人工智能实验

博弈树搜索实现黑白棋人机对战

16 级计科教务 2 班

16337327

郑映雪

实验题目

博弈树搜索实现黑白棋人机对战

实验内容

算法原理

1、MinMax 算法

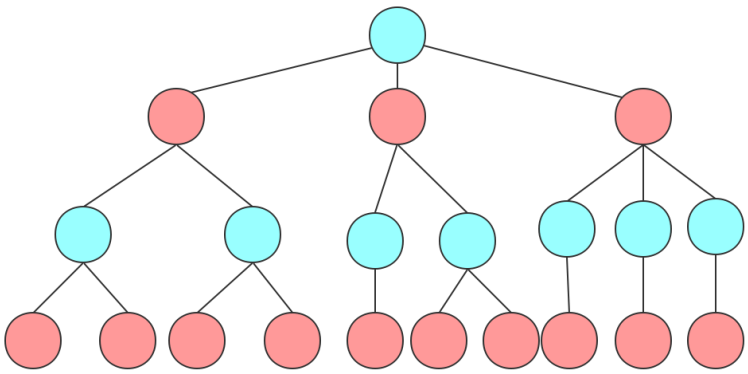
极大极小搜索的策略用一句简单的话概括就是，以当前状态为起始点，分别考虑双方接下来的若干步数，从所有的可能性中选择较好的走法来走。

极大极小算法适用于满足以下两个条件的游戏：

①零和游戏。零和的意思其实就是字面意思。假设一方赢，则得分为 1，另一方得分为-1；平局则双方得分为 0。这样两方的得分相加等于 0，所以这类游戏称为零和游戏。意义上来说，零和游戏是要么赢要么输要么平局的游戏，一方赢必定意味着另一方的输。

②完全信息的游戏。完全信息是指双方都知道之前所发生过的所有的步骤，从而可以进行下一步的推算。大部分棋类游戏都是完全信息的游戏。扑克牌就不是完全信息的游戏，因为玩家看不到对面的牌。

可以用树状图表示接下来可能的走法，假设我方赢是正，对方赢是负，则我一定希望我得到的分数越大越好，对方一定希望他得到的分数越小越好，于是我们有：



上图是一个 minmax 博弈树，其中蓝色为 max 节点，红色为 min 节点。由底层往上估值，若为 min 节点，则选取子节点中得分最小的值为自己的值，若为 max 节点，则选取子节点中得分最大的值为自己的值。这样可以达到“往下走几层从而选择一个较好的走法”的效果，这种算法也是“minmax”名字的原因。

但是，当层数增多的时候，博弈树会变得很庞大，从而每次建树需要进行很长时间。为了优化这种情况的发生，我们需要进行 alphabeta 剪枝。

2、alphabeta 剪枝

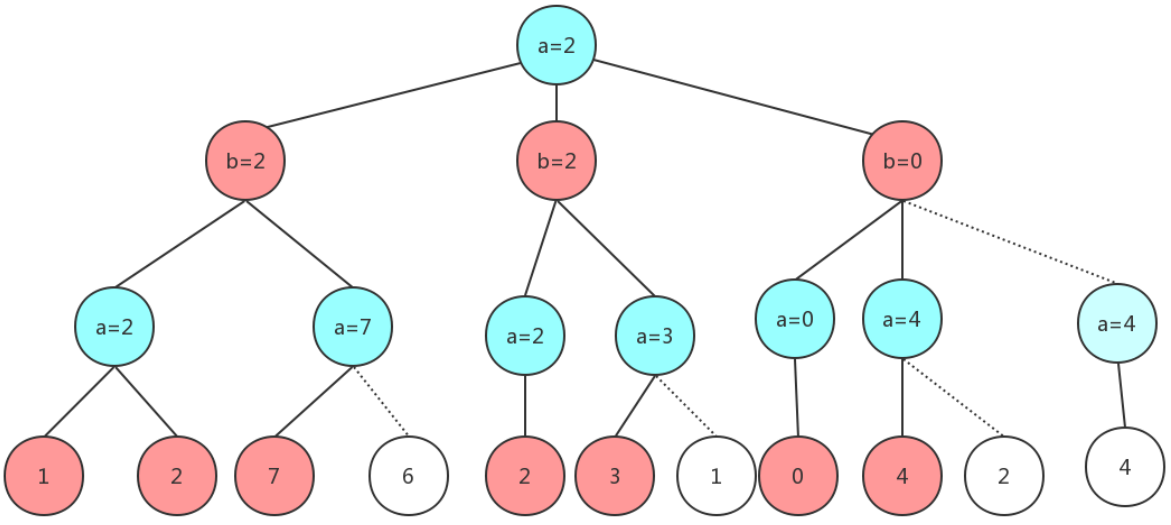
如下图所示，对上述博弈树的叶子结点进行赋值，再从底往上更新。每个结点的α值默认为-∞，β值默认为∞，若是 max 结点，则根据子节点更新自己的α值，若是 min 结点，则根据子节点更新自己的 beta 值。

如果存在：

当前结点如果是 max 结点，则：如果当前结点的α值大于其某个祖先结点的β值，则对该节点不进行继续搜索。

当前结点如果是 min 结点，则：如果当前节点的β值小于其某个祖先结点的α值，则对该节点不进行继续搜索。

下图是剪枝示例：



对博弈树进行 alphabeta 剪枝之后，可以省去很多情况的搜索，在很大程度上提升建树的效率。

3、博弈树搜索在本实验中的应用

①黑白棋的规则

黑白棋又称为翻转棋。棋盘为 8*8 共 64 格。开局时，棋盘正中央的 4 格放置黑白相隔的 4 个棋子。双方轮流落子。当落子的点和棋盘上任意一枚己方棋子在一条线上（直线或对角线八个方向），并且两个棋子之间夹的都是对方的棋子，就能将对方的棋子转变为己方棋子。

如果在当前局势下，自己没有可以下子的地方，则继续由对方下子。

如果双方皆不能下子（双方无子可下，或棋盘已满，或一方棋子被另一方完全吃掉），则计算棋子个数，棋子个数多的一方胜。

②人机对战实现机理

人机对战实际上就是电脑根据人下子的情况，思考接下来几步棋，作出一个较好的走步。所以每一次人下子之后，电脑根据当前的下子情况建立博弈树（层数越多代表思考越多，即电脑棋艺会越高），并根据博弈树选择的较优结果进行下一步下子。具体的流程请见后面的流程图表示。

③估价函数的设定

博弈树的叶节点需要进行局面评估，从而可以更新整棵树的α和β值。黑白棋的局面估价其实是比较困难的，甚至有时候最后一步棋才能判别出优劣。本次实验我采取的估价函数是结合行动力和各位权值分别相加。

各位权值来源是我查找到的黑白棋各个位置根据重要性分别的权值表。我们知道，黑白棋和围棋有点类似，都是“金角银边草肚皮”的规则，在黑白棋中占据四个角是很重要的。

具体的权值表如下:

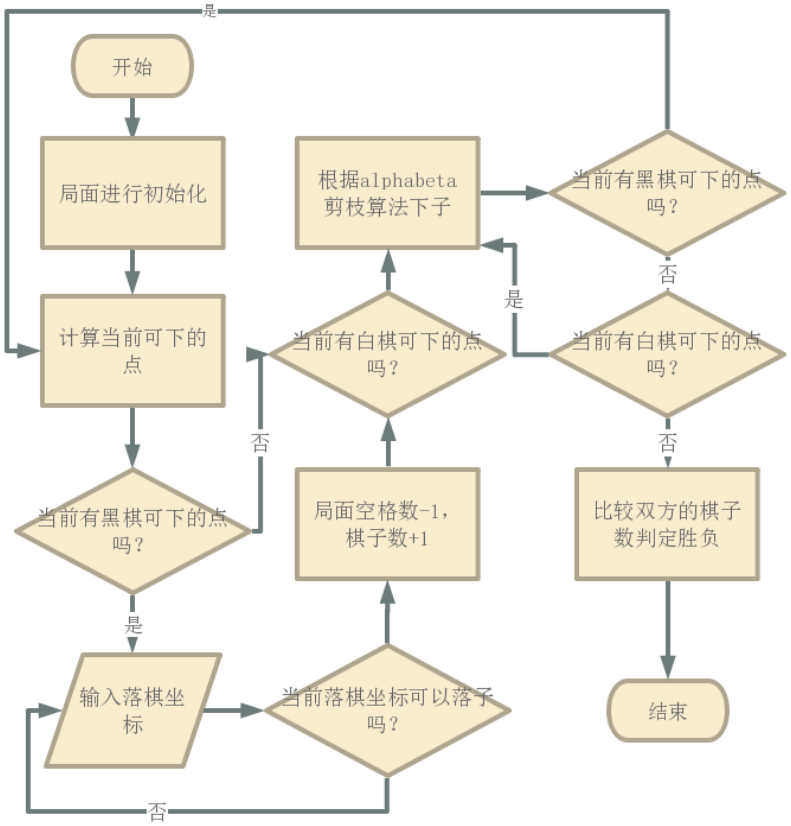
500	-25	10	5	5	10	-25	500
-25	-45	1	1	1	1	-45	-25
10	1	3	2	2	3	1	10
5	1	2	1	1	2	1	5
5	1	2	1	1	2	1	5
10	1	3	2	2	3	1	10
-25	-45	1	1	1	1	-45	-25
500	-25	10	5	5	10	-25	500

可以看到，黑白棋的四个角的权值很高，肚皮权值较低。但是围绕着 4 个角的 12 个子的权值为负。

局面的行动力是指各方在当前局面可以下子的位置，如果当前局面我方下子的位置多于对方下子的位置，那么这个局面我是较优的。我根据行动力的优劣性来进行+50 或-50，并于权重表的值进行加权运算，得到最终的估价指标。

流程图/伪代码

1、人机对战流程图



2、博弈树 alphabeta 剪枝伪代码

```
function alphabeta(node,depth,alpha,beta,maxormin)
    if depth==0 or node have no child
        return value of node
    if maxnode
        val:=-10000//近似为负无穷
        for each child of node
            if(alphabeta(child, depth - 1, alpha, beta, !maxormin)>val)//更大则更新更优结点
                val:=alphabeta(child, depth - 1, alpha, beta, !maxormin)
                bestchild of node:=currentnode
            alpha:=max(alpha,val)
            if alpha>=beta break//alpha 剪枝
        return val
    else
        val:=10000//近似为正无穷
        for each child of node
            if(alphabeta(child, depth - 1, alpha, beta, !maxormin)<val)//更小则更新更优结点
                val:=alphabeta(child, depth - 1, alpha, beta, !maxormin)
                bestchild of node:=currentnode
            beta:=min(alpha,val)
            if alpha>=beta break//beta 剪枝
        return val
```

关键代码

1、结点设计

设置结点的结构如下，包括当前棋局的状态和进行αβ剪枝必备的子节点向量、αβ值。另外，spacenum 和 piecenum 方便计算估价和计算棋子数，best 为当前状态最优的子节点的下标。

```
struct node {
    char state[9][9];
    vector <node> children;
    int alpha;
    int beta;
    int value;
    int spacenum;//空格数量
    int piecenum;//棋子数量（假设为黑棋）
    int best;
```

```
};
```

2、下棋过程

在黑子、白子下棋之前，都要先判断有没有可以下子的位置。每一方下子同时都要更新结点的相关参数。如果下一步有一方无法下子，则另一方可以连续下。若双方都无法下子，则计算棋子数目判定胜负。

```
while (true) {
    root.parent = NULL;
    vector <string> co;
    co = findthecorrect(root.state, 0);        //轮到人下
    if (co.size() != 0) {   //当前有下子的位置
        for (int k = 0; k < co.size(); k++)
            root.state[co[k][0] - '0'][co[k][1] - '0'] = '.';
        printstate(root);
        cout << "当前比分:
  " << root.piecenum << ':' << 64 - root.spacenum - root.piecenum << endl;
        cout << endl << "标记.的坐标可以落子，请输入落子的坐标，如'A2'：";

        string input;
        cin >> input;
        while (input.size() > 2) {
            cout << "请重新输入！坐标间不要有空格及其它无关字符：";
            cin >> input;
        }
        int tmpi = input[0] - 'A' + 1, tmpj = input[1] - '0';
        while (root.state[tmpi][tmpj] != '.') {
            cout << "该点不可下！请重新输入：";
            cin >> input;
            tmpi = input[0] - 'A' + 1;
            tmpj = input[1] - '0';
        }

        for (int k = 0; k < co.size(); k++)
            root.state[co[k][0] - '0'][co[k][1] - '0'] = ' ';
        int num = capture(root.state, tmpi, tmpj, 0);             //吃子
        root.state[tmpi][tmpj] = '*';
        root.spacenum--;
        root.piecenum += num + 1;
        cout << "下子结果：" << endl;
        printstate(root);
        cout << "当前比分:
  " << root.piecenum << ':' << 64 - root.spacenum - root.piecenum << endl;

    }
    else {  //没有下子的位置，电脑继续下子
        printstate(root);
```

```
39.         cout << "当前比分：
   " << root.piecenum << ':' << 64 - root.spacenum - root.piecenum << endl;
40.     }
41.     if (root.spacenum == 0 || (findthecorrect(root.state, 1).size() == 0 &
   & findthecorrect(root.state, 0).size() == 0)) {//黑子下了最后一个子
42.         int blackpiecenum = root.piecenum, whitepiecenum = 64 - root.piece
   num;
43.         if (blackpiecenum > whitepiecenum) {
44.             cout << "你赢了！最终比分：
   " << blackpiecenum << ':' << whitepiecenum << endl;
45.             system("pause");
46.             break;
47.         }
48.         else if (blackpiecenum == whitepiecenum) {
49.             cout << "平局！最终比分：
   " << blackpiecenum << ':' << whitepiecenum << endl;
50.             system("pause");
51.             break;
52.         }
53.         else {
54.             cout << "你输了！最终比分：
   " << blackpiecenum << ':' << whitepiecenum << endl;
55.             system("pause");
56.             break;
57.         }
58.     }
59.     if (findthecorrect(root.state, 1).size() != 0) {  //电脑有可以下子的位
   置
60.         cout << endl << "电脑正在思考……" << endl;
61.         double starttime = clock();
62.         int value=ai(root, 0, 5, 1);      //建树寻找较好的下法并下子
63.         double endtime = clock();
64.         cout << endl << "计算用时：
   " << (endtime - starttime) / 1000 << endl;
65.         node newnode;
66.         for (int i = 0; i < 9; i++) {
67.             for (int j = 0; j < 9; j++)
68.                 newnode.state[i][j] = root.children[root.best].state[i][j]
   ;
69.         }
70.         newnode.alpha = root.children[root.best].alpha;
71.         newnode.beta = root.children[root.best].beta;
72.         newnode.spacenum = root.children[root.best].spacenum;
73.         newnode.piecenum = root.children[root.best].piecenum;
74.         root = newnode;  //将当前局面更新为下子之后的局面
75.         root.children.clear();
76.     }
```

3、可下子位置判断（以上方向为例）

可下子位置的判断是比较简单的。如果该坐标为空格，则对八个方向进行搜索，如果有哪个方向上有己方的子，而且两个子之间所夹都是对方的子，则可以下。此处代码偏长，不完全贴上，仅以上方向为例，其余 7 个方向以此类推。

```
vector <int> judge1(int i, int j, char state[][9], int peopleorcomputer) {
    vector<int> tmpflag;
    for (int k = 0; k < 4; k++) {
        tmpflag.push_back(0);
    }
    int plusi = 0, plusj = 0, minusi = 0, minusj = 0;

    //上
    minusi--;
    if (i + minusi >= 1 && state[i + minusi][j] != ' ') {
        if (peopleorcomputer == 0) {
            if (state[i + minusi][j] == '+') {
                while (i + minusi >= 1) {
                    minusi--;
                    if (state[i + minusi][j] == ' ') break;
                    if (state[i + minusi][j] == '+') continue;
                    if (state[i + minusi][j] == '*') {
                        tmpflag[0] = 1;
                        break;
                    }
                }
            }
        }
        else {
            if (state[i + minusi][j] == '*') {
                while (i + minusi >= 1) {
                    minusi--;
                    if (state[i + minusi][j] == ' ') break;
                    if (state[i + minusi][j] == '*') continue;
                    if (state[i + minusi][j] == '+') {
                        tmpflag[0] = 1;
                        break;
                    }
                }
            }
        }
    }
```

4、寻找可走位置

有了上面的判断是否可下的函数，寻找所有当前可走的位置就容易得多。

```
vector <string> findthecorrect(char state[][9], int peopleorcomputer) {
    vector <string> position;
```

```
    for (int i = 1; i < 9; i++) {
        for (int j = 1; j < 9; j++) {
            if (state[i][j] != ' ')  continue;//该点若是空格才可以下子
            vector <int> flag1, flag2;
            flag1 = judge1(i, j, state, peopleorcomputer);//上下左右判定
            flag2 = judge2(i, j, state, peopleorcomputer);//对角线判定
            int flag11 = 0, flag22 = 0;
            for (int k = 0; k < flag1.size(); ++k)
                if (flag1[k] == 1) {
                    flag11 = 1;
                    break;
                }
            for (int k = 0; k < flag2.size(); ++k)
                if (flag2[k] == 1) {
                    flag22 = 1;
                    break;
                }
            if (flag11 == 1 || flag22 == 1) {
                char tmpa = char('0' + i);
                char tmpb = char('0' + j);
                string tmpc = "";
                tmpc += tmpa;
                tmpc += tmpb;
                position.push_back(tmpc);//坐标可行，则添加
            }

        }
    }
    return position;//返回坐标的向量
}
```

5、AI 部分——alphabeta 剪枝。

每个结点的α值默认为-∞，β值默认为∞，若是 max 结点，则根据子节点更新自己的α值，若是 min 结点，则根据子节点更新自己的 beta 值。当前结点如果是 max 结点，则：如果当前结点的α值大于其某个祖先结点的β值，则对该节点不进行继续搜索。当前结点如果是 min 结点，则：如果当前节点的β值小于其某个祖先结点的α值，则对该节点不进行继续搜索。

```
//ai 部分，alphabeta 剪枝
int ai(node &root, int k, int kmax, int peopleorcomputer) {//建立博弈树
    vector <string> childposition = findthecorrect(root.state, peopleorcomputer);//可以下子的点的坐标向量
    if (childposition.size() == 0 || k>kmax)
        return calcue(root.state, peopleorcomputer);//没有可下的结点再往下发展，则计算当前局面的评估
```

```
6.
7.      if (peopleorcomputer == 1) {//当前结点为 max 结点
8.          int maxscore = -10000;
9.          for (int i = 0; i < childposition.size(); i++) {
10.             node child;
11.             for (int k = 0; k < 9; k++) {
12.                 for (int j = 0; j < 9; j++)
13.                     child.state[k][j] = root.state[k][j];
14.             }
15.             int tmpi = childposition[i][0] - '0', tmpj = childposition[i][
    1] - '0';
16.             child.spacenum = root.spacenum - 1;
17.             child.alpha = root.alpha;
18.             child.beta = root.beta;
19.             child.parent = &root;
20.             if (peopleorcomputer == 1) {
21.                 child.state[tmpi][tmpj] = '+';
22.                 child.piecenum = root.piecenum - capture(child.state, tmpi
    , tmpj, peopleorcomputer);
23.             }
24.             else {
25.                 child.state[tmpi][tmpj] = '*';
26.                 child.piecenum = root.piecenum + capture(child.state, tmpi
    , tmpj, peopleorcomputer) + 1;
27.             }
28.             root.children.push_back(child);//子节点添加到根节点的孩子中
29.             int childscore = ai(root.children[i], ++k, kmax, peopleorcompu
    ter ^ 1);//对当前结点进行递归拓展
30.             if (childscore > maxscore) {//若当前值更大，则更新
31.                 maxscore = childscore;
32.                 root.alpha = maxscore;
33.                 root.best = i;
34.             }
35.             if (root.alpha >= root.beta)//若 alpha 大于等于 beta 值，break 则代
    表不继续搜索子节点下去
36.                 break;
37.         }
38.         return maxscore;
39.     }
40.     else {//当前结点为 min 结点
41.         int minscore = 10000;
42.         for (int i = 0; i < childposition.size(); i++) {
43.             node child;
44.             for (int k = 0; k < 9; k++) {
45.                 for (int j = 0; j < 9; j++)
46.                     child.state[k][j] = root.state[k][j];
47.             }
48.             int tmpi = childposition[i][0] - '0', tmpj = childposition[i][
    1] - '0';
49.             child.spacenum = root.spacenum - 1;
50.             child.alpha = root.alpha;
51.             child.beta = root.beta;
52.             child.parent = &root;
53.             child.state[tmpi][tmpj] = '*';
```

```
            child.piecenum = root.piecenum + capture(child.state, tmpi, tmpj, peopleorcomputer) + 1;
            root.children.push_back(child);//子节点添加到根节点的孩子中
            int childscore = ai(root.children[i], ++k, kmax, peopleorcomputer ^ 1);//对当前结点进行递归拓展
            if (childscore < minscore) {//若当前值更小，则更新
                minscore = childscore;
                root.alpha = minscore;
                root.best = i;
            }
            if (root.alpha >= root.beta)//若 alpha 大于等于 beta 值，break 则代表不继续搜索子节点下去
                break;
        }
        return minscore;
    }

}
```

6、局面评估

根据坐标权值+行动力加权计算对当前局面进行评估。

```
//计算局面评估，权值+行动力加权相加
int calcue(char state[][9] , int peopleorcomputer) {
    int result=0;
    if (peopleorcomputer == 1) {
        for (int i = 1; i < 9; i++) {
            for (int j = 1; j < 9; j++) {
                if (state[i][j] == '+') {
                    result += weight[i - 1][j - 1];

                }
            }
        }
        result /= 2;//权值计算
        vector <string> flag1 = findthecorrect(state, 1);
        vector <string> flag2 = findthecorrect(state, 0);
        if (flag1.size() > flag2.size()) result += 50; else result -= 50;//行动力计算

    }
    else {
        for (int i = 1; i < 9; i++) {
            for (int j = 1; j < 9; j++) {
                if (state[i][j] == '*') {
                    result += weight[i - 1][j - 1];
                }
            }
        }
        vector <string> flag1 = findthecorrect(state, 0);
```

```
28.         vector <string> flag2 = findthecorrect(state, 1);
29.         if (flag1.size() > flag2.size()) result += 50; else result -
   = 50;
30.     }
31.     return result;
32.}
```

实验结果及分析

实验结果展示

首先我在 alphabeta 剪枝之前，先用单纯的 minmax 算法写了一遍黑白棋，多次实验后发现了时间上的实验差异：

博弈树层数	minmax 平均时间	alphabeta 剪枝平均时间
3	0.5s	0.02s
4	2.9s	0.32s

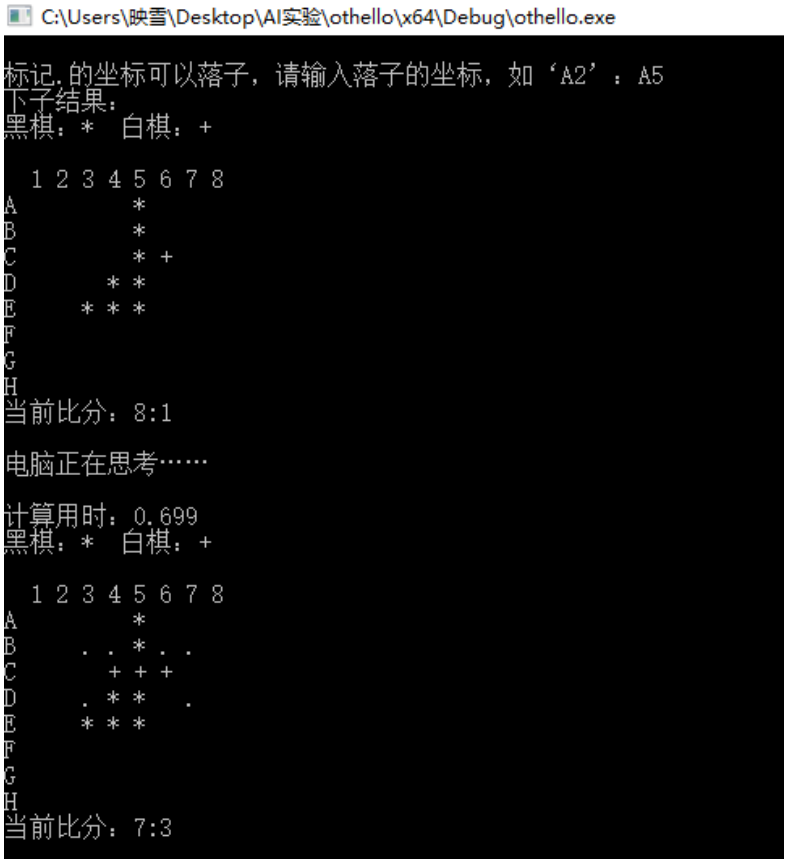
实验中证明了 alphabeta 剪枝的重要性，如果不剪枝，在 4 层时就需要花上几秒的时间，更不用说更深度地建树了。

接下来是游戏步骤展示。游戏初始界面可以选择难度，简单、中等和困难取决于每次电脑思考时博弈树建立的深度。展示中，我选择了中等难度（此处博弈树为 5 层）。选择难度后会显示游戏初始界面。（已经修改到标准的 8*8 棋盘）

……此处是下了几步的省略线……

选取棋局中的三步展示：

①这一步我成功下到 8：1 小虐了一下白子，但是白子下子之后把差距拉回了一些，拉到了 7:3。



②接下来我发现白棋在逐渐把差距拉小，最后拉到了 7:5：

(图片见下页)

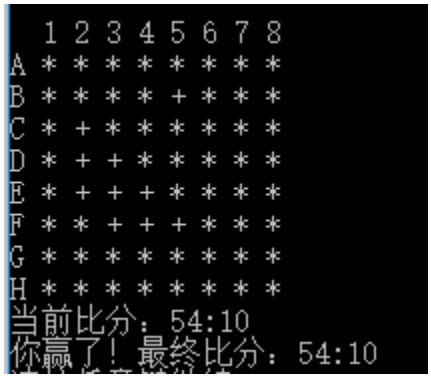
```
标记,的坐标可以落子，请输入落子的坐标，如‘A2’：B6
下子结果：
黑棋：*  白棋：+

  1 2 3 4 5 6 7 8
A         *
B         * *
C       + * +
D       * *
E     * * *
F
G
H
当前比分：9:2

电脑正在思考……

计算用时：1.151
黑棋：*  白棋：+

  1 2 3 4 5 6 7 8
A         * + .
B     . . + + .
C     . + * + .
D       * *
E     * * *
F
G
H
当前比分：7:5
```

③然后下一步我就被白棋反超了：

```
标记,的坐标可以落子，请输入落子的坐标，如‘A2’：B3
下子结果：
黑棋：*  白棋：+

  1 2 3 4 5 6 7 8
A         * +
B     *   + +
C       * * +
D       * *
E     * * *
F
G
H
当前比分：9:4

电脑正在思考……

计算用时：1.064
黑棋：*  白棋：+

  1 2 3 4 5 6 7 8
A         * + .
B     *   + + .
C       * + + .
D       * + .
E     * * + .
F         + .
G         . .
H
当前比分：6:8
```

终局显示：

之前展示中的结局我赢了的中等难度：



后来我与高等难度下，被虐了……（四个对角线被电脑占了 3 个，我很大的概率会输了）