# lab5：博弈树搜索

| 姓名 | 学号 | 专业 | 教学班级 |
| --- | --- | --- | --- |
| 张航悦 | 19335262 | 计算机科学与技术 | 19计科2班 |



# 一、 算法原理

## 1.1 博弈树

两玩家零和博弈问题中，玩家轮流行动，进行博弈使自己的优势最大。两玩家零和博弈具有确定性、信息完备性、零和性三个特点。而博弈树则是基于两玩家零和博弈问题，抽象表示其博弈的过程。

由于信息完备性和确定性，可以用博弈树的每个节点表示一个确定的状态，在行动后拓展的节点为子节点。两个玩家轮流拓展节点。对于每个节点，我们利用评价函数对当前节点的优劣进行评分。博弈树搜索的目的即找出对双方都是最优的子节点的值。

## 1.2 Minimax搜索

由于博弈树的零和性，两玩家的利益关系对立，一方通过行动使得自己的评价函数尽可能大，而另一方则让对手的评价函数尽可能小。由于玩家是交替行动，因此在树的每一层让一方的评价函数在取最大最小值间交替进行。

Minimax搜索找到内部节点的值，其中Max节点的每一步扩展要使收益最大，Min节点的扩展要使收益最小。

在本实验中，**将AI设为Max节点，人设为Min节点。**

## 1.3 Alpha-beta剪枝

随着博弈的进行，若暴力搜索所有的游戏状态将相当耗费内存资源和时间，效率十分低下。因此引入Alpha-beta剪枝，剪掉不可能影响对应决策的分支，尽可能地消除部分搜索树。

具体剪枝方法：

- Max节点记录$\alpha$值，Min节点记录$\beta$值
- 对于Max节点的剪枝：若当前效益值$\geq$任何祖先Min节点的$\beta$值，则进行剪枝。

  因为此时Max节点估值一定会大于某一祖先Min节点的估值上界，而祖先节点是Min节点，是必然不会选择当前节点的。因此所有的子节点可以停止拓展，从而实现了剪枝。
- 对于Min节点的剪枝：若当前效益值$\leq$任何祖先Max节点的$\alpha$值，则进行剪枝。

  因为此时Min节点估值一定会小于某一祖先Max节点的估值下界，而祖先节点是Max节点，是必然不会选择当前节点的。因此所有的子节点可以停止拓展，从而实现了剪枝。

## 1.4 评价函数设定

- 棋盘位置

黑白棋和围棋一样, 也遵守着"**金角银边烂肚皮**"的定律, 四个角的地势值非常大, 其次是四条边。因此我们再给8*8地图点分配地势值时, 大体满足角边重, 中腹轻的模式。棋盘的各位置的权重如下设置。

```
WEIGHT = [[120, -20, 20, 5, 5, 20, -20, 120],
          [-20, -40, -5, -5, -5, -5, -40, -20],
          [20, -5, 15, 3, 3, 15, -5, 20],
          [5, -5, 3, 3, 3, 3, -5, 5],
          [5, -5, 3, 3, 3, 3, -5, 5],
          [20, -5, 15, 3, 3, 15, -5, 20],
          [-20, -40, -5, -5, -5, -5, -40, -20],
          [120, -20, 20, 5, 5, 20, -20, 120]]
```

- 行动力

在某局面中, 选择多, 则灵活主动。而选择少, 则往往陷入被动。因此可以落子的位置个数, 就成为了评估局面好坏的参考因素了。将可落子的位置数目称为行动力。

- 稳定子

稳定子是指无论如何, 都不可能被翻覆的子。稳定值越多, 未来获胜的几率就越大。

- 评价函数

评价函数即上面三个因素的线性组合，赋予每个因素不同的权重。在这里我希望AI能更灵活，主动出击，所以将行动力设置了较高的权重。

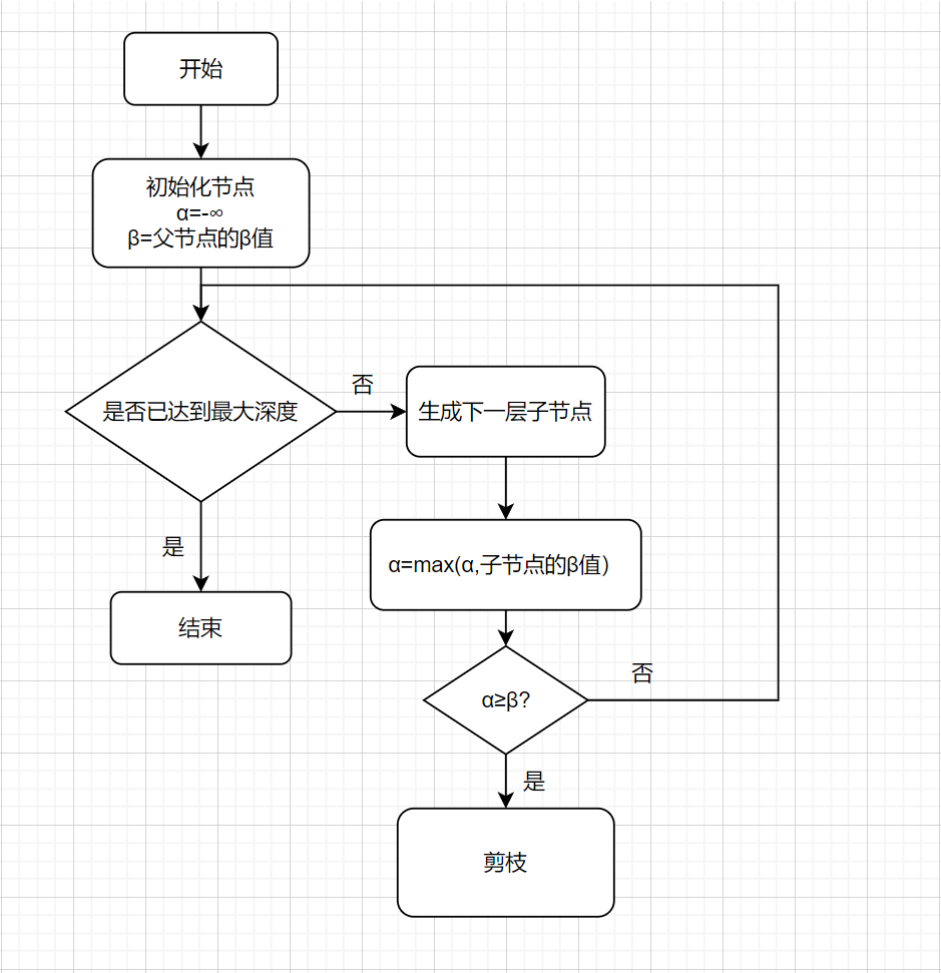
评价函数公式为

$$\text{分数} = \text{棋盘位置权重} + 2 \times \text{稳定子} + 5 \times \text{行动子}$$

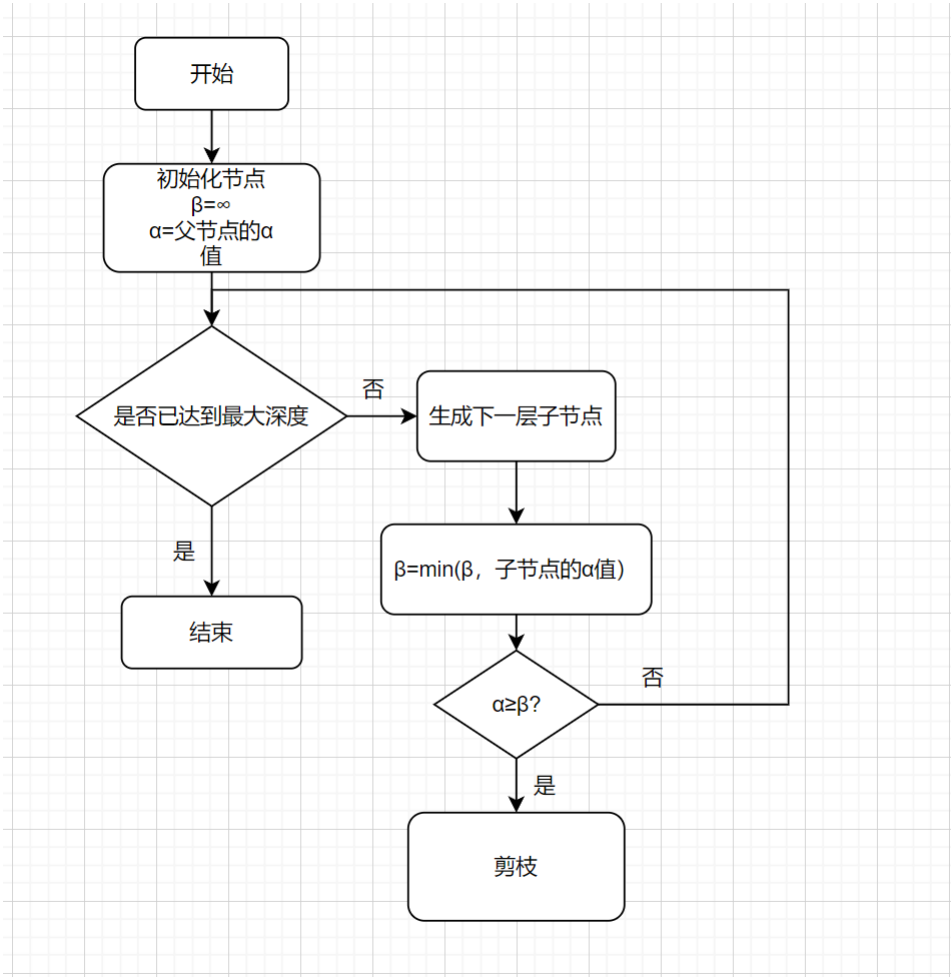
# 二、 流程图和伪代码

- Minimax搜索

生成Max节点过程如下图所示




生成Min节点过程如下图所示



可以看到Max节点和Min节点的生成有相当高的对称性，因此在这里我们可以使用同一个递归函数实现Max和Min节点的生成和剪枝。

**Algorithm 1:** Alpha-beta 搜索

**Input:** state: 当前状态，depth: 最大搜索深度，*last_alpha*: 父节点的 alpha 值，*last_beta*: 父节点的 beta 值，color: 当前下棋方的颜色

**Output:** alpha: 当前节点的 alpha 值，beta: 当前节点的 beta 值，pos: 当前落子的位置

```
//若已到达最大深度则直接返回评价函数值
if depth==0 then
    return score(state,color),score(state,color),(-1,-1)
end
//根据节点类型初始化节点
alpha = -inf
beta= inf
if type == Max then
    beta= last_beta
end
else alpha = last_alpha
//遍历所有可以落子操作
for eachpos is valid do
    //根据当前落子调整棋局状态
    nextstate=flipped(state,eachpos)
    //递归遍历子节点
    alpha,beta,_=Alphabeta(newstate, depth −
     1,oppo_color, alpha, beta)
    //根据节点类型更新 alpha 和 beta 值
    if type == Max and alpha < next_alpha then
        pos=eachpos
        alpha = next_alpha
    end
    if type == Min and beta > next_beta then
        act = eachpos
        beta = next_beta
    end
    //判断是否需要剪枝
    if alpha>beta then
        return alpha, beta,pos
    end
endfor
return alpha,beta,pos
```

# 三、 关键代码展示

- 获取可落子位置

```python
def validpos(chess, color):
    # 返回当前可以下棋的位置
```

```
    poslist = {}  # 用来记录是否有可以落子的位置 并记录其中可以翻转的棋子的坐标
    oppo_color = 1 - color

    for startx in range(8):
        for starty in range(8):
            # 若该位置非空 直接跳过
            if chess.board[startx][starty] != -1:
                continue
            # 遍历周围位置
            for ix, iy in move:
                x = startx + ix
                y = starty + iy
                while chess.onboard(x, y) and chess.board[x][y] ==
oppo_color:
                    x += ix
                    y += iy
                    # 将夹在自己棋子和对手棋子中间的棋子记录下来
                    if chess.onboard(x, y) and chess.board[x][y] == color:
                        if (startx, starty) not in poslist:
                            poslist[(startx, starty)] = [(startx, starty)]
                        # 存下中间可以翻转的棋子的坐标
                        if ix != 0:
                            m = (x - startx) * ix
                        else:
                            m = (y - starty) * iy
                        for i in range(1, m):
                            poslist[(startx, starty)].append((startx + i *
ix, starty + i * iy))
                        break

                    elif chess.onboard(x, y) and chess.board[x][y] == -1:
                        break

    return len(poslist), poslist
```

- 评价函数

```
def evaluate(chess, color):
    # 估值函数 为位置权重和稳定动点数
    self_wei = 0
    oppo_wei = 0
    self_val = 0
    oppo_val = 0
    oppo_color = 1 - color
    for i in range(8):
        for j in range(8):
            if chess.board[i][j] == color:
                self_wei += WEIGHT[i][j]
            elif chess.board[i][j] == oppo_color:
                oppo_wei += WEIGHT[i][j]
    # 可落子的数目
    temp_self_val, list = validpos(chess, color)
    temp_oppo_val, list = validpos(chess, oppo_color)
    self_val += temp_self_val
    oppo_val += temp_oppo_val
```

```
    # 稳定点数
    steady_self = steadypoint(chess, color)
    steady_oppo = steadypoint(chess, oppo_color)

    return (self_wei - oppo_wei) + 2 * (steady_self - steady_oppo) + 5 *
(self_val - oppo_val)
```

- 博弈树搜索函数

```
def alphabeta(chess, depth, last_alpha, last_beta, color):
    # 叶节点直接返回即可
    if depth == 0:
        score = evaluate(chess, ai_color)
        return score, score, (-1, -1)
    # 初始化节点 alpha beta记录该节点的a和b值
    if color == ai_color:
        alpha = float('-inf')
        beta = last_beta
    else:
        alpha = last_alpha
        beta = float('inf')

    posnum, posdict = validpos(chess, color)
    if posnum == 0:
        color = 1 - color
        posnum, posdict = validpos(chess, color)

        if posnum == 0:  # 棋局结束
            checkwin(chess)
        else:
            return alphabeta(chess, depth - 1, alpha, beta, color)

    validmove = posdict.keys()
    for move in validmove:
        for flipped in posdict[move]:
            chess.board[flipped[0]][flipped[1]] = color
        # 递归遍历
        next_alpha, next_beta, _ = alphabeta(chess, depth - 1, alpha, beta,
1 - color)

        # 撤销刚刚的操作 翻回颜色 取消落子
        for flipped in posdict[move]:
            chess.board[flipped[0]][flipped[1]] = 1 - color
        chess.board[move[0]][move[1]] = -1

        if color == ai_color and alpha < next_beta:
            alpha = next_beta
            pos = (move[0], move[1])
        if color == player_color and beta > next_alpha:
            beta = next_alpha
            pos = (move[0], move[1])
        if beta <= alpha:
            return alpha, beta, pos
    return alpha, beta, pos
```

```
45
```

# 四、 实验结果分析

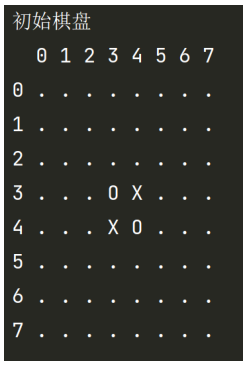
## 4.1 实验结果展示

其中'X'符号表示为黑棋，‘O’符号表示为白棋，‘·’符号表示为空闲。

得分以评价函数值得分作为评判标准。

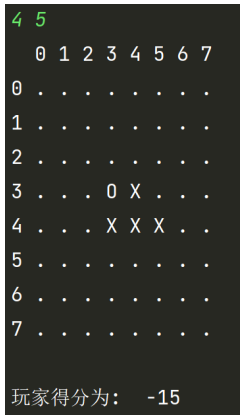
由于全部展示实验结果截图过于繁多，**在此只展示深度为3时，人先手和电脑先手的结果截图。**

- 初始棋盘




### 1. 深度为3，人先手

- **第一回合**
- 人 落子位置:(4,5)




- 电脑 落子位置:(3,5)

```
电脑落子
电脑落子为:  (3, 5)
电脑得分为:  0
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . . . . . .
3 . . . O O O . .
4 . . . X X X . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .
```

- 第一回合：人VS电脑=-15:0

- **第二回合**

- 人 落子位置:(2,3)

```
2 3
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . X . . . .
3 . . . X X O . .
4 . . . X X X . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .

玩家得分为:  -23
```

- 电脑 落子位置:(5,5)

```
电脑落子
电脑落子为:  (5, 5)
电脑得分为:  10
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . X . . . .
3 . . . X X O . .
4 . . . X X O . .
5 . . . . . O . .
6 . . . . . . . .
7 . . . . . . . .
```

- 第二回合：人VS电脑=-23:10

- **第三回合**

- 人 落子位置:(5,6)

```
5 6
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . X . . . .
3 . . . X X O . .
4 . . . X X X . .
5 . . . . . O X .
6 . . . . . . . .
7 . . . . . . . .

玩家得分为:  -15
```

- 电脑 落子位置:(5,7)

```
电脑落子
电脑落子为:  (5, 7)
电脑得分为:  5
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . X . . . .
3 . . . X X O . .
4 . . . X X X . .
5 . . . . . O O O
6 . . . . . . . .
7 . . . . . . . .
```

- 第三回合：人VS电脑=-15:5

- **第四回合**
- 人 落子位置:(2,5)

```
2 5
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . X . X . .
3 . . . X X X . .
4 . . . X X X . .
5 . . . . . O O O
6 . . . . . . . .
7 . . . . . . . .

玩家得分为:  -6
```

- 电脑 落子位置:(1,5)

```
电脑落子
电脑落子为:  (1, 5)
电脑得分为:  43
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . O . .
2 . . . X . O . .
3 . . . X X O . .
4 . . . X X O . .
5 . . . . . O O O
6 . . . . . . . .
7 . . . . . . . .
```

- 第四回合：人VS电脑=-6:43

- **第五回合**
- 人 落子位置:(6,7)

```
6 7
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . O . .
2 . . . X . O . .
3 . . . X X O . .
4 . . . X X X . .
5 . . . . . O X O
6 . . . . . . . X
7 . . . . . . . .

玩家得分为:  30
```

- 电脑 落子位置:(7,7)

```
电脑落子
电脑落子为:  (7, 7)
电脑得分为:  129
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . O . .
2 . . . X . O . .
3 . . . X X O . .
4 . . . X X X . .
5 . . . . . O X O
6 . . . . . . . O
7 . . . . . . . O
```

- 第五回合：人VS电脑=30:129

## 2 深度为3，电脑先手

- **第一回合**
- 电脑 落子位置(2,3)

```
电脑落子
电脑落子为:  (2, 3)
电脑得分为:  15
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . X . . . .
3 . . . X X . . .
4 . . . X O . . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .
```

- 人 落子位置(2,2)

```
2 2
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . O X . . . .
3 . . . O X . . .
4 . . . X O . . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .

玩家得分为:  -21
```

- 第一回合：人VS电脑=-21:15

- **第二回合**

- 电脑 落子位置(3,2)

```
电脑落子
电脑落子为:  (3, 2)
电脑得分为:  27
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . O X . . . .
3 . . X X X . . .
4 . . . X O . . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .
```

- 人 落子位置(2,4)

```
2 4
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . O O O . . .
3 . . X X O . . .
4 . . . X O . . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .

玩家得分为:  -2
```

- 第二回合：人VS电脑=-2:27

- **第三回合**

- 电脑 落子位置(1,2)

```
电脑落子
电脑落子为:  (1, 2)
电脑得分为:  2
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . X . . . . .
2 . . X O O . . .
3 . . X X O . . .
4 . . . X O . . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .
```

- 人 落子位置(0,1)

```
0 1
  0 1 2 3 4 5 6 7
0 . O . . . . . .
1 . . O . . . . .
2 . . X O O . . .
3 . . X X O . . .
4 . . . X O . . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .

玩家得分为:  32
```

- 第三回合：人VS电脑=32:2

- **第四回合**
- 电脑 落子位置(2,5)

```
电脑落子
电脑落子为:  (2, 5)
电脑得分为:  70
  0 1 2 3 4 5 6 7
0 . O . . . . . .
1 . . O . . . . .
2 . . X X X X . .
3 . . X X X . . .
4 . . . X O . . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .
```

- 人 落子位置(4,5)

```
4 5
  0 1 2 3 4 5 6 7
0 . O . . . . . .
1 . . O . . . . .
2 . . X O X X . .
3 . . X X O . . .
4 . . . X O O . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .

玩家得分为:  26
```

- 第四回合：人VS电脑=26:70

- **第五回合**
- 电脑 落子位置(5,4)

```
电脑落子
电脑落子为:  (5, 4)
电脑得分为:  70
  0 1 2 3 4 5 6 7
0 . O . . . . . .
1 . . O . . . . .
2 . . X O X X . .
3 . . X X X . . .
4 . . . X X O . .
5 . . . . X . . .
6 . . . . . . . .
7 . . . . . . . .
```

- 人 落子位置(6,3)

```
6 3
  0 1 2 3 4 5 6 7
0 . O . . . . . .
1 . . O . . . . .
2 . . X O X X . .
3 . . X X X . . .
4 . . . X X O . .
5 . . . . O . . .
6 . . . O . . . .
7 . . . . . . . .

玩家得分为:  62
```

- 第五回合：人VS电脑=62:70

## 4.2 实验结果分析

对深度为3，人先手和电脑先手；深度为2，人先手的结果进行分析。

| 用户得分：电脑得分 | 深度为3，人先手 | 深度为2，人先手 | 深度为3，电脑先手 |
| --- | --- | --- | --- |
| 第一回合 | -15:10 | -15:21 | -21:15 |
| 第二回合 | -23：10 | -7:22 | -2:27 |
| 第三回合 | -15:5 | 34:23 | 32:2 |
| 第四回合 | -6:43 | 15:35 | 26:70 |
| 第五回合 | 30:129 | 35:53 | 62:70 |

- 先后手对比

  对比第1，3列可以看到电脑先手时，可以很快的建立起优势，且人的优势大幅度下降。可以看出，先后手选择非常重要，奠定了整个棋盘的一个基调。

- 搜索深度对比

  对比第1，2列，反而看到深度为3时，电脑的优势建立反而比深度为2时慢。**在这里我认为是我的评估函数权重设置不够合理导致的，**由于行动力的权重过于大，导致电脑放弃了一些可以翻转更多的位置而选择了更灵活的位置，导致优势建立的较慢。通过对比可以看到当深度为3时，电脑一旦建立优势，将显著高于人，后续将持续处于领先地位。

后我对评价函数的权重进行了修改，改为了$分数 = 棋盘位置权重 + 2 \times 稳定子 + 5 \times 行动子$，再次重复实验查看结果。

可以看见修改评价函数后在深度为3时电脑相较于以前更快的建立起了优势。并且随着电脑思考的深度的增加，人的优势大幅减少。且经常在电脑落棋之后处于很大劣势，十分被动。

| **用户得分：电脑得分/人落子，电脑落子** | **深度为3，人先手** | **深度为2，人先手** |
| --- | --- | --- |
| 第一回合 | -15:18 / (4,5) (5,5) | -15:18 /(3,2) (2,2) |
| 第二回合 | 3：12 / (3,5) (5,6) | -13:10 / (1,2) (4,2) |
| 第三回合 | 9:25 / (5,6) (5,7) | -7:31 /(5,4) (0,2) |
| 第四回合 | 6:45 / (2,5) (1,5) | 14:34 /(3,1) (2,0) |
| 第五回合 | 57:151 / (6,7) (7,7) | 23:52 /(3,0) (4,0) |