# **亚马逊期末大作业报告**

1900094805 裴虎镇 1900094806 王逸凡

# 一、代码结构

代码由多个函数组成

1. Main函数（主界面）

有最开始的初始动画，之后可以输入1，2，3，4来进入不同的功能中，最多可输错3次，并每次会出现不同的警告。

int main()//主页

{

    dance()；

    isRunning = true;

    system("cls");

    cout << "====================================================\n";

    Sleep(0.1 \* 1000);

    cout << "                  来啊来啊快来快活啊\n";

    cout << "====================================================\n";

    Sleep(0.5 \* 1000);

    system("cls");

    cout << "====================================================\n";

    cout << "                    快乐亚马逊棋\n";

    cout << "====================================================\n";

    Sleep(0.5 \* 1000);

    cout << "                   1. 新游戏\n";

    cout << "                   2. 继续上次游戏\n";

    cout << "                   3. 游戏说明\n";

    cout << "                   4. 退出游戏\n";

    cout << "                   请输入 [1/2/3]: ";

    //int line;

    int wrongcnt = 0;

    while (1)

    {

        // 4.2:尝试更严格的输入

        cin.getline (strcin, 1000);

        if (strcin[0] != '1' && strcin[0] != '2' && strcin[0] != '3' && strcin[0] != '4')

        {

            cout << "别闹! 选择[1/2/3]中的数字" << endl;

        }

        else if (strlen (strcin) != 1)

        {

            cout << "你干啥打这么多字？输入一个就够了" << endl;

        }

        else break;

        wrongcnt++;

        switch (wrongcnt)

        {

            case 1: cout << "再给你一次重来得机会" << endl; break;

            case 2: cout << "事不过三 别太过分了翔哥" << endl; break;

            case 3: cout << "你有本事再输错一次？" << endl; break;

        }

        if (wrongcnt == 4)

        {

            isRunning = false;

            break;

        }

    }

    switch (strcin[0])

    {

        case '1':

            Board(true);

            while(isRunning)

            {

                DrawBoard();

                ShowMenu();

            }

            break;

        case '2':

            Board(false);

            while(isRunning)

            {

                DrawBoard();

                ShowMenu();

            }

            break;

        case '3':

            rule();

            char hhh;

            cout << "Start a new game? [y/n]: ";

            cin >> hhh;

            if(hhh == 'y')

            {

                Board(true);

                while(isRunning)

                {

                    DrawBoard();

                    ShowMenu();

                }

                break;

            }

            else break;

        case '4':

            break;

    }

}

1. Board函数（运行新游戏或继续上次保存游戏）

inline void Board(bool isNewGame)

{

    int turnID = 0;

    if (isNewGame)

    {

        for (int i = 0; i < GRIDSIZE; i++)

        {

            for (int j = 0; j < GRIDSIZE; j++)

                map[i][j] = 0;

        }

        // 定义黑白棋子位置

        map[0][2] = map[2][0] = map[5][0] = map[7][2] = 1;

        map[0][5] = map[2][7] = map[5][7] = map[7][5] = -1;

        // 把棋子的位置存入pos中

    }

    else

    {

        // 把文件中的信息存回来

        ifstream fin(SAVE\_FILE);

        if (!fin)

        {

            cout << "算了，你还是玩新的一局吧";

            Sleep(1 \* 1000);

            isRunning = false;

            return;

        }

        int x;

        fin >> x;

        round\_amazon = x;

        for (int i = 0; i < GRIDSIZE; i++)

        {

            for (int j = 0; j < GRIDSIZE; j++)

            {

                fin >> x;

                map[i][j] = x;

            }

        }

        fin.close();

    }

}

主界面上输入1或者是输入3之后再输入y的时候，isNewGame = true,会初始化游戏状态。

输入2的时候，会从文件夹中寻找存档文件来调回上次的游戏状态

3. DrawLine, DrawBoard（画棋盘）

inline void DrawLine(int len)

{

    cout << "  +";

    for (int i = 0; i < len; i++)

        cout << "---+";

    cout << "\n";

}

inline void DrawBoard()

{

    system("cls");//清屏

    ///////////////////////////////////////////////////////////////////////

    cout << "            --Round " << round\_amazon << "--" << endl;

    ////////////////////////////////////////////////////////////////////////

    cout << "    "; for (int x = 0; x < GRIDSIZE; x++) cout << x << "   "; cout << "\n";

    DrawLine(GRIDSIZE);

    for (int y = 0; y < GRIDSIZE; y++)

    {

        cout << y << " |";

        for (int x = 0; x < GRIDSIZE; x++)

        {

            char ch;

            switch (map[x][y])

            {

            case 1:

                ch = 'B'; break;

            case -1:

                ch = 'W'; break;

            case 2:

                ch = 'O'; break;

            default:

                ch = ' '; break;

            }

            cout << ' ' << ch << " |";

        }

        cout << "\n";

        DrawLine(GRIDSIZE);

    }

    cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

}

4. SaveBoard, SaveBoard\_tmp

void SaveBoard()

{

    // 向SAVE\_FILE存入数据

    ofstream fout(SAVE\_FILE);

    fout << round\_amazon << '\n';

    for (int i = 0; i < GRIDSIZE; i++)

    {

        for (int j = 0; j < GRIDSIZE; j++)

        {

            fout << map[i][j] << ' ';

        }

        fout << '\n';

    }

    fout.close();

}

inline void SaveBoard\_tmp()

{

    // 向SAVE\_FILE存入数据

    ofstream fout(SAVE\_FILE\_tmp);

    fout << round\_amazon << ' ';

    fout << '\n';

    for (int i = 0; i < GRIDSIZE; i++)

    {

        for (int j = 0; j < GRIDSIZE; j++)

        {

            fout << map[i][j] << ' ';

        }

        fout << '\n';

    }

    fout.close();

}

SaveBoard在当退出游戏时若选择保存游戏，就会新创一个文件来保存当前的对局状态。

SaveBoard\_tmp在每回合保存棋局的状态，并在选择悔棋的时候调出相应状态。

5. 游戏菜单（游戏开始后的主要界面）

inline void ShowMenu()//下棋界面的菜单

{

    cout << "1. 选择移动\n";

    cout << "2. 悔棋\n";

    cout << "3. 新游戏\n";

    cout << "4. 退出游戏\n";

    cout << "请输入[1/2/3/4]: ";

    //int line;

    while (1)

    {

        // 4.2:尝试更严格的输入判别

        cin.getline (strcin, 1000);

        if (strcin[0] != '1' && strcin[0] != '2' && strcin[0] != '3' && strcin[0] != '4')

        {

            cout << "别调皮! 选择[1/2/3/4]中的数字 :";

            continue;

        }

        if (strlen (strcin) != 1)

        {

            cout << "你干啥打这么多字？输入一个就够了" << endl;

            continue;

        }

        break;

    }

    switch (strcin[0])

    {

        case '1':   // 选择移动

        {

            SaveBoard\_tmp();

            int tmp2 = 1;

            int flag1, flag2;

            while(tmp1)

            {

                // 4.2 尝试更严格的输入

                cout << "选择黑棋(1) or 白棋(-1) [1/-1]: " << endl;

                cin.getline (strcin, 1000);

                while (1)

                {

                    if (strlen (strcin) != 1 && strlen (strcin) != 2)

                    {

                        cout << "再给你一次重来的机会" << endl;

                        cin.getline (strcin, 1000);

                        continue;

                    }

                    if (strlen (strcin) == 1)

                    {

                        if (strcin[0] != '1')

                        {

                            cout << "再给你一次重来的机会" << endl;

                            cin.getline (strcin, 1000);

                            continue;

                        }

                        else

                        {

                            player = 1;

                            tmp1 = 0;

                            break;

                        }

                    }

                    else

                    {

                        if (strcin[0] != '-' || strcin[1] != '1')

                        {

                            cout << "再给你一次重来的机会" << endl;

                            cin.getline (strcin, 1000);

                            continue;

                        }

                        else

                        {

                            player = -1;

                            tmp1 = 0;

                            break;

                        }

                    }

                }

            }

算法部分！！！！！！！！！！！

太长了，后面详解！

                                round\_amazon++;

                }

                else

                    cout << "请再尝试一次输入" << endl;

            }

        }

        break;

        case '2': // 悔棋

        {

            ifstream fin(SAVE\_FILE\_tmp);

            if (!fin)

            {

                cout << "呵呵，想多了吧";

                Sleep(1 \* 1000);

                break;

            }

            int x;

            fin >> x;

            round\_amazon = x;

            for (int i = 0; i < GRIDSIZE; i++)

            {

                for (int j = 0; j < GRIDSIZE; j++)

                {

                    fin >> x;

                    map[i][j] = x;

                }

            }

            fin.close();

            break;

        }

        case '3': // 新游戏

        {

            round\_amazon = 1;

            Board(true);

            DrawBoard();

            break;

        }

        case '4': // 退出游戏

        {

            fange();

            cout << "你要保存吗 [y/n]: ";

            char ans;

            cin >> ans;

            if (ans == 'y') SaveBoard();

            else

            {

                remove(SAVE\_FILE\_tmp);

            }

            //若不选择保存的话就把原来的临时文件删除

            isRunning = false;

            break;

        }

    }

}

可以选择4种不同操作：

1. 选择1的时候，若是第一次输入，则可以选择黑白方。后需输入走法；
2. 选择2的时候，调出临时文件，来进行悔棋操作；
3. 选择3的时候，直接使isNewGame重新为1，调出Board函数，初始化棋局状态；
4. 选择4的时候，可选择y来保存当前棋局，也可以选择n来直接退出游戏。

# 二、算法部分

采用极大极小值算法，极大化自己最好的棋步并极小化对方最好的棋步。评估函数考虑了territory,position,mobility三个因素，并分成开局、残局、终局三种情况处理。考虑到时间问题，前六回合只往后搜一步，之后往后搜两步，并且用到alpha剪枝来节省时间。

1. 判断用户输入的是否合法

首先，判断是不是按照格式输入。这里用getline可以做到更加严格的判别用户的输入。

cout << "输入你的走法 格式[x0 y0 x1 y1 x2 y2]" << endl;

cin.getline (strcin, 1000);

while (1)

{

    if (strlen (strcin) != 11)

    {

     cout << "请按照[x0 y0 x1 y1 x2 y2]的格式输入！" << endl;

       cin.getline (strcin, 1000);

       continue;

}

    if (strcin[1] != ' ' || strcin[3] != ' ' || strcin[5] != ' ' || strcin[7] != ' ' || strcin[9] != ' ')

    {

    cout << "请按照[x0 y0 x1 y1 x2 y2]的格式输入！" << endl;

     cin.getline (strcin, 1000);

        continue;

    }

    else

    {

         x0 = strcin[0] - '0';

         y0 = strcin[2] - '0';

         x1 = strcin[4] - '0';

         y1 = strcin[6] - '0';

        x2 = strcin[8] - '0';

        y2 = strcin[10] - '0';

            break;

    }

之后，判断用户的输入是否符合亚马逊棋的规则。

这里先是要用到inMap和ProcStep函数。

inMap判断(x,y)是否在棋盘内，ProcStep判断输入的三个点是否有可能合法（并没有判断是否是皇后走法）

inline bool inMap (int x, int y)

{

    if (x < 0 || x >= GRIDSIZE || y < 0 || y >= GRIDSIZE)

        return false;

    return true;

}

bool ProcStep (int x0, int y0, int x1, int y1, int x2, int y2, int player, bool check\_only)

{

    if (!inMap (x0, y0) || !inMap (x1, y1) || !inMap (x2, y2))

        return false;

    if (map[x0][y0] != player || map[x1][y1] != 0)

        return false;

    if (map[x2][y2] != 0 && !(x2 == x0 && y2 == y0))

        return false;

    if (!check\_only)

    {

        map[x0][y0] = 0;

        map[x1][y1] = player;

        map[x2][y2] = OBSTACLE;

    }

    return true;

}

最后，判断是否符合皇后走法。

// 判断是否合法

int checkrule = 0,i1, dio, i2, jojo;

for (i1 = 0; i1 < 8; i1++)

{

     for (dio = 1; dio < 8; dio++)

     {

          if (x0 + dx[i1] \* dio == x1 && y0 + dy[i1] \* dio == y1)

          {

               checkrule = 1;

                 break;

         }

     }

       if (checkrule)

            break;

}

if (!checkrule)

{

      cout << "请想好了再输入！" << endl;

        continue;

}

checkrule = 0;

for (i2 = 0; i2 < 8; i2++)

{

      for (jojo = 1; jojo < 8; jojo++)

      {

          if (x1 + dx[i2] \* jojo == x2 && y1 + dy[i2] \* jojo == y2)

          {

              checkrule = 1;

                 break;

           }

    }

      if (checkrule)

            break;

}

if (!checkrule)

{

      cout << "请想好了再输入！" << endl;

           continue;

}

1. 枚举走法

前六个回合往后搜一步，之后往后搜两步。

在搜两步的时候用到了flag1和flag2，初始值均为0。如果AI经过枚举后发现自己已经没有可以走的选择了，说明人类赢了，此时就令flag1=1，代表人类赢。如果在第二步的枚举发现人类已经没有可以走的选择了，说明AI赢了，此时令flag2=1并不再继续枚举下去，AI赢。

以下为alpha剪枝。temp为当前局面的评估函数值，alpha为当前的极大值，first为1时代表这是第一次枚举（即不可能剪枝），0代表不是第一次枚举。checkalpha代表是否剪枝。

注：此时枚举的玩家走法为（ii，jj，xxxx，yyyy，xxxxx，yyyyy）

if (temp < alpha && !first)

{

    checkalpha = 1;

   player \*= -1;

    map[xxxxx][yyyyy] = 0;

    map[xxxx][yyyy] = 0;

    map[ii][jj] = player;

        break;

 }

由于代码过长，就不全贴了。

1. 评估函数

评估函数参考了文献《亚马逊棋机器博弈系统中评估函数的研究》。

评估函数包括evaluate，MOBILITY，Queenmove\_W，Queenmove\_B，Kingmove\_W，Kingmove\_B六个函数。

对于每个空格，Queenmove\_W将白方的四个棋子通过Queen走法走到该空格最少需要走的步数记录到数组Qwhite。Queenmove\_B将黑棋情况记录到数组Qblack。Kingmove\_W将白棋的king走法记录到数组Kwhite。Kingmove\_B将黑棋的king走法记录到数组Kblack。（这四个数组中所有数初始值均为1000，因此若走不到某个空格，该空格的值即为1000）

由于之前我尝试过自己写这四个函数，但是均以失败告终（或是时间过长），于是我采用了网上使用的宽搜的方法。

MOBILITY函数计算每个空格的周围八个格子中有多少空格，将数据存入数组mobility。

void MOBILITY ()

{

    for (int i = 0; i < GRIDSIZE; i++)

        for (int j = 0; j < GRIDSIZE; j++)

        {

            for (int k = 0; k < 8; k++)

            {

                int tempx = i + dx[k];

                int tempy = j + dy[k];

                if (inMap (tempx, tempy) && map[tempx][tempy] == 0)

                    mobility[i][j]++;

            }

        }

}

evaluate为double型函数，返回的是最终的评估值。利用文献中的内容，计算出t1，t2，p1，p2，m，之后再根据回合数进行不同的加权算法。（48步后就不用计算m）

double value;

if (turn <= 14)

   value = 0.2 \* t1 + 0.48 \* t2 + 0.11 \* p1 + 0.11 \* p2 + 0.2 \* m;

else if (turn <= 48)

    value = 0.4 \* t1 + 0.25 \* t2 + 0.2 \* p1 + 0.2 \* p2 + 0.06 \* m;

else

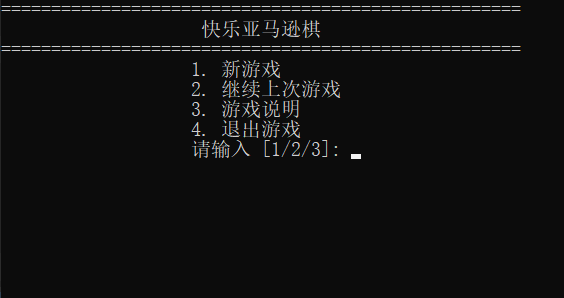
    value = 0.8 \* t1 + 0.1 \* t2 + 0.05 \* p1 + 0.05 \* p2;

return value;

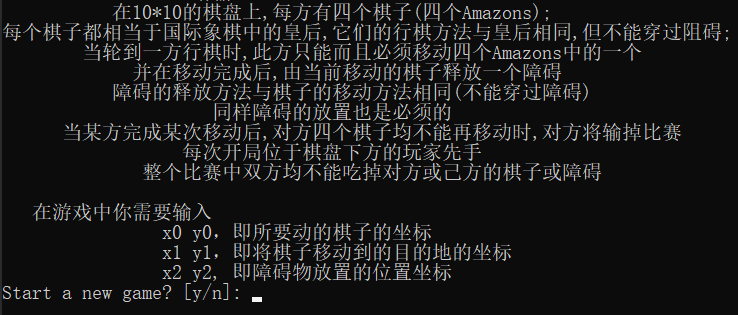
# 三、代码运行过程

开始

先有开始动画

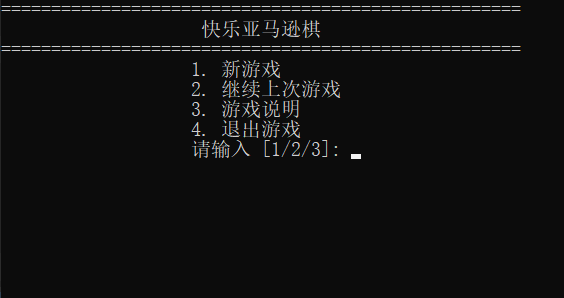


若选择游戏说明

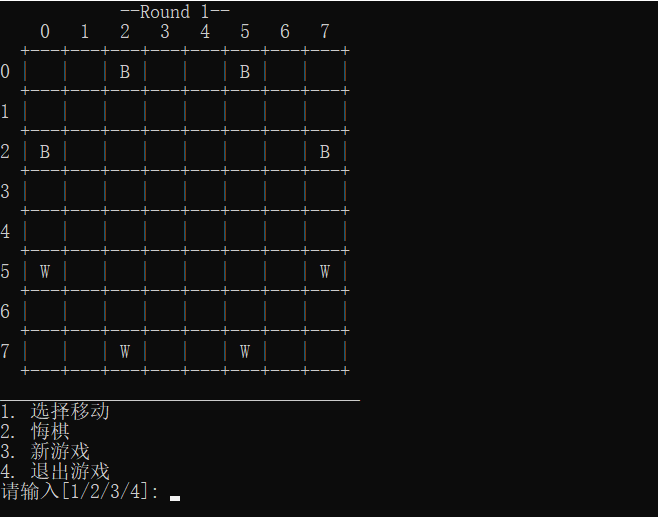


输入n，就关闭

输入y，开始新游戏

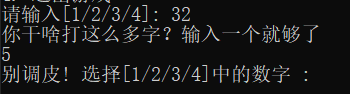
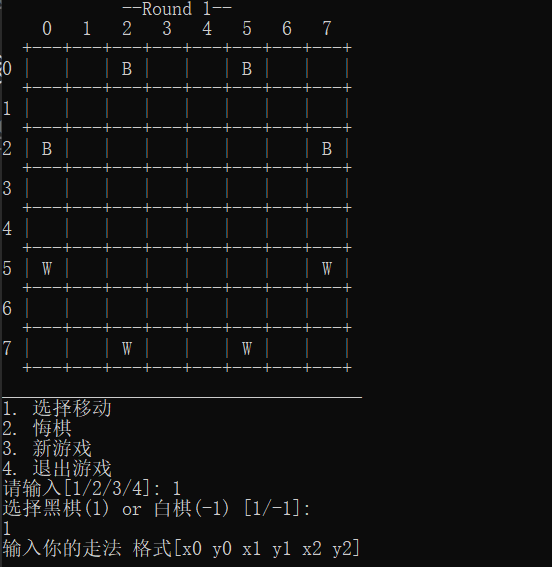


菜单中若输入1，相当于是输入3之后再输入y



上面的round显示回合数。

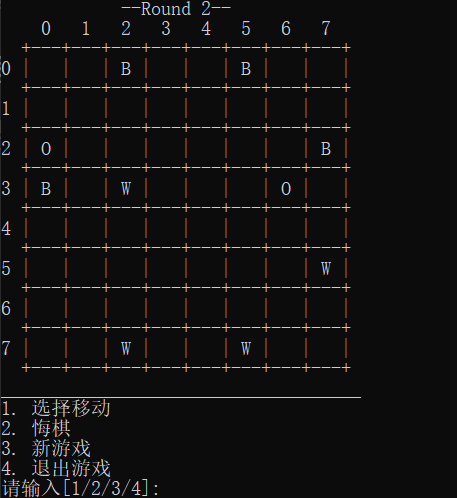
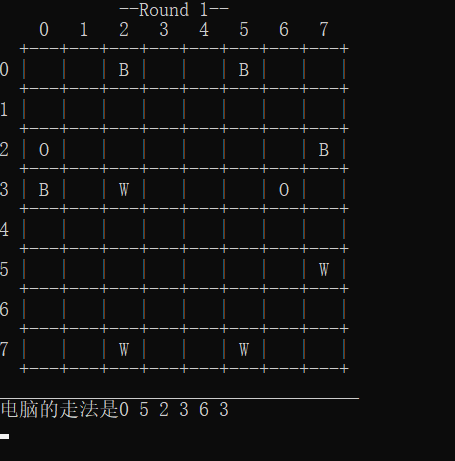
若输入1



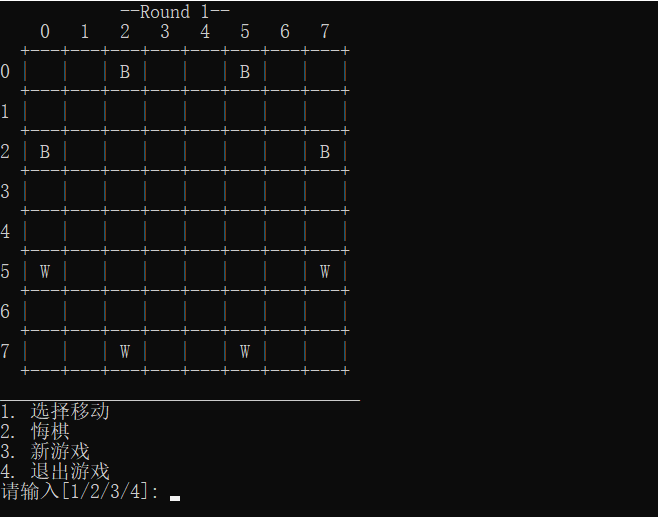
如果输错了，就会出现如上的消息

可输入1/-1来选择黑棋或白棋，然后就要再次输入走法。

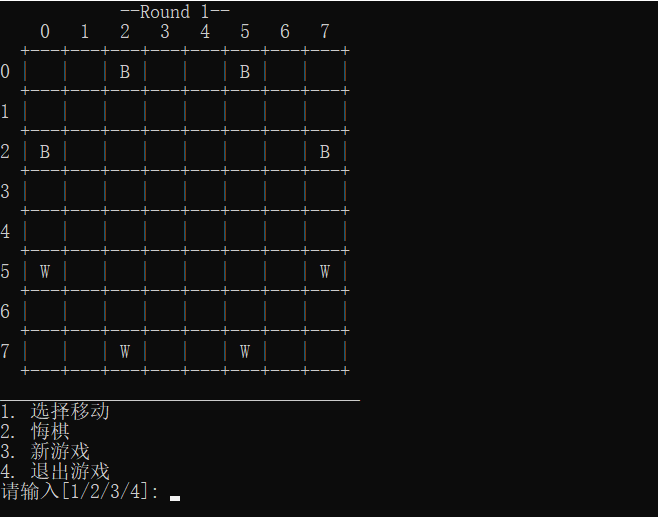
若走法为（0，2，0，3，0，2）的话，棋会一步步移动并后来会显示电脑的走法。



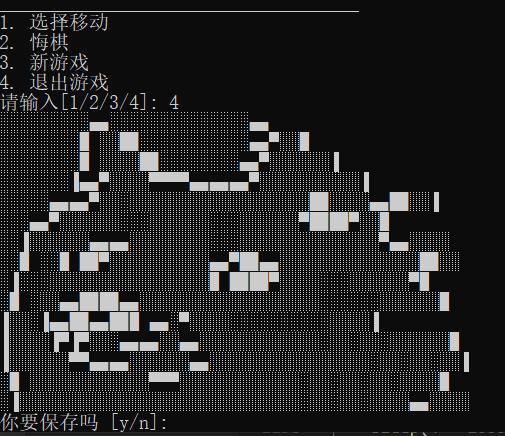
然后再次显示菜单



若输入2，就会回到前一个位置



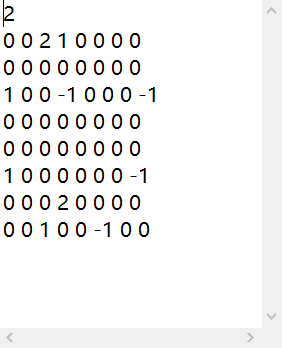
若输入3，将回到初始的棋局



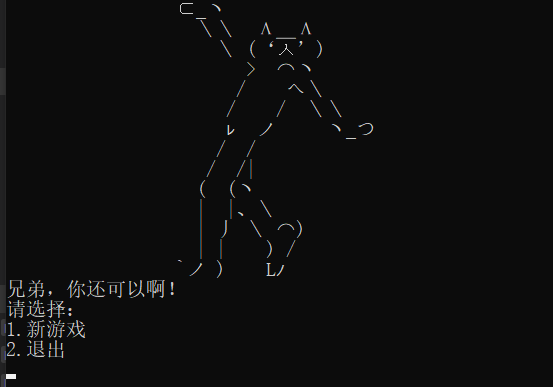
若输入4，出现图片，可输入y或n来选择保存或不保存



若输入y，即选择保存的话会文件夹中会出现save\_board.txt文件



如果输了，就会显示如上的画面，并显示开始新游戏或选择退出



若赢了就有如上的动画，并显示开始新游戏或选择退出

# 四、程序特色

1.严格的输入判别

程序中的每一次输入基本都是用了cin.getline用来判断输入的字符串是否符合要求。

2.小彩蛋

开局错误输入四次后自动退出程序，并且每次错误输入都有不同的提示。

3.很特别的开局动画，还有胜利和失败的动画

4.利用sleep来完成下棋的特效