School of Data and Computer Science, Sun Yat-sen University

# E3 Othello Game ($\alpha - \beta$ pruning)

18340013 Conghao Chen

September 17, 2020

## Contents

**1 Othello** **2**

**2 Tasks** **2**

**3 Codes** **3**

**4 Results** **20**

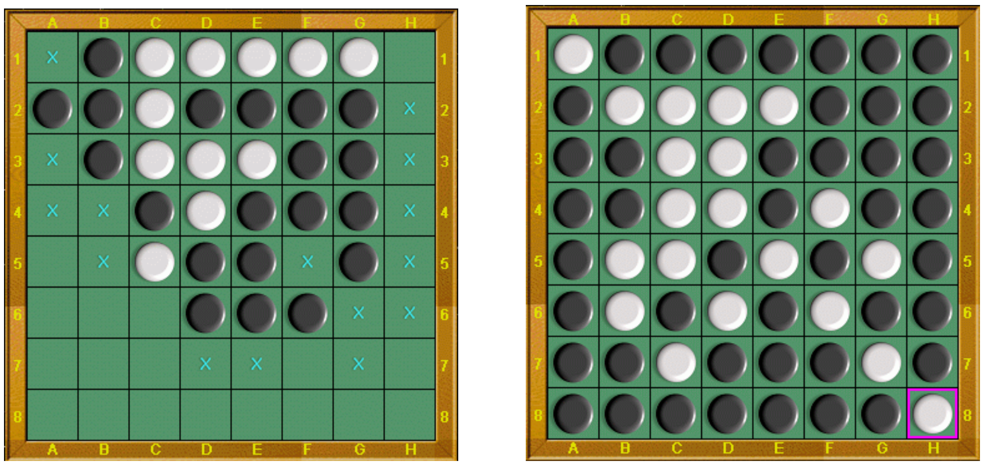


Figure 1: Othello Game

## 1 Othello

Othello (or Reversi) is a strategy board game for two players, played on an $8 \times 8$ uncheckered board. There are sixty-four identical game pieces called disks (often spelled ”discs”), which are light on one side and dark on the other. Please see figure 1.

Players take turns placing disks on the board with their assigned color facing up. During a play, any disks of the opponent’s color that are in a straight line and bounded by the disk just placed and another disk of the current player’s color are turned over to the current player’s color.

The object of the game is to have the majority of disks turned to display your color when the last playable empty square is filled.

You can refer to `http://www.tothello.com/html/guideline_of_reversed_othello.html` for more information of guideline, meanwhile, you can download the software to have a try from `http://www.tothello.com/html/download.html`. The game installer `tothello_trial_setup.exe` can also be found in the current folder.

## 2 Tasks

1. In order to reduce the complexity of the game, we think the board is $6 \times 6$.

2. There are several evaluation functions that involve many aspects, you can turn to `http://blog.sina.com.cn/s/blog_53ebdba00100cpy2.html` for help. In order to reduce the difficulty of the task, I have gaven you some hints of evaluation function in the file `Heuristic Function for Reversi (Othello).cpp`.

3. Please choose an appropriate evaluation function and use min-max and $\alpha - \beta$ prunning to implement the Othello game. The framework file you can refer to is `Othello.cpp`. Of course, I wish your program can beat the computer.

4. Write the related codes and take a screenshot of the running results in the file named E03_YourNumber.pdf, and send it to ai_2020@foxmail.com.

## 3 Codes

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int const MAX = 65534;
int  depth = 12;          //最大搜索深度   （可调节）
//基本元素   棋子，颜色，数字变量
enum Option
{
        WHITE = -1, SPACE, BLACK          //是否能落子   //黑子
};
struct Do
{
        pair<int , int > pos;
        int score;
};
struct WinNum
{       enum Option color;
        int stable;
                    // 此次落子赢棋个数
};

//主要功能     棋盘及关于棋子的所有操作，功能
struct Othello

{
        WinNum cell[6][6];
                                                  //定义棋盘中有6*6个格子
        int whiteNum;
                                                          //白棋数目
        int blackNum;
                                                          //黑棋数目
        void Create(Othello *board);                                //初始化棋盘
        void Copy(Othello *boardDest, const  Othello *boardSource);    //复制棋盘
        void Show(Othello *board);                                      //显示棋盘
        int Rule(Othello *board, enum Option player);    //判断落子是否符合规则
        int Action(Othello *board, Do *choice, enum Option player);    //落子,并修改棋盘
        void Stable(Othello *board);                                //计算赢棋个数
        int Judge(Othello *board, enum Option player);          //计算本次落子分数
};//主要功能


//我用的剪枝算法
Do * Find1(Othello *board, enum Option player, int step, int min, int max, Do *choice)
/*  step: 极大极小树的深度，从大往小递减  */
{
        int i, j, k, num;
        Do *allChoices;
        choice->score = -MAX;
        choice->pos.first = -1;
        choice->pos.second = -1;

        num = board->Rule(board, player);
    /*  找出player可以落子的数量，对应于图像界面里面的‘+’的个数  */
        if (num == 0)    /* 无处落子 */
        {
                if (board->Rule(board, (enum Option) - player))    /* 对方可以落子,让对方下.*/
                {
                        Othello tempBoard;
                        Do nextChoice;
                        Do *pNextChoice = &nextChoice;
                        board->Copy(&tempBoard, board);
                        pNextChoice = Find1(&tempBoard, (enum Option) - player, step - 1, -max, -min, pNextChoice);
                        choice->score = -pNextChoice->score;
```

```
                    choice->pos.first = -1;
                    choice->pos.second = -1;
                    return choice;
                }
                else    /* 对方也无处落子,游戏结束. */
                {
                    int value = WHITE*(board->whiteNum) + BLACK*(board->blackNum);
                    if (player*value>0)
                    {
                        choice->score = MAX - 1;
                    }
                    else if (player*value<0)
                    {
                        choice->score = -MAX + 1;
                    }
                    else
                    {
                        choice->score = 0;
                    }
                    return choice;
                }
        }
        if (step <= 0)    /* 已经考虑到step步,直接返回得分 */
        {
                choice->score = board->Judge(board, player);
                return choice;
        }

        /* 新建一个do*类型的数组，其中num即为玩家可落子的数量 */
        allChoices = (Do *)malloc(sizeof(Do)*num);


/*
下面三个两重for循环其实就是分区域寻找可落子的位置，代码num = board->Rule(board, player)
    只返回了可落子的数量，并没有返回可落子的位置，因此需要重新遍历整个棋盘去寻找可落子的
    位置。下面三个for循环分别按照最外一圈、最中间的四个位置、靠里的一圈这三个顺序来寻找
    可落子的位置，如下图所示(数字表示寻找的顺序)
                1 1 1 1 1 1
                1 3 3 3 3 1
                1 3 2 2 3 1
                1 3 2 2 3 1
```

```
            1 3 3 3 3 1
            1 1 1 1 1 1
*/
        k = 0;
        for (i = 0; i<6; i++)   /* 在最外圈寻找可落子位置 */
        {
            for (j = 0; j<6; j++)
            {
                if (i == 0 || i == 5 || j == 0 || j == 5)
                {
                    /* 可落子的位置需要满足两个条件：1、该位置上没有棋子，2
                      、如果把棋子放在这个位置上可以吃掉对方的棋子(可以夹
                      住对方的棋子)。stable记录的是可以吃掉对方棋子的数
                      量，所以stable>0符合条件2*/
                    if (board->cell[i][j].color == SPACE && board->cell[i][j
                      ].stable)
                    {
                        allChoices[k].score = -MAX;
                        allChoices[k].pos.first = i;
                        allChoices[k].pos.second = j;
                        k++;
                    }
                }
            }
        }

        for (i = 0; i<6; i++)  // 分析同上
        {
            for (j = 0; j<6; j++)
            {
                if ((i == 2 || i == 3 || j == 2 || j == 3) && (i >= 2 && i <= 3
                    && j >= 2 && j <= 3))
                {
                    if (board->cell[i][j].color == SPACE && board->cell[i][j
                      ].stable)
                    {
                        allChoices[k].score = -MAX;
                        allChoices[k].pos.first = i;
                        allChoices[k].pos.second = j;
                        k++;
                    }
                }
            }
        }

        for (i = 0; i<6; i++)  // 分析同上
        {
            for (j = 0; j<6; j++)
            {
                if ((i == 1 || i == 4 || j == 1 || j == 4) && (i >= 1 && i <= 4
                    && j >= 1 && j <= 4))
                {
                    if (board->cell[i][j].color == SPACE && board->cell[i][j
                      ].stable)
                    {
                        allChoices[k].score = -MAX;
                        allChoices[k].pos.first = i;
                        allChoices[k].pos.second = j;
                        k++;
                    }
                }
            }
        }

        for (k = 0; k<num; k++)   /* 尝试在之前得到的num个可落子位置进行落子 */
        {
            Othello tempBoard;
            Do thisChoice, nextChoice;
            Do *pNextChoice = &nextChoice;
            thisChoice = allChoices[k];
            board->Copy(&tempBoard, board);  // 为了不影响当前棋盘，需要复制一份作为
                虚拟棋盘
            board->Action(&tempBoard, &thisChoice, player);  // 在虚拟棋盘上落子
            pNextChoice = Find1(&tempBoard, (enum Option) - player, step - 1, -max,
                -min, pNextChoice); // 递归调用 α-β 剪枝，得到对手的落子评分
            thisChoice.score = -pNextChoice->score;

            /* 使用Negamax算法代替minmax算法，实现 α-β 剪枝*/
            // 其中，max 取上一层min的相反数，min取当前选择的score。
            // 对每一层，我方行棋选择我方获益分数最大的，对手行棋选择我方获益分数最
                小的；
            // 因此，实际上只需要将每一层的max min调换并取反即可；
                // 故假设根节点为第0层，beta层的数值为负。
                // 剪枝条件：beta <= alpha，即score >= max。

                if (player == WHITE) {
                        int alpha = -max, beta = -min;
                        if (thisChoice.score > -beta) {
                                beta = -thisChoice.score;
                                choice->score = thisChoice.score;
                                choice->pos.first = thisChoice.pos.first;
                                choice->pos.second = thisChoice.pos.second;
                                min = -beta;
                                if (beta <= alpha) break;
                        }
                }
                else if(player == BLACK) {
                        int alpha = min, beta = max;
                        if (thisChoice.score > alpha) {
                                alpha = thisChoice.score;
                                choice->score = thisChoice.score;
                                choice->pos.first = thisChoice.pos.first;
                                choice->pos.second = thisChoice.pos.second;
                                min = alpha;
                                if (beta <= alpha) break;
                        }
                }
        }
        free(allChoices);
        return choice;
}

int main()
{
        Othello board;
        Othello *pBoard = &board;
        enum Option player , present ;
        Do choice;
        Do *pChoice = &choice;
        int  num , result = 0;
        char restart = ' ';

start:
    player = SPACE;
    present = BLACK;
    num = 4;
    restart = ' ';

    cout << ">>>人机对战开始： \n";
        while (player != WHITE && player != BLACK)
        {
            cout << ">>>请选择执黑棋( ),或执白棋( )：输入1为黑棋，-1为白棋" << endl;
            scanf("%d", &player);
            cout << ">>>黑棋行动： \n";
            if (player != WHITE && player != BLACK)
            {
                cout << "输入不符合规范，请重新输入\n";
            }
        }

        board.Create(pBoard);
        while (num<36)                    // 棋盘上未下满36子
        {
            char *Player = "";
            if (present == BLACK)
            {
                Player = "黑棋( )";
            }
            else if (present == WHITE)
            {
                Player = "白棋( )";
            }

            if (board.Rule(pBoard, present) == 0)                    //未下满并且无子可下
            {
                if (board.Rule(pBoard, (enum Option) - present) == 0)
                {
                    break;
                }

                cout << Player << "GAME OVER! \n";
```

```
                        }
                        else
                        {
                                int i, j;
                                board.Show(pBoard);

                                if (present == player)      //我的AI下棋
                                {
                                        cout << Player << ".........................";

                                        pChoice = Find1(pBoard, present, depth, -MAX,
                                            MAX, pChoice);
                                        i = pChoice->pos.first;
                                        j = pChoice->pos.second;
                                        system("cls");
                                        cout << ">>>我的AI本手棋得分为       " << pChoice
                                            ->score << endl;
                                        board.Action(pBoard, pChoice, present);
                                num++;
                                cout << Player << ">>>我的AI于" << i + 1 << "," << j +
                                    1<<"落子，该你了！";
                                }
                                else                            //电脑的AI下棋
                                {
                                        cout << Player << ".........................";

                                        pChoice = Find1(pBoard, present, depth, -MAX,
                                            MAX, pChoice);
                                        i = pChoice->pos.first;
                                        j = pChoice->pos.second;
                                        system("cls");
                                        cout << ">>>电脑的AI本手棋得分为       " <<
                                            pChoice->score << endl;
                                board.Action(pBoard, pChoice, present);
                                num++;
                                cout << Player << ">>>电脑的AI于" << i + 1 << "," << j +
                                    1<<"落子，该你了！";
                                }
                        }
                        present = (enum Option) - present;     //交换执棋者
                }
```

```
        board.Show(pBoard);
        result = pBoard->whiteNum - pBoard->blackNum;
        if (result>0)
        {
            cout << "\n-------白棋( )胜-------\n";
        }
        else if (result<0)
        {
            cout << "\n-------黑棋( )胜-------\n";
        }
        else
        {
            cout << "\n----------平局----------\n";
        }

        cout << "\n ----------GAME OVER!----------\n";
        cout << "\n";

        while (restart != 'Y' && restart != 'N')
        {
            cout <<"|--------------------------|\n";
            cout <<"|                          | \n";
            cout <<"|                          |   \n";
            cout <<"|>>>>>>>>>>>>>>>Again?(Y,N)<<<<<<<<<<<<<<<|\n";
            cout <<"|                          | \n";
            cout <<"|                          |  \n";
            cout <<"|--------------------------|\n";
            cout << "                            \n";
            cout << "                            \n";
            cout << "                            \n";
            cout << " ------            ------    \n";
            cout << " |  YES |          |  NO  |   \n";
            cout << " ------            ------    \n";

            cin >> restart;
            if (restart == 'Y')
            {
                goto start;
            }
        }


        return 0;
}


void Othello::Create(Othello *board)
{
        int i, j;
        board->whiteNum = 2;
        board->blackNum = 2;
        for (i = 0; i<6; i++)
        {
                for (j = 0; j<6; j++)
                {
                        board->cell[i][j].color = SPACE;
                        board->cell[i][j].stable = 0;
                }
        }
        board->cell[2][2].color = board->cell[3][3].color = WHITE;
        board->cell[2][3].color = board->cell[3][2].color = BLACK;
}


void Othello::Copy(Othello *Fake, const  Othello *Source)
{
        int i, j;
        Fake->whiteNum = Source->whiteNum;
        Fake->blackNum = Source->blackNum;
        for (i = 0; i<6; i++)
        {
                for (j = 0; j<6; j++)
                {
                        Fake->cell[i][j].color = Source->cell[i][j].color;
                        Fake->cell[i][j].stable = Source->cell[i][j].stable;
                }
        }
}

void Othello::Show(Othello *board)
{
        int i, j;
```

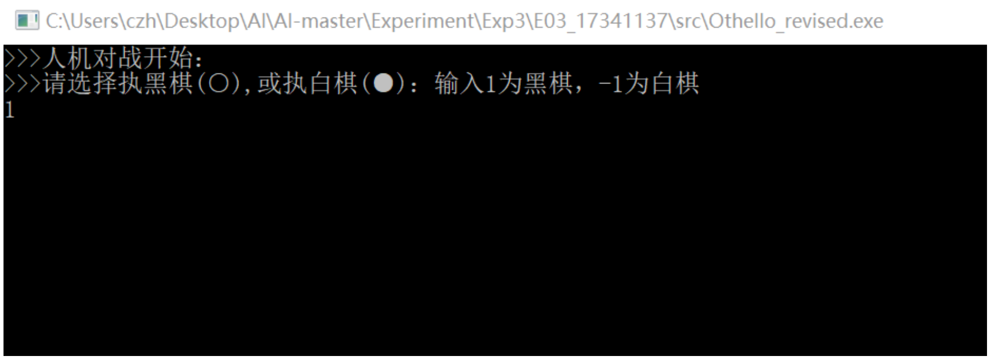
```
    cout << "\n  ";
    for (i = 0; i<6; i++)
    {
        cout << "   " << i + 1;
    }
    cout << "\n                \n";
    for (i = 0; i<6; i++)
    {
        cout << i + 1 << "-- ";
        for (j = 0; j<6; j++)
        {
            switch (board->cell[i][j].color)
            {
            case BLACK:
                cout << "  ";
                break;
            case WHITE:
                cout << "  ";
                break;
            case SPACE:
                if (board->cell[i][j].stable)
                {
                    cout << " + ";
                }
                else
                {
                    cout << "   ";
                }
                break;
            default:    /* 棋子颜色错误 */
                cout << "*  ";
            }
        }
        cout << "\n                \n";
    }

    cout << ">>>白棋( )个数为:" << board->whiteNum << "          ";
    cout << ">>>黑棋( )个数为:" << board->blackNum << endl << endl << endl;
}

int Othello::Rule(Othello *board, enum Option player)
```

```c
{
    int i, j;
    unsigned num = 0;
    for (i = 0; i<6; i++)
    {
        for (j = 0; j<6; j++)
        {
            if (board->cell[i][j].color == SPACE)
            {
                int x, y;
                board->cell[i][j].stable = 0;
                for (x = -1; x <= 1; x++)
                {
                    for (y = -1; y <= 1; y++)
                    {
                        if (x || y)    /* 8个方向 */
                        {
                            int i2, j2;
                            unsigned num2 = 0;
                            for (i2 = i + x, j2 = j + y; i2 >= 0 && i2 <= 5 && j2 >= 0 && j2 <= 5; i2 += x, j2 += y)
                            {
                                if (board->cell[i2][j2].color == (enum Option) - player)
                                {
                                    num2++;
                                }
                                else if (board->cell[i2][j2].color == player)
                                {
                                    board->cell[i][j].stable += player*num2;
                                    break;
                                }
                                else if (board->cell[i2][j2].color == SPACE
                                      )
                                {
                                    break;
                                }
                            }
                        }
                    }
                }

                if (board->cell[i][j].stable)
                {
                    num++;
                }
            }
        }
    }
    return num;
}


int Othello::Action(Othello *board, Do *choice, enum Option player)
{
    int i = choice->pos.first, j = choice->pos.second;
    int x, y;

    /* 要准备落子的位置上已经有棋子，或者在这个位置落子不能吃掉对方任何棋子的话，说
       明这个action不合理，直接返回 */
    if (board->cell[i][j].color != SPACE || board->cell[i][j].stable == 0 || player
       == SPACE)
    {
        return -1;
    }


    board->cell[i][j].color = player;
    board->cell[i][j].stable = 0;


    if (player == WHITE)
    {
        board->whiteNum++;
```

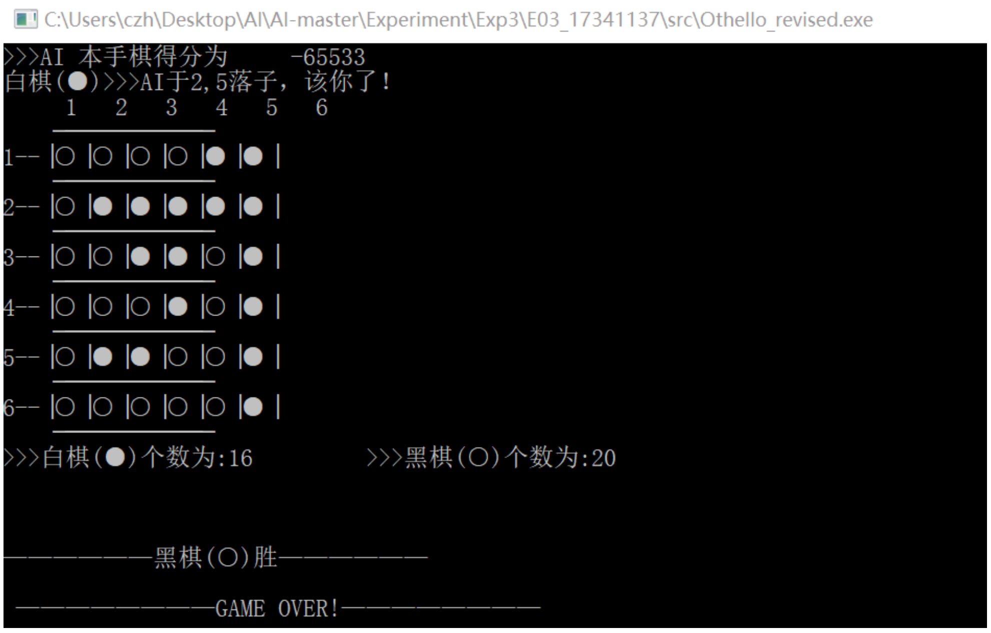
```c
        }
        else if (player == BLACK)
        {
            board->blackNum++;
        }



        for (x = -1; x <= 1; x++)
        {
            for (y = -1; y <= 1; y++)
            {

                //需要在每个方向（8个）上检测落子是否符合规则（能否吃子）


                if (x || y)
                {
                    int i2, j2;
                    unsigned num = 0;
                    for (i2 = i + x, j2 = j + y; i2 >= 0 && i2 <= 5 && j2 >= 0 && j2 <= 5; i2 += x, j2 += y)
                    {
                        if (board->cell[i2][j2].color == (enum Option) -player)
                        {
                            num++;
                        }
                        else if (board->cell[i2][j2].color == player)
                        {
                            board->whiteNum += (player*WHITE)*num;
                            board->blackNum += (player*BLACK)*num;

                            for (i2 -= x, j2 -= y; num>0; num--, i2 -= x, j2 -= y)
                            {
                                board->cell[i2][j2].color = player;
                                board->cell[i2][j2].stable = 0;
                            }
                            break;
                        }
                        else if (board->cell[i2][j2].color == SPACE)
                        {
                            break;
                        }
                    }
                }
            }
        }
        return 0;
    }


    void Othello::Stable(Othello *board)
    {
        int i, j;
        for (i = 0; i<6; i++)
        {
            for (j = 0; j<6; j++)
            {
                if (board->cell[i][j].color != SPACE)
                {
                    int x, y;
                    board->cell[i][j].stable = 1;

                    for (x = -1; x <= 1; x++)
                    {
                        for (y = -1; y <= 1; y++)
                        {
                            /* 4个方向 */
                            if (x == 0 && y == 0)
                            {
                                x = 2;
                                y = 2;
                            }
                            else
                            {
                                int i2, j2, flag = 2;
                                for (i2 = i + x, j2 = j + y; i2
                                    >= 0 && i2 <= 5 && j2 >= 0
                                    && j2 <= 5; i2 += x, j2 += y
                                )
                                {
                                    if (board->cell[i2][j2].color != board->cell[i][j].color)
                                    {
                                        flag--;
                                        break;
                                    }
                                }

                                for (i2 = i - x, j2 = j - y; i2 >= 0 && i2 <= 5 && j2 >= 0 && j2 <= 5; i2 -= x, j2 -= y)
                                {
                                    if (board->cell[i2][j2].color != board->cell[i][j].color)
                                    {
                                        flag--;
                                        break;
                                    }
                                }

                                if (flag)    /* 在某一条线上稳定 */
                                {
                                    board->cell[i][j].stable++;
                                }
                            }
                        }
                    }
                }
            }
        }
    }

    int Othello::Judge(Othello *board, enum Option player)
    {
        int value = 0;
        int i, j;
        Stable(board);

        // 对稳定子给予奖励
        for (i = 0; i<6; i++)
        {
                for (j = 0; j<6; j++)
                {
                        value += (board->cell[i][j].color)*(board->cell[i][j].stable);
                }
        }

         int V[6][6] = {{ 20,  -8,  11,  11,  -8,  20},
                                        { -8, -15,  -4,  -4, -15,  -8},
                                        { 11,  -4,   2,   2,  -4,  11},
                                        { 11,  -4,   2,   2,  -4,  11},
                                        { -8, -15,  -4,  -4, -15,  -8},
                                        { 20,  -8,  11,  11,  -8,  20}};

         for (int i = 0; i < 6; ++i)
         {
                for (int j = 0; j < 6; ++j)
                {
                        value += V[i][j] * board->cell[i][j].color;
                }
         }

        // 行动力计算
        int my_mov, opp_mov, mov = 0;
        my_mov = Rule(board, player);
        opp_mov = Rule(board, (enum Option) - player);
        if(my_mov > opp_mov)
                value += 78.922 * (100.0 * my_mov)/(my_mov + opp_mov);
        else if(my_mov < opp_mov)
                value += 78.922 * -(100.0 * opp_mov)/(my_mov + opp_mov);

        return value*player;
}
```

## 4 Results

从上面的图可以看到，最终是执黑棋的我的 AI 击败了执白棋的电脑的 AI。代码主要修改了两部分，一部分是 Judge 函数里的 Evaluation function，还有就是剪枝算法做了修改：使用自己写的 Negamax 算法，而不是原文件内的算法。不过这个程序要执行好久，大约 7-10min 左右才能看到结果。运行后不用进行任何操作，自己会交替显示双方的情况。



```
C:\Users\czh\Desktop\AI\AI-master\Experiment\Exp3\E03_17341137\src\Othello_revised.exe
>>>人机对战开始:
>>>请选择执黑棋(○),或执白棋(●): 输入1为黑棋，-1为白棋
1
```



```
C:\Users\czh\Desktop\AI\AI-master\Experiment\Exp3\E03_17341137\src\Othello_revised.exe
>>>AI 本手棋得分为      -65533
白棋(●)>>>AI于2,5落子, 该你了!
     1   2   3   4   5   6
   ________________________
1-- |○ |○ |○ |○ |● |● |
   ________________________
2-- |○ |● |● |● |● |● |
   ________________________
3-- |○ |○ |● |● |○ |● |
   ________________________
4-- |○ |○ |○ |● |○ |● |
   ________________________
5-- |○ |● |● |○ |○ |● |
   ________________________
6-- |○ |○ |○ |○ |○ |● |
   ________________________
>>>白棋(●)个数为:16          >>>黑棋(○)个数为:20


————————黑棋(○)胜————————

 ————————————GAME OVER!————————————
```