- Tree Search +2 point

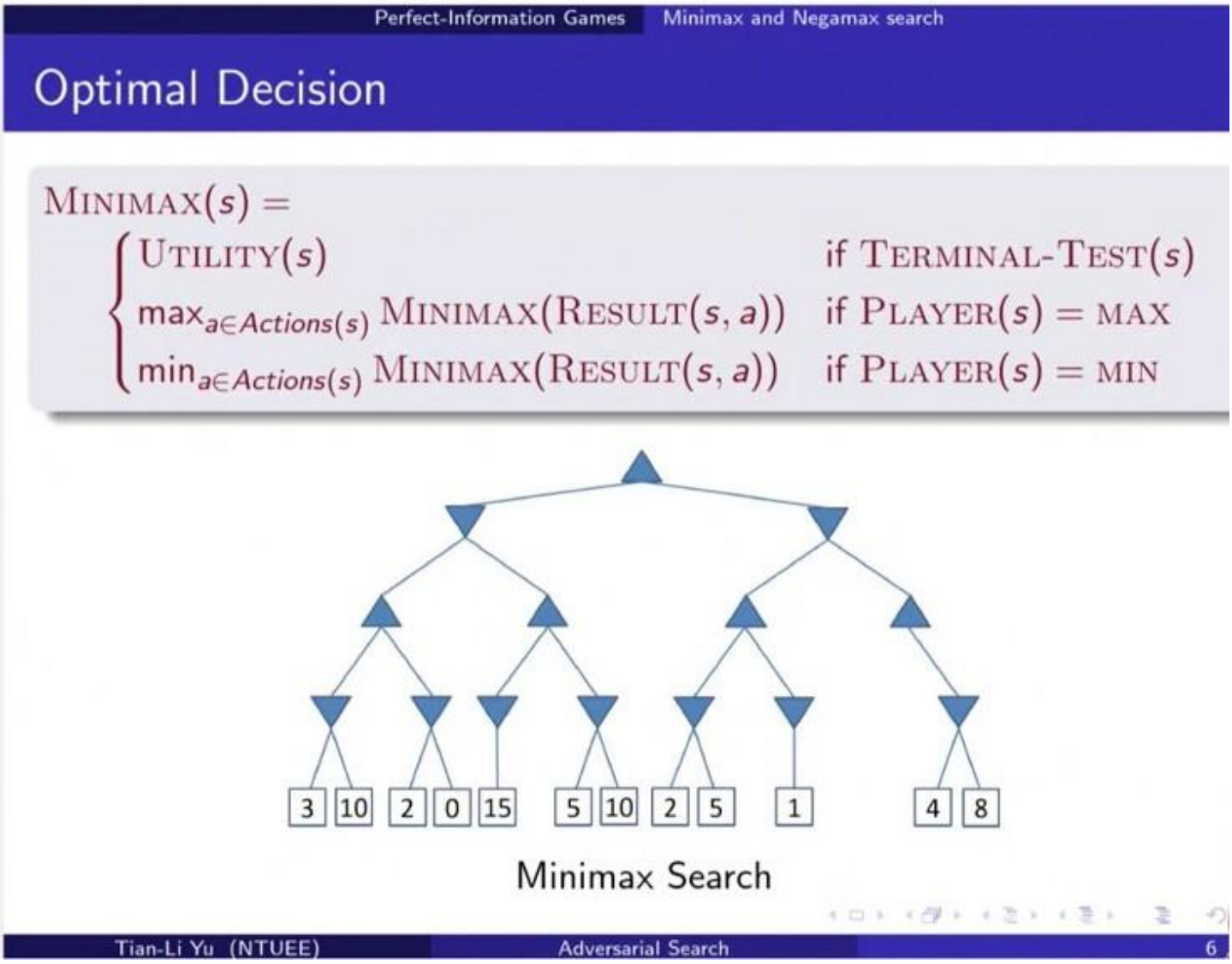
```cpp
void write_valid_spot(std::ofstream& fout) {
    int n_valid_spots = next_valid_spots.size();//總共有這麼多可以下的盤面

    OthelloBoard curState(brd, player);//先把目前main.exe所已經提供的盤面建構出來。題目跟我講我現在要下的是player(ex: 這裡以及以下的例子都用我想要下白色來說明)

    for (int terminal_depth = 0; /*terminal_depth < 6*/; terminal_depth++) {
      std::vector<int>State_Value;//紀錄每個valid下法的好壞程度
      State_Value.resize(n_valid_spots, _Val: 0);

      for (int i = 0; i < n_valid_spots; i++) {
        Point p = next_valid_spots[i];

        OthelloBoard temp = curState;
        temp.put_disc(p);//我已經把白色棋子下下去了，而目前temp裡的cur_player會自動改成黑色，
        State_Value[i] = temp.MiniMax(cur_depth: 0, terminal_depth);
      }
      //開始輸出最佳的放法
      //因為准許無限次輸出，所以我只要每找到一個比較好的放法，就輸出一次，即可
      //記得區分現在的player到底是BLACK還是WHITE
      if (player == curState.BLACK) {
        int now_MAX = INT_MIN;
        for (int i = 0; i < n_valid_spots; i++)if (State_Value[i] > now_MAX) {
          now_MAX = State_Value[i];

          fout << next_valid_spots[i].x << " " << next_valid_spots[i].y << std::endl;
          fout.flush();
        }
      }
      else if (player == curState.WHITE) {
        //因為我下的是白色的，所以State_Value裡面儲存的是我每個下法對「黑色」來講有多糟，所以我只要取對黑色來講最糟的(數值最小的)，就是對我白色最好的
        int now_min = INT_MAX;
        for (int i = 0; i < n_valid_spots; i++)if (State_Value[i] < now_min) {
          now_min = State_Value[i];

          fout << next_valid_spots[i].x << " " << next_valid_spots[i].y << std::endl;
          fout.flush();
        }
      }
```

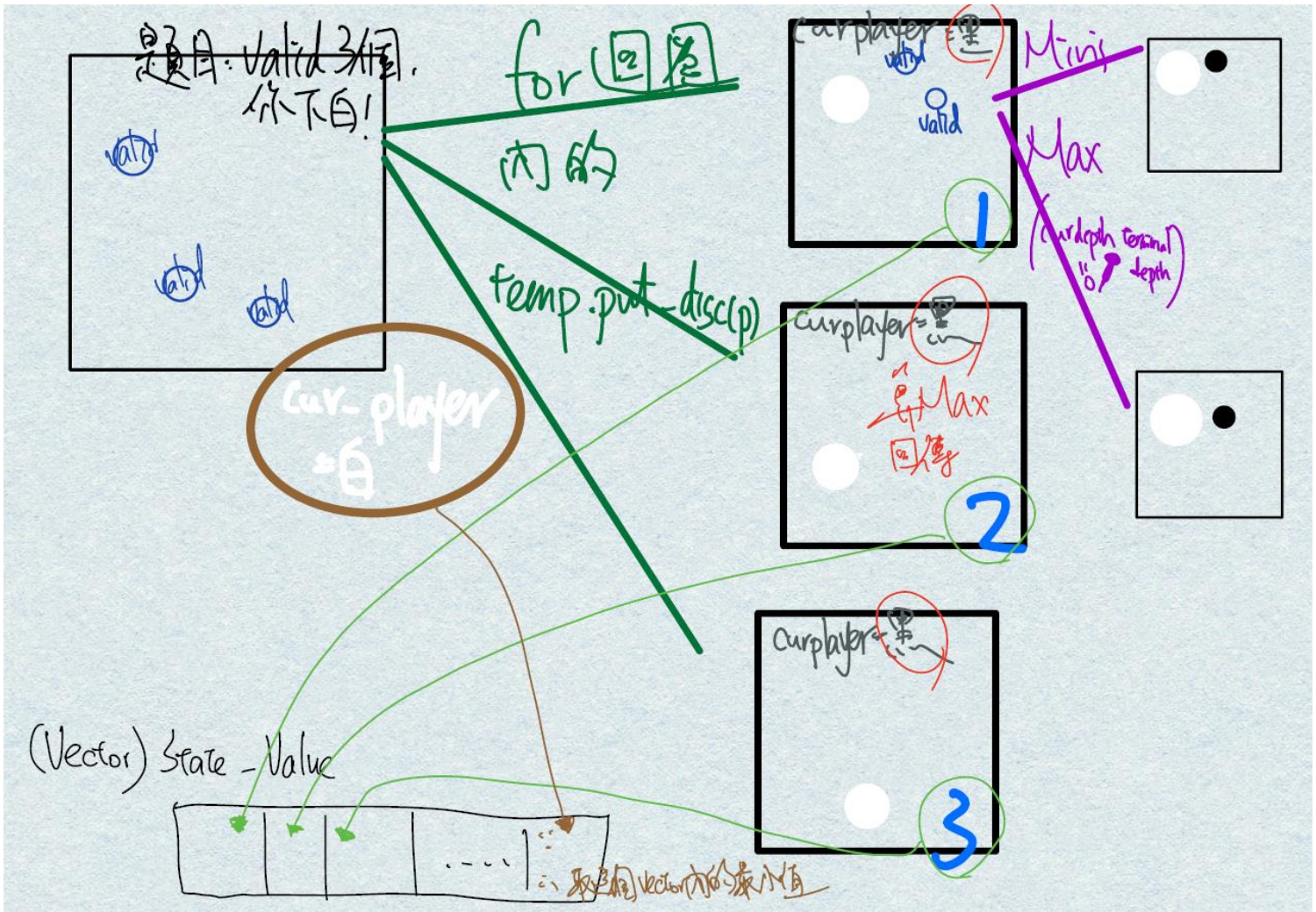
■

```cpp
int OthelloBoard::MiniMax(int cur_depth, int terminal_depth) {
	//先計算目前的valid的格子有哪些
	//不管是if(cur_depth == terminal_depth)還是後面兩個，都會用到
	next_valid_spots = get_valid_spots();

	if (cur_depth == terminal_depth) {
		int score = 0;
		if (cur_player == BLACK) {
			for (int i = 0; i < SIZE; i++)for (int j = 0; j < SIZE; j++) {
				Point p(i, j);
				if (p == Corners[0] || p == Corners[1] || p == Corners[2] || p == Corners[3]) {
					if (board[p.x][p.y] == BLACK)score += Good_Corner;
					else if (board[p.x][p.y] == WHITE)score += Bad_Corner;
				}
				else if (p.x == 0 || p.x == SIZE - 1) {
					if (board[p.x][0] == BLACK && board[p.x][SIZE - 2] == BLACK)score += Four_Sides;
					else if (board[p.x][0] == WHITE && board[p.x][SIZE - 2] == WHITE)score += Bad_Four_Sides;
				}
				else if (p.y == 0 || p.y == SIZE - 1) {
					if (board[0][p.y] == BLACK && board[SIZE - 2][p.y] == BLACK)score += Four_Sides;
					else if (board[0][p.y] == WHITE && board[SIZE - 2][p.y] == WHITE)score += Bad_Four_Sides;
				}

				if (disc_count[WHITE] + disc_count[BLACK] >= Latter_Quarter_Threshold) {
					if (board[p.x][p.y] == BLACK)score += Ordinary_Disc;
					else if (board[p.x][p.y] == WHITE)score += Enemy_Ordinary_Disc;
				}
			}
		}
		else if (cur_player == WHITE) {
			for (int i = 0; i < SIZE; i++)for (int j = 0; j < SIZE; j++) {
				Point p(i, j);
				if (p == Corners[0] || p == Corners[1] || p == Corners[2] || p == Corners[3]) {
					if (board[p.x][p.y] == WHITE)score += Good_Corner;
					else if (board[p.x][p.y] == BLACK)score += Bad_Corner;
				}
				else if (p.x == 0 || p.x == SIZE - 1) {
					if (board[p.x][0] == WHITE && board[p.x][SIZE - 2] == WHITE)score += Four_Sides;
					else if (board[p.x][0] == BLACK && board[p.x][SIZE - 2] == BLACK)score += Bad_Four_Sides;
				}
				else if (p.y == 0 || p.y == SIZE - 1) {
					if (board[0][p.y] == WHITE && board[SIZE - 2][p.y] == WHITE)score += Four_Sides;
					else if (board[0][p.y] == BLACK && board[SIZE - 2][p.y] == BLACK)score += Bad_Four_Sides;
				}

				if (disc_count[WHITE] + disc_count[BLACK] >= Latter_Quarter_Threshold) {
					if (board[p.x][p.y] == WHITE)score += Ordinary_Disc;
					else if (board[p.x][p.y] == BLACK)score += Enemy_Ordinary_Disc;
				}
			}
		}

		if (disc_count[WHITE] + disc_count[BLACK] < Latter_Quarter_Threshold) score += next_valid_spots.size()*Mobility;

		//注意return的時候，到底是誰在下棋?黑色?白色?
		return (cur_player == BLACK) ? score : -score;
	}

	if (this->cur_player == BLACK/*MAX*/) {
		//max of(a belongs to Action(cur state), MINIMAX(result of (cur state, a)))

		//針對現有的盤面，去依序放上有所valid的格子，得到所有新的盤面。把所有新的盤面取MINIMAX，找最大的MINIMAX值回傳
		int cur_MAX = INT_MIN;
		for (int i = 0; i < next_valid_spots.size(); i++) {
			OthelloBoard temp = *this;
			temp.put_disc(next_valid_spots[i]);
```

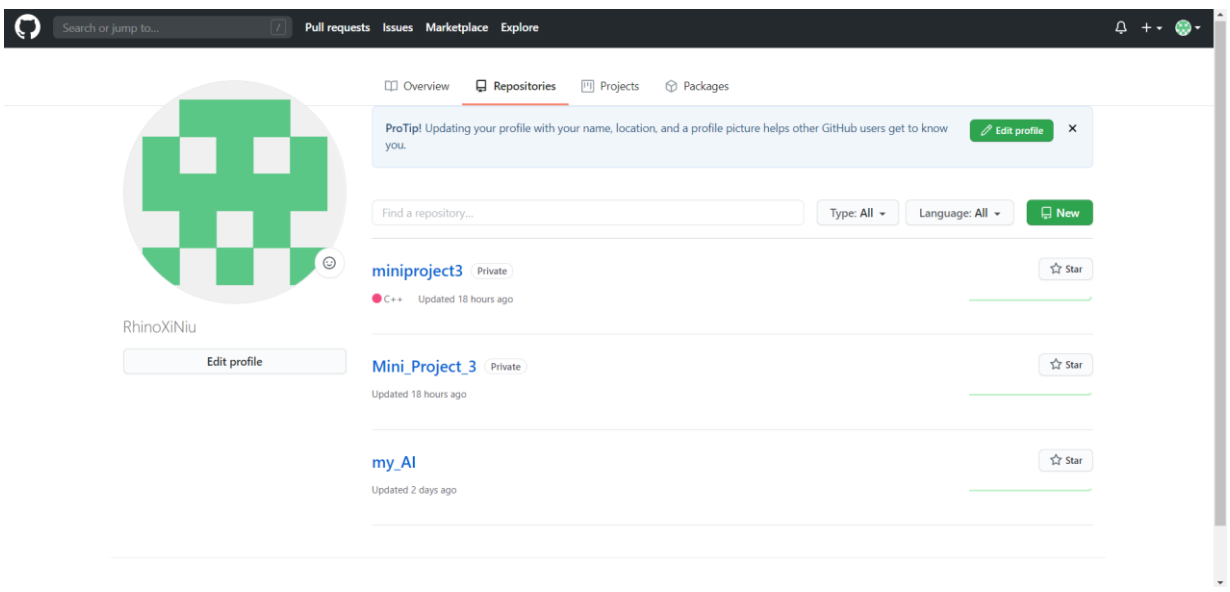
```
                int tmp = temp.MiniMax(cur_depth + 1, terminal_depth);

                if (tmp > cur_MAX)cur_MAX = tmp;
            }
            return cur_MAX;
        }
        else if (this->cur_player == WHITE) {
            //min of(a belongs to Action(cur state), MINIMAX(result of (cur state, a)))

            int cur_min = INT_MAX;
            for (int i = 0; i < next_valid_spots.size(); i++) {
                OthelloBoard temp = *this;
                temp.put_disc(next_valid_spots[i]);
                int tmp = temp.MiniMax(cur_depth + 1, terminal_depth);

                if (tmp < cur_min)cur_min = tmp;
            }
            return cur_min;
        }
```

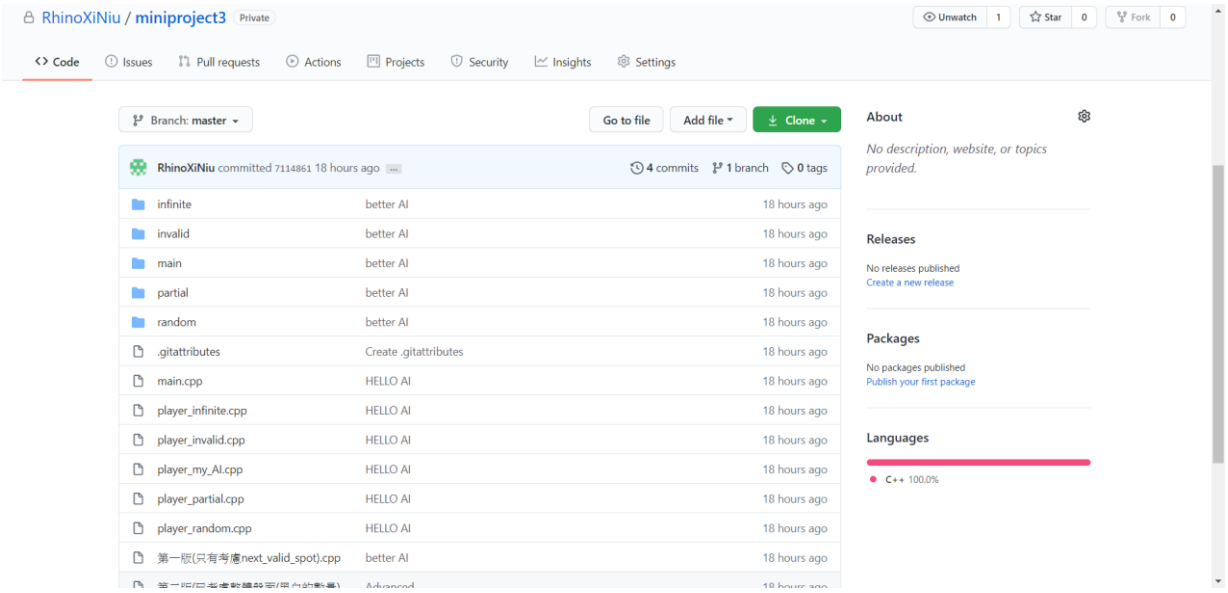
- }





- State Value Function Design +1 point
  - 在上方文字函式的開頭處
- Alpha-beta Pruning +1 point
  - 沒有實作
- Version Control (Bonus) +1 point





Struct Point、void read_board、void read_valid_spots、int main。

```cpp
struct Point {
    int x, y;
    Point() : Point(0, 0) {}
    Point(float x, float y) : x(x), y(y) {}
    bool operator==(const Point& rhs) const {
        return x == rhs.x && y == rhs.y;
    }
    bool operator!=(const Point& rhs) const {
        return !operator==(rhs);
    }
    Point operator+(const Point& rhs) const {
        return Point(x + rhs.x, y + rhs.y);
    }
    Point operator-(const Point& rhs) const {
        return Point(x - rhs.x, y - rhs.y);
    }
};
```

```cpp
int player;
const int SIZE = 8;
std::array<std::array<int, SIZE>, SIZE> brd;
std::vector<Point> next_valid_spots;

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

```

Class OthelloBoard: int get_next_player、bool is_spot_on_board。

```cpp
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:(float)x, y:(float)y });
    }
}

void write_valid_spot(std::ofstream& fout);

int main(int, char** argv) {
    std::ifstream fin(argv[1]);
    std::ofstream fout(argv[2]);
    read_board(fin);
    read_valid_spots(fin);
    write_valid_spot(fout);
    fin.close();
    fout.close();
    return 0;
}

```

```cpp
class OthelloBoard {
public:
    enum SPOT_STATE {
        EMPTY = 0,
        BLACK = 1,
        WHITE = 2
    };
    static const int SIZE = 8;
    const std::array<Point, 8> directions{ _Elems:{
        Point(-1, -1), Point(-1, 0), Point(-1, 1),
        Point(0, -1), /*[0, 0], */Point(0, 1),
        Point(1, -1), Point(1, 0), Point(1, 1)
    } };
    std::array<std::array<int, SIZE>, SIZE> board;
    std::vector<Point> next_valid_spots;
    std::array<int, 3> disc_count;
    int cur_player;
    bool done;
    int winner;
private:
    //透過傳入目前的player編號，抓取下一個player編號
    int get_next_player(int player) const {
        return 3 - player;
    }
    //檢查是否超出螢幕有效範圍
    bool is_spot_on_board(Point p) const {
        return 0 <= p.x && p.x < SIZE && 0 <= p.y && p.y < SIZE;
    }
```

Int get_disc、void set_disc、bool is_disc_at。

```
93    //把某一個特定座標所具有的disc編號傳回。可能的值: 0-EMPTY、1-BLACK、2-WHITE
94    int get_disc(Point p) const {
95        return board[p.x][p.y];
96    }
97    //把某一個特定座標修改為特定的disc編號。可能的值: 0-EMPTY、1-BLACK、2-WHITE
98    //不必用到，因為後面會再外包給put_disc執行
99    void set_disc(Point p, int disc) {
100       board[p.x][p.y] = disc;
101   }
102   //輸入位置座標和想要的disc編號，檢查是否確實在那個位置有著想要的disc
103   bool is_disc_at(Point p, int disc) const {
104       if (!is_spot_on_board(p))
105           return false;
106       if (get_disc(p) != disc)
107           return false;
108       return true;
109   }
```

Bool is_spot_valid。

```
110   //在想要把一個棋子下下去一個位置之前，所做的檢查。做了以下幾件事情:
111   //(1)先檢查該點是否已經不可下下去
112   //(2)往八個方向檢查，看看是否有同樣顏色的東西在做末端有包圍住，以供有效的下法。
113   bool is_spot_valid(Point center) const {
114       if (get_disc(center) != EMPTY)
115           return false;
116       for (Point dir : directions) {
117           // Move along the direction while testing.
118           Point p = center + dir;
119           if (!is_disc_at(p, disc:get_next_player(cur_player)))
120               continue;
121           p = p + dir;
122           while (is_spot_on_board(p) && get_disc(p) != EMPTY) {
123               if (is_disc_at(p, cur_player))
124                   return true;
125               p = p + dir;
126           }
127       }
128       return false;
129   }
```

Void flip_discs。

```
130   //下下一個棋子之後，往八個方位進行檢查，看看如果這個方位確實是可以被flip的話，先用list記下來需要flip的座標，之後一個方位檢查後統一處理。
131   //之後，disc_count相關的資訊一併調整。
132   //!!!不必使用使用此function，因為會外包給put_disc使用
133   void flip_discs(Point center) {
134       for (Point dir : directions) {
135           // Move along the direction while testing.
136           Point p = center + dir;
137           if (!is_disc_at(p, disc:get_next_player(cur_player)))
138               continue;
139           std::vector<Point> discs(_Ilist:{ p });
140           p = p + dir;
141           while (is_spot_on_board(p) && get_disc(p) != EMPTY) {
142               if (is_disc_at(p, cur_player)) {
143                   for (Point s : discs) {
144                       set_disc(s, cur_player);
145                   }
146                   disc_count[cur_player] += discs.size();
147                   disc_count[get_next_player(cur_player)] -= discs.size();
148                   break;
149               }
150               discs.push_back(p);
151               p = p + dir;
152           }
153       }
154   }
```

Constructor * 3。

```
public:
    //Constructor
    OthelloBoard() {
        reset();
    }
    //Copy Constructor
    OthelloBoard(const OthelloBoard & rhs) :cur_player(rhs.cur_player), done(rhs.done), winner(rhs.winner) {
        for (auto i = 0; i < rhs.SIZE; i++) for (auto j = 0; j < rhs.SIZE; j++) {
            board[i][j] = rhs.board[i][j];
        }

        for (auto i = 0; i < 3; i++)disc_count[i] = rhs.disc_count[i];
    }
    //從現有的2D盤面建構Board
    OthelloBoard(std::array<std::array<int, SIZE>, SIZE> brd, int player_idx) : done(false), winner(-1), cur_player(player_idx) {
        disc_count[EMPTY] = disc_count[BLACK] = disc_count[WHITE] = 0;

        for (int i = 0; i < SIZE; i++)for (int j = 0; j < SIZE; j++) {
            board[i][j] = brd[i][j];

            disc_count[board[i][j]]++;//是什麼棋子，就把那個棋子的count做++。
        }

    }
```

Void reset、std::vector<Point> get_valid_spots。

```
    //把盤面還原到最初始的棋盤
    void reset() {
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                board[i][j] = EMPTY;
            }
        }
        board[3][4] = board[4][3] = BLACK;
        board[3][3] = board[4][4] = WHITE;
        cur_player = BLACK;
        disc_count[EMPTY] = 8 * 8 - 4;
        disc_count[BLACK] = 2;
        disc_count[WHITE] = 2;
        next_valid_spots = get_valid_spots();
        done = false;
        winner = -1;
    }
```

```
    //在一個有效位置被下下去之後，更新所有新的valid spots。
    std::vector<Point> get_valid_spots() const {
        std::vector<Point> valid_spots;
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                Point p = Point(x:i, y:j);
                if (board[i][j] != EMPTY)
                    continue;
                if (is_spot_valid(p))
                    valid_spots.push_back(p);
            }
        }
        return valid_spots;
    }
```

Bool put_disc。

```
//如果下的位置是錯的，那麼贏家是誰就出爐了。否則的話就是更新valid spots。而如果現在的valid spots是0個，
//而再下一輪還是0個的話，代表兩個玩家都已經沒有路可以走了，意味著遊戲已經結束。
bool put_disc(Point p) {
    if (!is_spot_valid(p)) {
        winner = get_next_player(cur_player);
        done = true;
        return false;
    }
    set_disc(p, cur_player);
    disc_count[cur_player]++;
    disc_count[EMPTY]--;
    flip_discs(p);
    // Give control to the other player.
    cur_player = get_next_player(cur_player);
    next_valid_spots = get_valid_spots();
    // Check Win
    if (next_valid_spots.size() == 0) {
        cur_player = get_next_player(cur_player);
        next_valid_spots = get_valid_spots();
        if (next_valid_spots.size() == 0) {
            // Game ends
            done = true;
            int white_discs = disc_count[WHITE];
            int black_discs = disc_count[BLACK];
            if (white_discs == black_discs) winner = EMPTY;
            else if (black_discs > white_discs) winner = BLACK;
            else winner = WHITE;
        }
    }
    return true;
```

參數設定。

```
const std::array<Point, 4> Corners{ ._Elems:{
    Point(x:0, y:0), Point(x:0, y:SIZE - 1),
    Point(x:SIZE - 1, y:0),Point(x:SIZE - 1, y:SIZE - 1)
    //左上、右上
    //左下、右下
} };

const std::array<Point, 4> Corners_Inner{ ._Elems:{
    Point(x:Corners[0].x + 1, y:Corners[0].y + 1), Point(x:Corners[1].x + 1, y:Corners[1].y - 1),
    Point(x:Corners[2].x - 1, y:Corners[2].y + 1), Point(x:Corners[3].x - 1, y:Corners[3].y - 1)
} };

const std::array<Point, 8> Corners_two_sides{ ._Elems:{
    Point(x:Corners[0].x + 1, Corners[0].y),
    Point(Corners[0].x , y:Corners[0].y + 1),
    Point(x:Corners[1].x + 1, Corners[1].y),
    Point(Corners[1].x , y:Corners[1].y - 1),
    Point(x:Corners[2].x - 1, Corners[2].y),
    Point(Corners[2].x , y:Corners[2].y + 1),
    Point(x:Corners[3].x - 1, Corners[3].y),
    Point(Corners[3].x , y:Corners[3].y - 1)
} };
```