144024 - 3) {
        if (game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].color == type) near_by_same_color_point_update_2011144024
(inner_x, inner_y, type);
        else if (game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].color == -
1 * type) near_by_other_color_point_update_2011144024(inner_x, inner_y, -1 * type);
    }
}
```

Colored by Color Scripter

흑색플레이어 금지수 제외

흑색플레이어는 금지수에 두면 패배한다는 패널티가 있다. 반칙배를 막기위해 해당 점에 절대 두지 못하도록 가장 낮은 점수를 부과하도록 한다.

```cs
void escape_foul_black_turn_2011144024(int inner_x, int inner_y)
```

여기서 장목의 패널티 옵션이 가장 큰 이유는 다음과 같다.

1. 3-3, 4-4 를 만들어도 5 목이 완성되는 위치에 착수하게 되면 흑색 플레이어가 승리한다.
2. 6 목 이상을 만들게 되면 어떠한 경우에도 패배한다.
3. 5 목을 두는 수는 +(양수), 장목 또는 3-3, 4-4 를 두는 수는 - (음수)의 포인트를 갖는다.
4. 6 목은 무조건 회피해야 하며 3-3, 4-4 는 5 목이 되지 않는 경우만 회피하면 된다.
5. 세 경우의 절대값을 비교하면 다음의 관계식이 성립한다.

장목 〉 5 목 〉 3-3 또는 4-4

5 목의 점수

```cs
else if (PY == 4 && MY == 0) game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].point += 1000001;
else if (PY == 0 && MY == 4) game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].point += 1000001;
else if (PY == 3 && MY == 1) game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].point += 1000001;
else if (PY == 1 && MY == 3) game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].point += 1000001;
else if (PY == 2 && MY == 2) game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].point += 1000001;
```

반칙 수의 점수

```cs
if (over_six_check_using_foul > 0) game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].point += -99999999;
else if (fourfour_count_using_foul > 1) game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].point += -999999;
else if (threethree_count_using_foul > 1) game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].point += -999999;
```

백색플레이어 공격루트 확보

백색플레이어는 장목으로 공격루트를 하나 더 만들 수 있게 된다. 그러므로 6 목 이상을 만드는 위치에 착수하는 것 또한 점수로 인식하게 한다.

```cs
void long_linear_white_turn_2011144024(int inner_x, int inner_y) {

    int PX = 0, PY = 0, PU = 0, PD = 0, MX = 0, MY = 0, MU = 0, MD = 0;
    xEnd_check_2011144024(inner_x, inner_y, BLACK, 1, 0, &PX);
    xEnd_check_2011144024(inner_x, inner_y, BLACK, -1, 0, &MX);
    yEnd_check_2011144024(inner_x, inner_y, BLACK, 0, 1, &PY);
    yEnd_check_2011144024(inner_x, inner_y, BLACK, 0, -1, &MY);
    uEnd_check_2011144024(inner_x, inner_y, BLACK, 1, -1, &PU);
    uEnd_check_2011144024(inner_x, inner_y, BLACK, -1, 1, &MU);
    dEnd_check_2011144024(inner_x, inner_y, BLACK, 1, 1, &PD);
    dEnd_check_2011144024(inner_x, inner_y, BLACK, -1, -1, &MD);

    if (PX + MX >= 5) game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].point += 99999999;
    else if (PY + MY >= 5) game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].point += 99999999;
    else if (PU + MU >= 5) game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].point += 99999999;
    else if (PD + MD >= 5) game_board_2011144024[BOARD_SIZE_2011144024 * inner_y + inner_x].point += 99999999;
}
```