E03 Othello Game (α − β pruning)

学号	姓名	日期
18340215	张天祎	2020.9.22

1、Othello

Othello (or Reversi) is a strategy board game for two players, played on an 8 × 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、Task

- In order to reduce the complexity of the game, we think the board is 6 × 6.
- There are several evaluation functions that involve many aspects, you can turn to http://www.cs.cornell.edu/~yuli/othello/othello.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.
- Please choose an appropriate evaluation function and use min-max and α − β 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.
- Write the related codes and take a screenshot of the running results in the file named E03 StudentNumber.pdf,and send it to ai 2020@foxmail.com, the deadline is 2020.09.23 23:59:59.

3、Codes

```
//使用了Othello.cpp框架，主要添加的函数为dynamic_heuristic_evaluation_function()和
MyFind()

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <stdlib.h>

using namespace std;

int const MAX = 65534;

int deepth = 10; //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

    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

//基于参考文件形成的启发式函数
int dynamic_heuristic_evaluation_function(Othello *board, enum Option player)
{
    int value = 0, my_front_tiles = 0, opp_front_tiles = 0, my_tiles = 0,
opp_tiles = 0;
    int i, j, k, x, y;
    double d = 0, p = 0, f = 0, c = 0, l = 0;
    board->Stable(board);
    int X1[] = {-1, -1, 0, 1, 1, 1, 0, -1};
    int Y1[] = {0, 1, 1, 1, 0, -1, -1, -1};
    int V[6][6] = {
        (32, -16, 16, 16, -16, 32),
        (-16, -7, -2, -2, -7, -16),
        (16, -2, 0, 0, -2, 16),
        (16, -2, 0, 0, -2, 16),
        (-16, -7, -2, -2, -7, -16),
        (32, -16, 16, 16, -16, 32)};

    for (i = 0; i < 6; i++)
```

```
        for (j = 0; j < 6; j++)
        {
            if (board->cell[i][j].color == player)
            {
                d += V[i][j];
                my_tiles++;
            }
            else if (board->cell[i][j].color == -player)
            {
                d -= V[i][j];
                opp_tiles++;
            }
            if (board->cell[i][j].color)
            {
                for (k = 0; k < 8; k++)
                {
                    x = i + X1[k];
                    y = j + Y1[k];
                    if (x >= 0 && x < 6 && y >= 0 && y < 6 && board->cell[i]
[j].color == 0)
                    {
                        if (board->cell[i][j].color == player)
                            my_front_tiles++;
                        else
                            opp_front_tiles++;
                        break;
                    }
                }
            }
        }
    if (my_tiles > opp_tiles)
        p = (100.0 * my_tiles) / (my_tiles + opp_tiles);
    else if (my_tiles < opp_tiles)
        p = (100.0 * opp_tiles) / (my_tiles + opp_tiles);
    else
        p = 0;

    if (my_front_tiles > opp_front_tiles)
        f = -(100.0 * my_front_tiles) / (my_front_tiles + opp_front_tiles);
    else if (my_front_tiles < opp_front_tiles)
        f = (100.0 * opp_front_tiles) / (my_front_tiles + opp_front_tiles);
    else
        f = 0;

    my_tiles = opp_tiles = 0;
    if (board->cell[0][0].color == player)
        my_tiles++;
    else if (board->cell[0][0].color == -player)
        opp_tiles++;
    if (board->cell[0][5].color == player)
        my_tiles++;
    else if (board->cell[0][5].color == -player)
        opp_tiles++;
    if (board->cell[5][0].color == player)
        my_tiles++;
    else if (board->cell[5][0].color == -player)
        opp_tiles++;
    if (board->cell[5][5].color == player)
```

```
        my_tiles++;
    else if (board->cell[5][5].color == -player)
        opp_tiles++;
    c = 50 * (my_tiles - opp_tiles);

    my_tiles = opp_tiles = 0;
    if (board->cell[0][0].color == 0)
    {
        if (board->cell[0][1].color == player)
            my_tiles++;
        else if (board->cell[0][1].color == -player)
            opp_tiles++;
        if (board->cell[1][1].color == player)
            my_tiles++;
        else if (board->cell[1][1].color == -player)
            opp_tiles++;
        if (board->cell[1][0].color == player)
            my_tiles++;
        else if (board->cell[1][0].color == -player)
            opp_tiles++;
    }
    if (board->cell[0][5].color == 0)
    {
        if (board->cell[0][4].color == player)
            my_tiles++;
        else if (board->cell[0][4].color == -player)
            opp_tiles++;
        if (board->cell[1][4].color == player)
            my_tiles++;
        else if (board->cell[1][4].color == -player)
            opp_tiles++;
        if (board->cell[1][5].color == player)
            my_tiles++;
        else if (board->cell[1][5].color == player)
            opp_tiles++;
    }
    if (board->cell[5][0].color == 0)
    {
        if (board->cell[5][1].color == player)
            my_tiles++;
        else if (board->cell[5][1].color == -player)
            opp_tiles++;
        if (board->cell[4][1].color == player)
            my_tiles++;
        else if (board->cell[4][1].color == -player)
            opp_tiles++;
        if (board->cell[4][0].color == player)
            my_tiles++;
        else if (board->cell[4][0].color == -player)
            opp_tiles++;
    }
    if (board->cell[5][5].color == 0)
    {
        if (board->cell[4][5].color == player)
            my_tiles++;
        else if (board->cell[4][5].color == -player)
            opp_tiles++;
        if (board->cell[4][4].color == player)
```

```
            my_tiles++;
        else if (board->cell[4][4].color == -player)
            opp_tiles++;
        if (board->cell[5][4].color == player)
            my_tiles++;
        else if (board->cell[5][4].color == player)
            opp_tiles++;
    }
    l = -25 * (my_tiles - opp_tiles);

    return (int)(p + f * 7 + c * 80 + l * 40);
}

//在原AI的Find操作上修改得到
Do *MyFind(Othello *board, enum Option player, int step, int min, int max, Do
*choice)
{
    int i, j, k, num;
    Do *allChoices;
    choice->score = -MAX;
    choice->pos.first = -1;
    choice->pos.second = -1;

    num = board->Rule(board, player);
    if (num == 0)
    {
        if (board->Rule(board, (enum Option) - player))
        {
            Othello tempBoard;
            Do nextChoice;
            Do *pNextChoice = &nextChoice;
            board->Copy(&tempBoard, board);
            pNextChoice = MyFind(&tempBoard, (enum Option) - player, step - 1, -
max, -min, pNextChoice);
            choice->score = -pNextChoice->score;
            return choice;
        }
        else
        {
            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)
    {
        choice->score = dynamic_heuristic_evaluation_function(board, player);
```

```
        return choice;
    }

    allChoices = (Do *)malloc(sizeof(Do) * num);
    k = 0;
    for (i = 0; i < 6; i++) //这里粗糙地直接处理，没有像AI那样分组处理，可能会慢一点
（step=10情况下变化不大），但不影响结果
    {
        for (j = 0; j < 6; j++)
        {
            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++) //永远求最大值，最小值的情况通过取负解决
    {
        Othello tempBoard;
        Do thisChoice, nextChoice;
        Do *pNextChoice = &nextChoice;
        thisChoice = allChoices[k];
        board->Copy(&tempBoard, board);
        board->Action(&tempBoard, &thisChoice, player);
        pNextChoice = MyFind(&tempBoard, (enum Option) - player, step - 1, -max,
-min, pNextChoice);
        thisChoice.score = -pNextChoice->score;

        if (thisChoice.score > min && thisChoice.score < max) //接下来的分支中可能还
有值在thisChoice.score和max之间，所以需要继续搜索
        {                                                     //只有
thisChoice.score>min时才需要更新min
            min = thisChoice.score;
            choice->score = thisChoice.score;
            choice->pos.first = thisChoice.pos.first;
            choice->pos.second = thisChoice.pos.second;
        }
        else if (thisChoice.score >= max) //上层的min肯定不会选择当前的节点，所以剩余的
分支都可以减掉
        {
            choice->score = thisChoice.score;
            choice->pos.first = thisChoice.pos.first;
            choice->pos.second = thisChoice.pos.second;
            break;
        }
    }
    free(allChoices);
    return choice;
}

//最大最小博弈与α-β剪枝
Do *Find(Othello *board, enum Option player, int step, int min, int max, Do
*choice)
{
```

```
    int i, j, k, num;
    Do *allChoices;
    choice->score = -MAX;
    choice->pos.first = -1;
    choice->pos.second = -1;

    num = board->Rule(board, player);
    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 = Find(&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;
    }

    allChoices = (Do *)malloc(sizeof(Do) * num);
    k = 0;
    for (i = 0; i < 6; i++)
    {
        for (j = 0; j < 6; j++)
        {
            if (i == 0 || i == 5 || j == 0 || j == 5)
            {
                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++)
    {
        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++)
    {
        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++)
    {
        Othello tempBoard;
        Do thisChoice, nextChoice;
        Do *pNextChoice = &nextChoice;
        thisChoice = allChoices[k];
        board->Copy(&tempBoard, board);
        board->Action(&tempBoard, &thisChoice, player);
        pNextChoice = Find(&tempBoard, (enum Option) - player, step - 1, -max, -
min, pNextChoice);
        thisChoice.score = -pNextChoice->score;
```

```
        if (thisChoice.score > min && thisChoice.score < max) /* 可以预计的更优值
*/
        {
            min = thisChoice.score;
            choice->score = thisChoice.score;
            choice->pos.first = thisChoice.pos.first;
            choice->pos.second = thisChoice.pos.second;
        }
        else if (thisChoice.score >= max) /* 好的超乎预计 */
        {
            choice->score = thisChoice.score;
            choice->pos.first = thisChoice.pos.first;
            choice->pos.second = thisChoice.pos.second;
            break;
        }
        /* 不如已知最优值 */
    }
    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);
        player = (enum Option) - 1;
        cout << player << endl;
        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 = "";
```

```
        char Player[10] = {};
        if (present == BLACK)
        {
            //Player = "黑棋(○)";
            strcpy(Player, "黑棋(○)");
        }
        else if (present == WHITE)
        {
            //Player = "白棋(●)";
            strcpy(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)
            {
                while (1)
                {
                    cout << Player << " \n >>>请输入棋子坐标（空格相隔 如“3 5”代表第3
行第5列）:\n";

                    //cin >> i >> j;
                    pChoice = MyFind(pBoard, present, deepth, -MAX, MAX,
pChoice);
                    //pChoice = Find(pBoard, present, deepth, -MAX, MAX,
pChoice);
                    i = pChoice->pos.first;
                    j = pChoice->pos.second;
                    cout << i << ' ' << j << endl;
                    if (i < 0 || i > 5 || j < 0 || j > 5 || pBoard->cell[i]
[j].color != SPACE || pBoard->cell[i][j].stable == 0)
                    {
                        cout << ">>>此处落子不符合规则，请重新选择    \n";
                        board.Show(pBoard);
                    }
                    else
                    {
                        break;
                    }
                }
                //system("cls");
                cout << ">>>玩家 本手棋得分为      " << pChoice->score << endl;
                //system("pause");
                cout << ">>>按任意键继续" << endl;
            }
            else //AI下棋
```

```
            {
                cout << Player << ".........................";

                pChoice = Find(pBoard, present, deepth, -MAX, MAX, pChoice);
                i = pChoice->pos.first;
                j = pChoice->pos.second;
                //system("cls");
                cout << ">>>AI 本手棋得分为      " << pChoice->score << endl;
            }

            board.Action(pBoard, pChoice, present);
            num++;
            cout << Player << ">>>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)
{
    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;
```

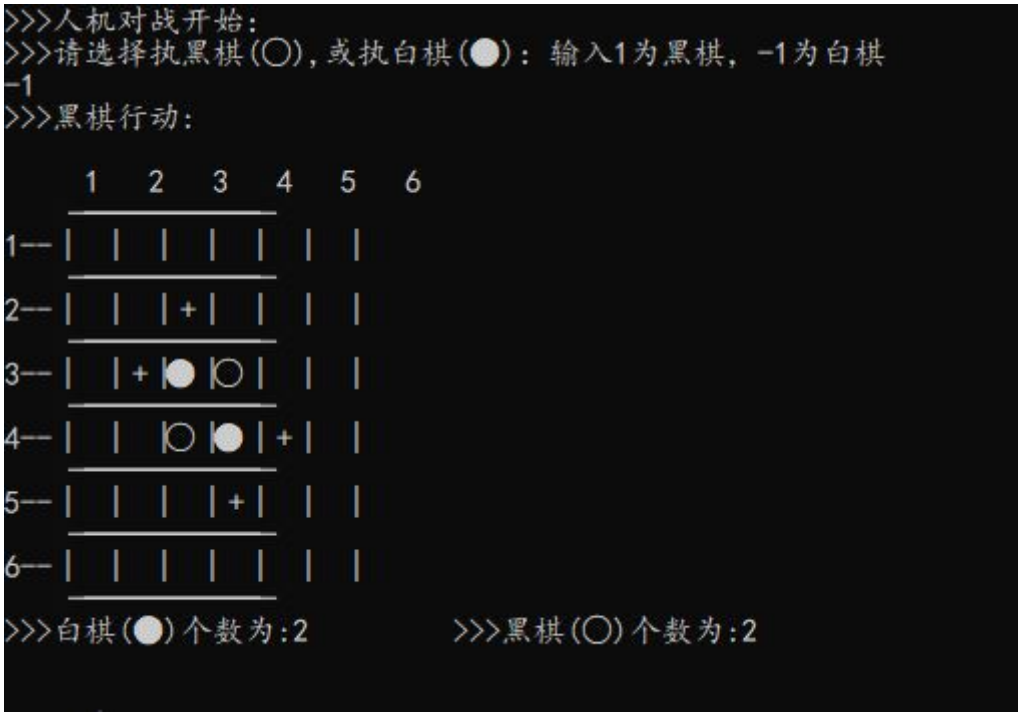
```
                                }
                                else if (board->cell[i2][j2].color == SPACE)
                                {
                                    break;
                                }
                            }
                        }
                    }
                }

                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;

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

    value += 64 * board->cell[0][0].color;
    value += 64 * board->cell[0][5].color;
    value += 64 * board->cell[5][0].color;
    value += 64 * board->cell[5][5].color;
    value -= 32 * board->cell[1][1].color;
    value -= 32 * board->cell[1][4].color;
    value -= 32 * board->cell[4][1].color;
    value -= 32 * board->cell[4][4].color;

    return value * player;
}
```

4、Results

我的启发式函数仅能在执白时获胜，结果如下



启发式函数的设计主要考虑到优先占边角、不能让对手占边角、不能让对手轻易地吃掉自己的子等几方面的均衡。框架中α-β剪枝的代码写得十分精彩，大部分都沿用了。