109000114 吳振群 mini project3 report

```
void read_board(std::ifstream& fin) {
    fin >> player;
    //me = player;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            fin >> board[i][j];
        }
    }
}

void read_valid_spots(std::ifstream& fin) {
    int n_valid_spots;
    fin >> n_valid_spots;
    int x, y;
    for (int i = 0; i < n_valid_spots; i++) {
        fin >> x >> y;
        next_valid_spots.push_back({ x, y });
    }
}
```

首先按照 play_random.cpp 的寫法讀入 main.cpp 的「現在狀態棋盤」，包含了我是執黑棋或白棋、現在的棋盤有幾顆黑棋幾顆白棋還是空格以及還有幾個點可以下，和這些點的確切座標，這些位置存入寫在全域範圍裡的 std::vector<Point> next_valid_spots，以便讓整個程式使用。

```
void write_valid_spot(std::ofstream& fout) {
    Node greatest;
    OthelloBoard now_board;
    now_board.board = board;
    now_board.cur_player = player;
    me = player;
    now_board.next_valid_spots = next_valid_spots;
    //greatest = minimax(now_board, 2, true);
    greatest = ab_pruning(now_board, 4, true, INT_MIN, INT_MAX);
    //std::cout << greatest.score << std::endl;
    fout << greatest.p.x << " " << greatest.p.y << std::endl;
    fout.flush();
}
```

我們的目的是回傳我們再來要下的那個位置座標給 main.cpp，我是以一個 Node

裡面的 point 的 x 以及 y 來表示回傳的值。這邊我記錄下了傳過來的現在狀態棋盤以做 AI 搜尋，很基本的先把一些值都令給我自己設在全域變數的 now_board，倒數第四行則開始用 alpha-beta pruning 進行演算找點。

```
Node ab_pruning(OthelloBoard now_board, int num, bool ismax, int a, int b) {
    Node ans;
    if (num == 0 || now_board.next_valid_spots.empty()) {
        Node leaf;
        leaf.p = Point(-5, -5);
        leaf.score = now_board.calc_value();
        return leaf;
    }

    if (ismax) {
        Node tmp;
        int max_ans = INT_MIN;
        for (Point n : now_board.next_valid_spots) {
            OthelloBoard tmp_board = now_board;
            if (tmp_board.put_disc(n)) {
                tmp = ab_pruning(tmp_board, num - 1, false, a, b);
                max_ans = std::max(max_ans, tmp.score);
                if (max_ans > a){
                    a = max_ans;
                    ans.score = max_ans;
                    ans.p = n;
                }
                if (a >= b) break;
            }
        }
    }
```

```
else {
    Node tmp_1;
    int min_ans = INT_MAX;

    for (Point p : now_board.next_valid_spots) {
        OthelloBoard tmp_board_1 = now_board;
        if (tmp_board_1.put_disc(p)) {
            tmp_1 = ab_pruning(tmp_board_1, num - 1, true, a, b);
            min_ans = std::min(min_ans, tmp_1.score);
            if (min_ans < b) {
                b = min_ans;
                ans.score = min_ans;
                ans.p = p;
            }
            if (b <= a) break;
        }
    }
}
return ans;
```

這裡用布林運算子 ismax 來代表我要找的是下一手帶給我價值最大的點。首先用 for 迴圈把每個可以下的點遍歷，對每個點進行 ab-pruning，用 max_ans 紀錄目前我能找到最大的值，並存入 Node ans 裡面的 score，也順便把此可以下的點記錄到 Node 裡，最後回傳 ans 便可以從裡面的 point 找到我最終要下得點。然而在基礎狀態也就是整棵樹最底部，我們要找的是 evaluation function，可以想像成我現在這個局面的分數或者勝率，分數越高則勝率越高。這個分數的實做可以透過下圖來呈現。

```
int calc_value() {
    if (cur_player != me) {
        //對手如果下一步可以下角落或來到這盤棋最後一手棋，則我這手棋價值會降低
        //對手如果可下的點小於等於四步，我下這手價值就提高
        int ans = 0;
        ans = now_score;
        if (done && winner == me) ans += 800;
        else if (done && winner == 3 - me) ans -= 800;
        if (next_valid_spots.size() == 0) ans += 100;
        for (long unsigned i = 0; i < 50; i++) {
            if (next_valid_spots.size() == i) {
                ans = ans - i*2;
                break;
            }
        }

        for (Point i : next_valid_spots) {
            for (Point dir : directions) {
                Point p = i + dir;
                if (!is_disc_at(p, get_next_player(cur_player)) || !is_spot_on_board(p))
                    continue;
                std::vector<Point> tmp_discs({ p });
                p = p + dir;
                while (is_spot_on_board(p) && get_disc(p) != EMPTY) {
                    if (is_disc_at(p, cur_player)) {
                        ans -= tmp_discs.size()*2;
                        break;
                    }
                    tmp_discs.push_back(p);
                    p = p + dir;
                }
            }
        }
        /* ... */
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++){
                if (board[i][j] == me) ans += node_value[i][j];
                else if (board[i][j] != me && board[i][j] != 0) ans -= node_value[i][j];
            }
        }

        if (next_valid_spots.size() == 1 && disc_count[EMPTY] == 1) ans -= 60;
        return ans;
    }
```

我把狀態分成兩個，一個是我準備要下棋，一個是對手準備要下棋。由於在 alpha-beta pruning 裡我把每個可以下的點利用 put_disk 函式來更新狀態，而這個函式轉換過程，目前的下棋者就被轉移到了對手那邊，也就是我下了棋之後 cur_player 會變成對手。上圖的函數裡我用 ans 紀錄最終會回傳的值，一開始令它等於 now_score，也就是在 flip_discs 函數裡面所寫的(最後一頁的圖)，目前狀態下我比對手多幾顆棋子或少幾顆，這是最基本的計算輸贏方式，而我也加上了一些更進階的判斷方法。首先式如果這盤棋結束了而且贏家是我，我的分數就加很多，

代表下了這個點後目前盤面就是我獲勝，這個值很大，最頂層 ab_prunig 的 Node 保證能夠接收到這個值。相反地，對手贏了我就減很多分。而對手目前如果連一手可以下的位置都沒有，那我就會勝率提升很多，因此也給他一個頗大的值，希望 ai 下這手棋。接著是一個 for 迴圈表示對手可以下的點越多，我的情況越不樂觀，可能會被翻盤的點就在其中，所以越多話我就減越多分。再來我也判斷了對手可下的點能夠吃掉我多少棋，依照多寡做等比例的扣分，因為我能被吃越多代表輸的機率越高。接著是用一個預先寫好的每棋盤上每點的價值二微陣列判斷我現在的價值高不高，下圖可以看見我自己如果下在了四個角落，分數會非常高，

```
const int SIZE = 8;

int node_value[SIZE][SIZE] = {
    66,  -6, 7, 4, 4, 7,  -6, 66,
    -6, -30, 3, 1, 1, 3, -30, -6,
    7, 3, 6, 2, 2, 6, 3, 7,
    4, 1, 2, 1, 1, 2, 1, 4,
    4, 1, 2, 1, 1, 2, 1, 4,
    7, 3, 6, 2, 2, 6, 3, 7,
    -6, -30, 3, 1, 1, 3, -30, -6,
    66, -6, 7, 4, 4, 7, -6, 66
};
```

因為在角落一定不會被吃掉，敵人沒辦法夾擊，也就是我的穩定性很高，算是我的實地，而不是暫時的分數。然而角落旁邊的三格都非常低分，因為如果自己下到了那些點，有可能被敵人下在角落夾擊我方棋子而被吃掉，並且敵人佔據最穩定的角落，一石二鳥。

接著是如果對手僅剩一步棋可以走，而且這步棋將結束本局，那我的分數會降很低，因為很有可能因為這一步棋就被翻盤。

而反之用我準備要下棋的狀態來思考，就跟上述提到的內容相反過來。

以上是本次 AI 的實做，雖然強度很弱，但也讓我更熟悉演算法的過程。

```
}
void flip_discs(Point center) {    //把中間的棋變成兩端棋子的顏色
    for (Point dir : directions) {
        // Move along the direction while testing.
        Point p = center + dir;
        if (!is_disc_at(p, get_next_player(cur_player)))
            continue;
        std::vector<Point> discs({ p });
        p = p + dir;
        while (is_spot_on_board(p) && get_disc(p) != EMPTY) {
            if (is_disc_at(p, cur_player)) {
                for (Point s : discs) {
                    set_disc(s, cur_player);    //88行
                }
                disc_count[cur_player] += discs.size();
                disc_count[get_next_player(cur_player)] -= discs.size();

                //add
                if (cur_player == me)
                    now_score = disc_count[cur_player]- disc_count[get_next_player(cur_player)];
                else
                    now_score = disc_count[get_next_player(cur_player)]- disc_count[cur_player];
                break;
            }
            discs.push_back(p);
            p = p + dir;
        }
    }
}
```