李梓豪　作业 1　2021012088（已标书签）

● 简答题

1. 图搜索旨在解决树搜索中的空间复杂度问题，避免重复搜索相同状态。建立探索集，在扩张时不搜索已经被探索过的状态。

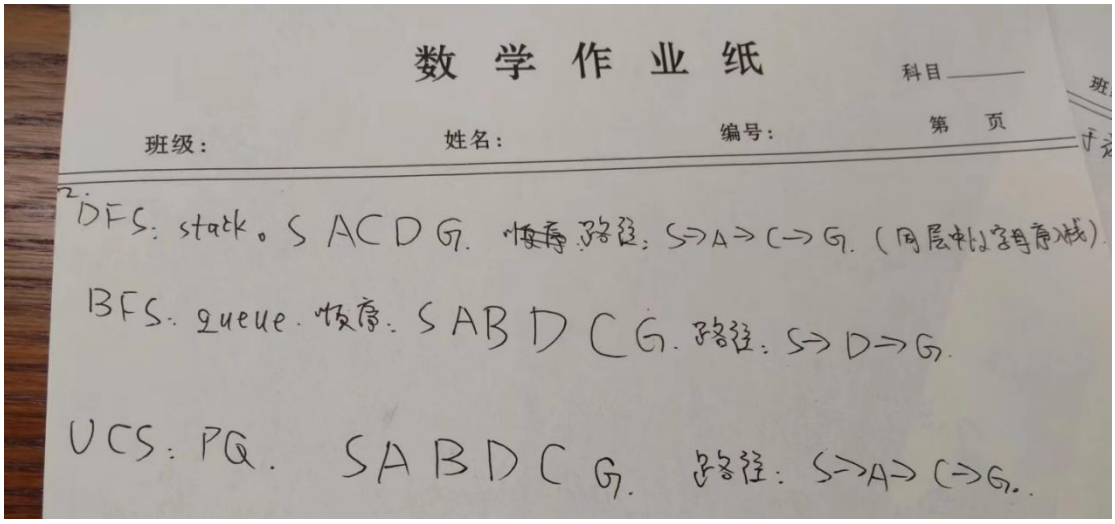
2. 宽度优先搜索（**BFS**）具有完备性和最短路径保证，但是可能需要存储大量节点，空间复杂度较高。深度优先搜索（**DFS**）空间复杂度较低，但不能保证找到最优解，容易陷入深度较大的分支。一致代价搜索（**UCS**）找到最小代价的解决方案，是一种最优搜索算法，但时空复杂度都较高。

3. 约束传播可以帮助减少待探索变量的值域，在探索到该节点时对后续搜索达到剪枝的效果，缩小搜索空间，提高搜索效率。**AC-3** 算法的时间复杂度为 $O(cd^3)$，其中 **c** 是变量之间的约束数（有向弧的数量），**d** 是变量的值域大小，因为对于每一个 **arc**，和尾节点的每一个值域中的值，进行强制弧相容需要 d^2（遍历首尾节点），而每次弧相容完进一次队，最多进 **d** 次（否则值域就没了）。

4. 集束搜索的基本思想是每层贪心地拓展 **k** 个“最好的”状态。通过保留一定数量的最优扩展节点，依次进行搜索和剪枝，直到找到最优解或达到搜索深度限制。该算法不能保证找到最优解，但可以在较短时间内找到接近最优解的解，因为局部的最优解并不一定是全局的最优（联合概率最大）的解。

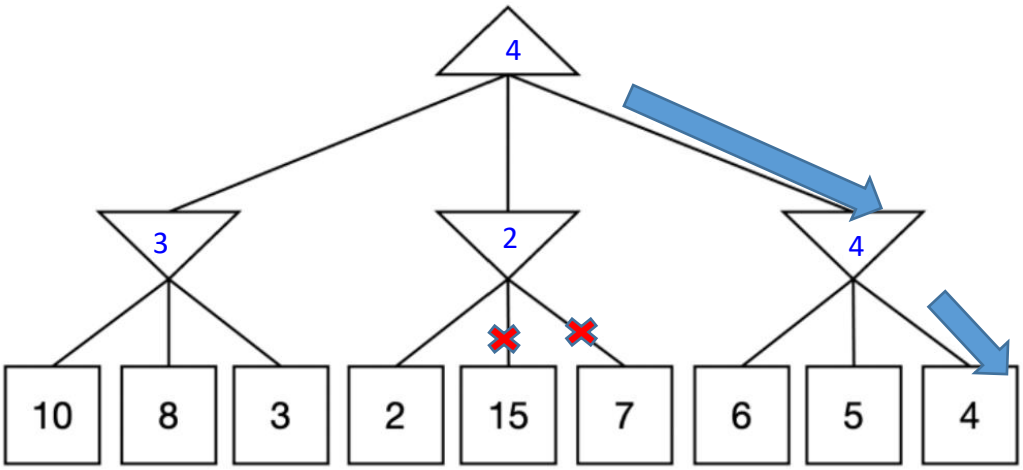
5. 蒙特卡洛搜索树（**MCTS**）算法构建搜索树的步骤包括选择、扩展、模拟和更新。该算法在资源有限的情况下可能会导致搜索树过于浅，难以找到最优解（比如围棋象棋等游戏在前几步需要一些人工经验来避免一开始就 **MCTS**）。

● 路径搜索



● Min-Max 搜索

1.2.（蓝箭头为第一问路径，红叉叉为第二问剪枝的）



● A* 算法的性质

1.

（a）考虑 T 即将取出边缘集时，最优路径上位于边缘集中的点 n*。若 n*=T，则 g(T)=h*(S)，显然成立；

若不等，则由于 h*(S)=g(n*)+h*(n*),而即将出来的是 T 不是 n*，则说明 f(n*)≥f(T)。则 g(n*)+h(n*)≥f(T)=g(T)，结合条件：h(n)≤h*(n)+C_1，得：

$$g(T)\leqslant g(n^*)+h^*(n^*)-(h^*(n^*)-h(n^*))\leqslant h^*(S)+C_1.$$

（说明此时边缘集中必有最优路径上的点，若无，则已找到最优路径，且 T 已经出过边缘集了）

（b）同（a），

$$g(T)\leqslant g(n^*)+h^*(n^*)-(h^*(n^*)-h(n^*))\leqslant h^*(S)+(C_2-1)\,h^*(n^*)\leqslant C_2\cdot h^*(S)$$

（因为 h*(n*)≤h*(S)）

2.

记 h'(n)≡W •h(n)≤W •h*(n)，故根据 1 中结论，C_2=W,则 g(T) 上界为：W •h*(S)

3.

对任意节点 n，无论其是否在 S 到 T 的最优路径上，都找到从当前节点 n 到终点 T 的最短路径（若找不到则 h*(n)为无穷，也成立），可知该路径上的所有点到 T 的路径都是最短的。假设路径为 n -> n1 -> n2 ... -> T，则 h(n)-h(n1)≤c(n,n1)，h(n1)-h(n2)≤c(n1,n2)...

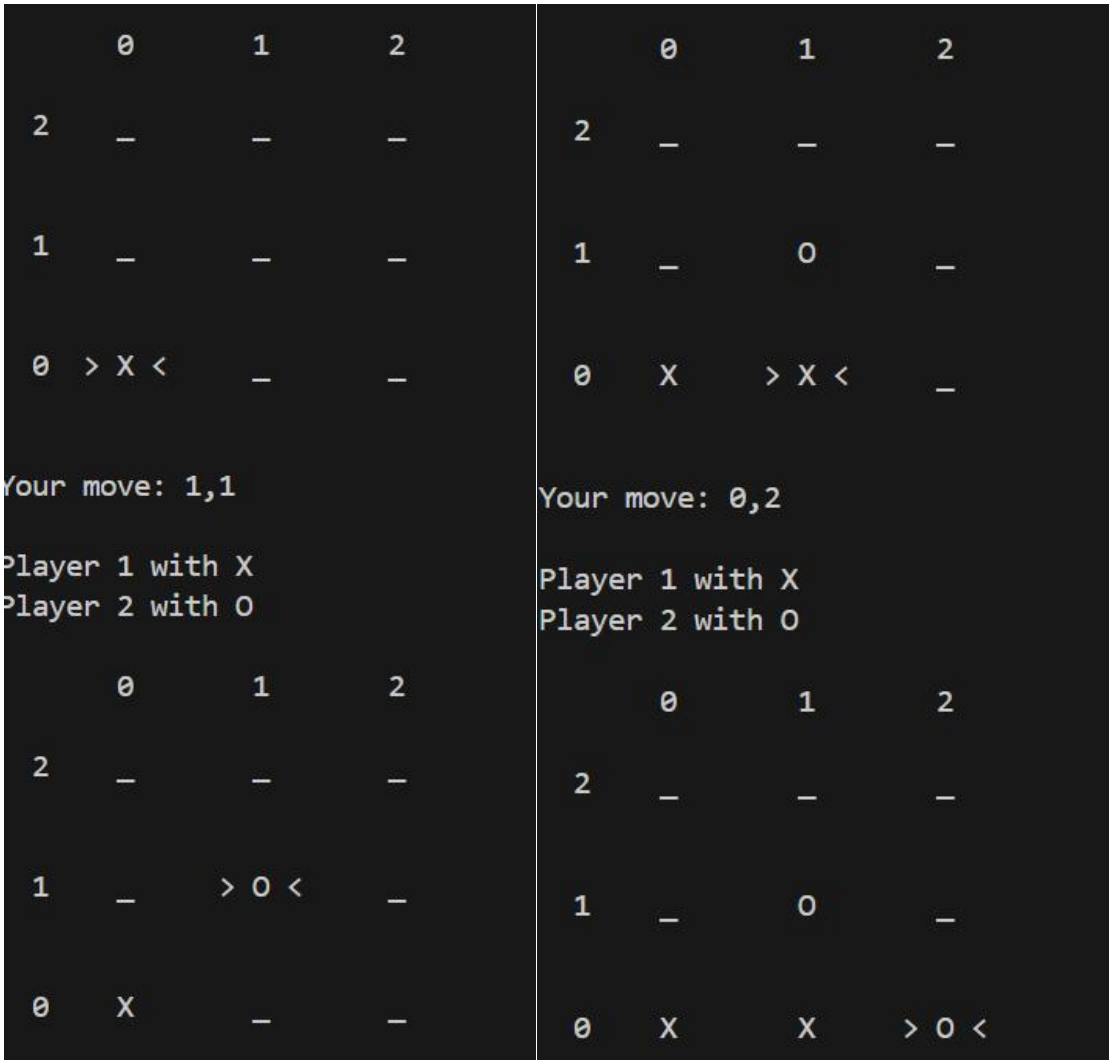
加和得：h(n)-h(T)≤路径 c 之和=h*(n)，而 h(T)=0，即可得：h(n)≤h*(n).

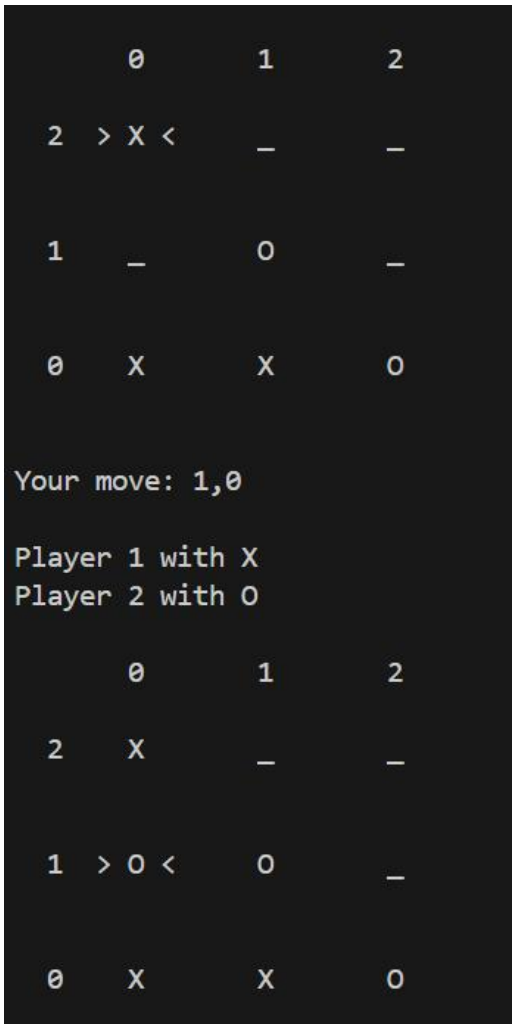
● 五子棋

问题 1：

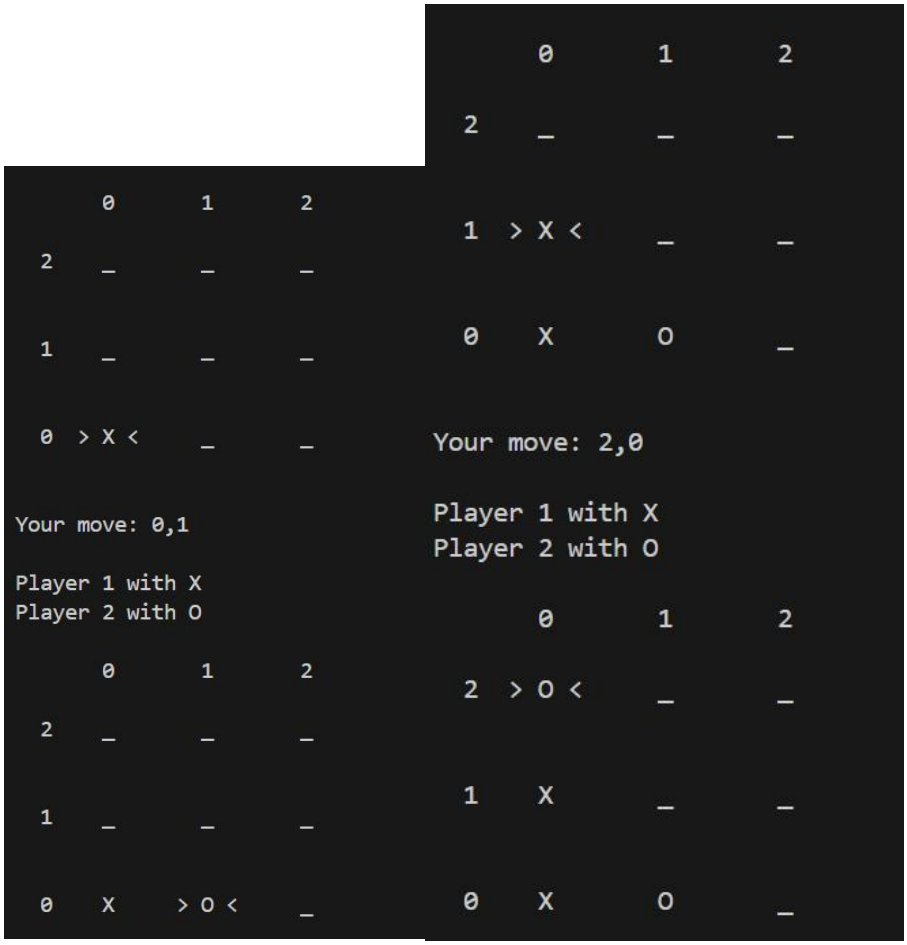
找到必不输策略。
--AI 先手下左下角，则我只能下中间才能平局，否则都是它赢。

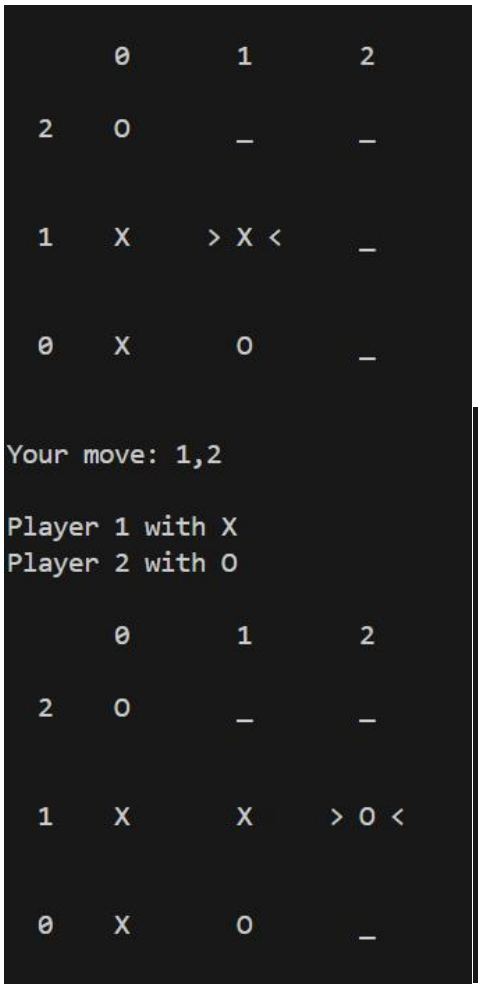
a.我下中间

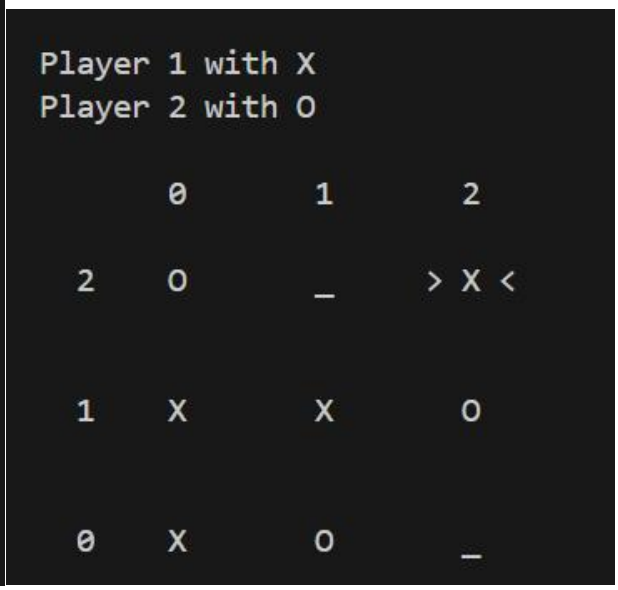



0 1 2
2 > X < _ _
1 _ O _
0 X X O
Your move: 1,0
Player 1 with X
Player 2 with O
0 1 2
2 X _ _
1 > O < O _
0 X X O

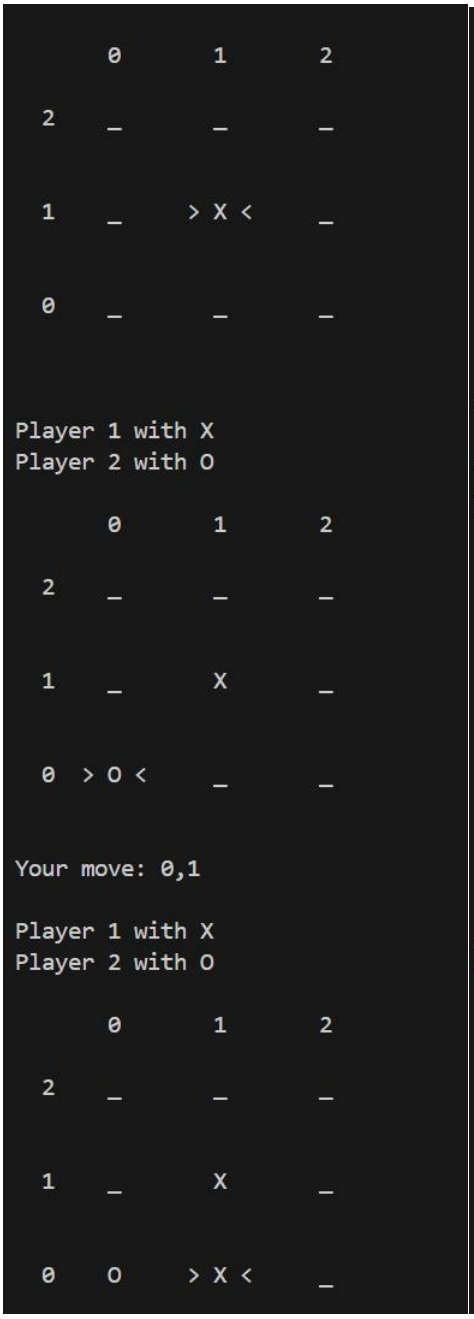
b.我下边

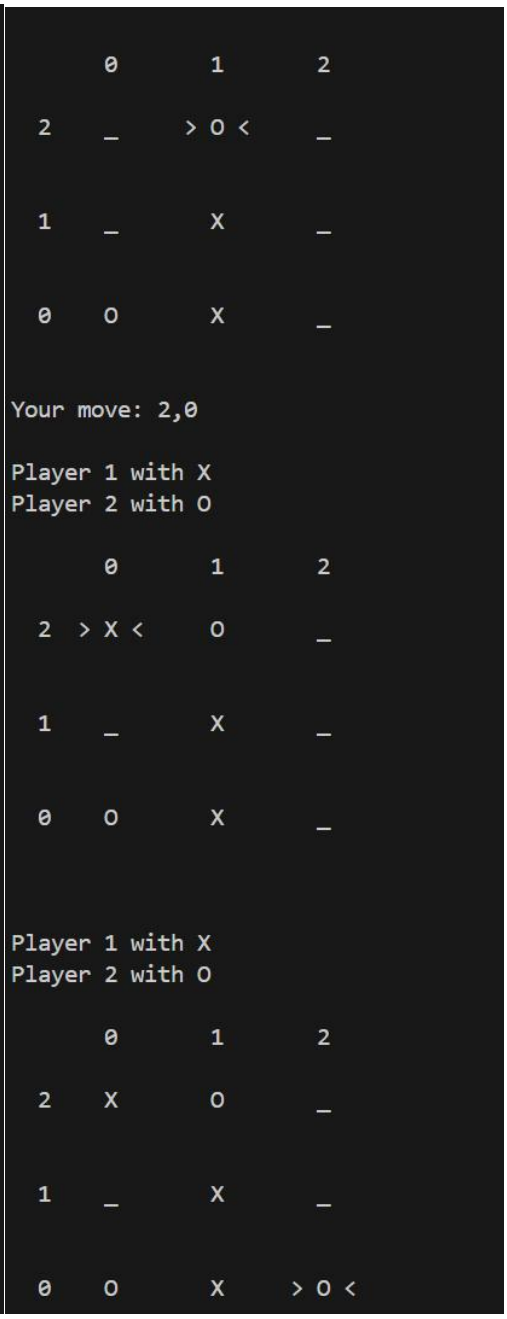

0 1 2
2 _ _ _
1 _ _ _
0 > X < _ _
Your move: 0,1
Player 1 with X
Player 2 with O
0 1 2
2 _ _ _
1 _ _ _
0 X > O < _
0 1 2
2 _ _ _
1 > X < _ _
0 X O _
Your move: 2,0
Player 1 with X
Player 2 with O
0 1 2
2 > O < _ _
1 X _ _
0 X O _

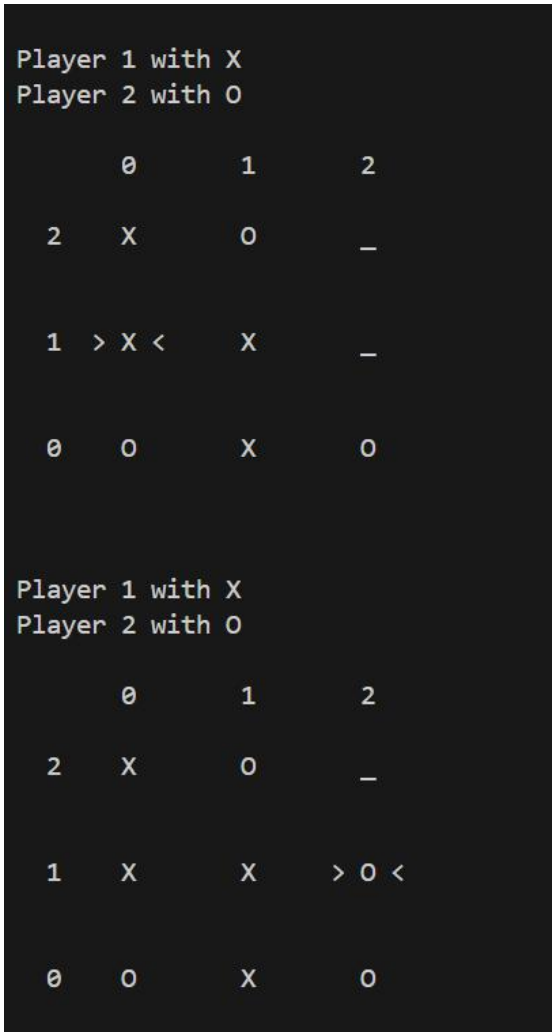

0 1 2
2 O _ _
1 X > X < _
0 X O _
Your move: 1,2
Player 1 with X
Player 2 with O
0 1 2
2 O _ _
1 X X > O <
0 X O _


Player 1 with X
Player 2 with O
0 1 2
2 O _ > X <
1 X X O
0 X O _

--我先手。若我下中间，它下角，我下边它堵另一边，我堵十字角，他走**(0,2)**，我走**(2,2)**，平局；下边和下角同样情况，平局。







问题 2：

-- MinimaxSearchPlayer 花了十几分钟仍无法跑通。

--AlphaBeta 找到先手必胜策略。

不过我发现他会“折磨”我，有些已经有两组两个连起来的和一个对应的空位，下空位就赢了，它却还要继续磨。不过其实很好理解，本质是采用 **DFS**，可能先对深的“长痛局”进行搜索，然后在后续搜索短的一击毙命的招时要么被剪枝要么因为值相等所以不更新。

```
Player 1 with X
Player 2 with O

      0       1       2       3

  2   _       _       _       _

  1   _       _       _       _

  0  > X <     _       _       _

Your move: 0,0
invalid move
Your move: 1,0

Player 1 with X
Player 2 with O

      0       1       2       3

  2   _       _       _       _

  1  > O <     _       _       _

  0   X       _       _       _
```

```
Player 1 with X
Player 2 with O

      0       1       2       3

  2   _       _       _       _

  1   O       _       _       _

  0   X     > X <     _       _

Your move: 0,2

Player 1 with X
Player 2 with O

      0       1       2       3

  2   _       _       _       _

  1   O       _       _       _

  0   X       X     > O <     _
```

```
Player 1 with X
Player 2 with O

      0       1       2       3

  2   _       _       _       _

  1   O     > X <     _       _

  0   X       X       O       _

Your move: 2,2

Player 1 with X
Player 2 with O

      0       1       2       3

  2   _       _     > O <     _

  1   O       X       _       _

  0   X       X       O       _
```

```
Player 1 with X
Player 2 with O

      0       1       2       3

  2   _       _       O       _

  1   O       X     > X <     _

  0   X       X       O       _

Your move: 1,3

Player 1 with X
Player 2 with O

      0       1       2       3

  2   _       _       O       _

  1   O       X       X     > O <

  0   X       X       O       _
```

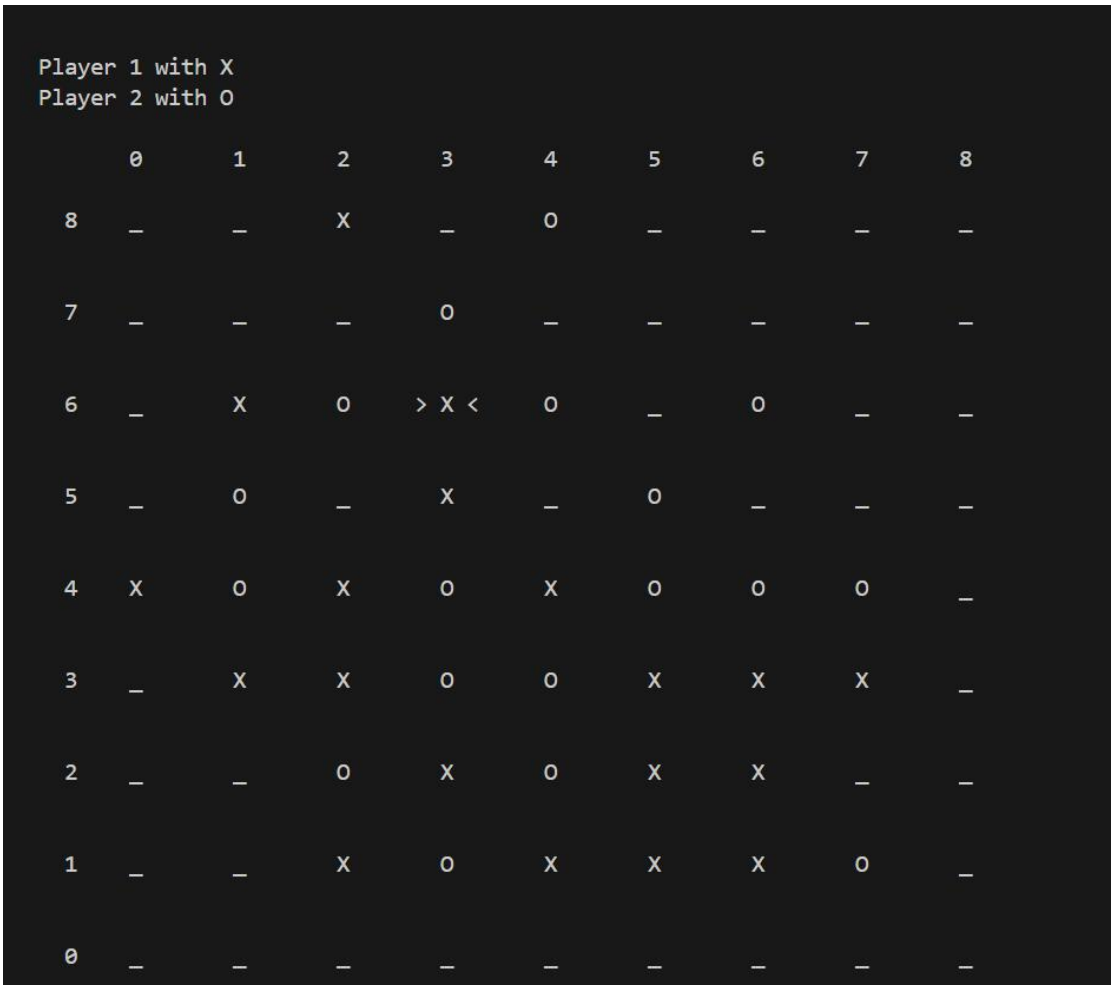
```
Player 1 with X
Player 2 with O

      0      1      2      3

  2   _      _      O      _


  1   O      X      X      O


  0   X      X      O    > X <


Your move: 2,1

Player 1 with X
Player 2 with O

      0      1      2      3

  2   _    > O <    O      _


  1   O      X      X      O


  0   X      X      O      X
```

```
Player 1 with X
Player 2 with O

       0      1      2      3

  2    _      O      O    > X <


  1    O      X      X      O


  0    X      X      O      X
```

问题 3：

很厉害，尤其是前中期，完全被追着打，甚至还会造一些不容易发现的活 3+冲 4，但是后期有时候莫名其妙地会不堵我的活 3，可能是由于棋盘中太多的其他棋型导致活 3 的影响下降了，后来又进一步调整比例才好了。

最终状态 1(活四的参数值拉得不够大时）：

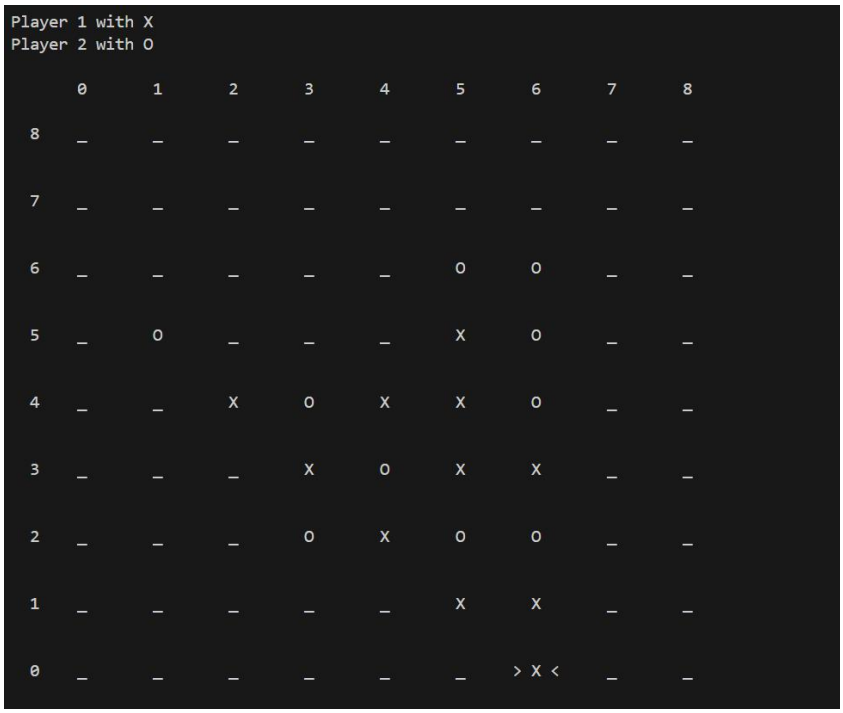


参数拉大后的最终状态 2：这把打得很憋屈，全程被压着打，一点还手之力都没有，它甚至不让我造活 3。

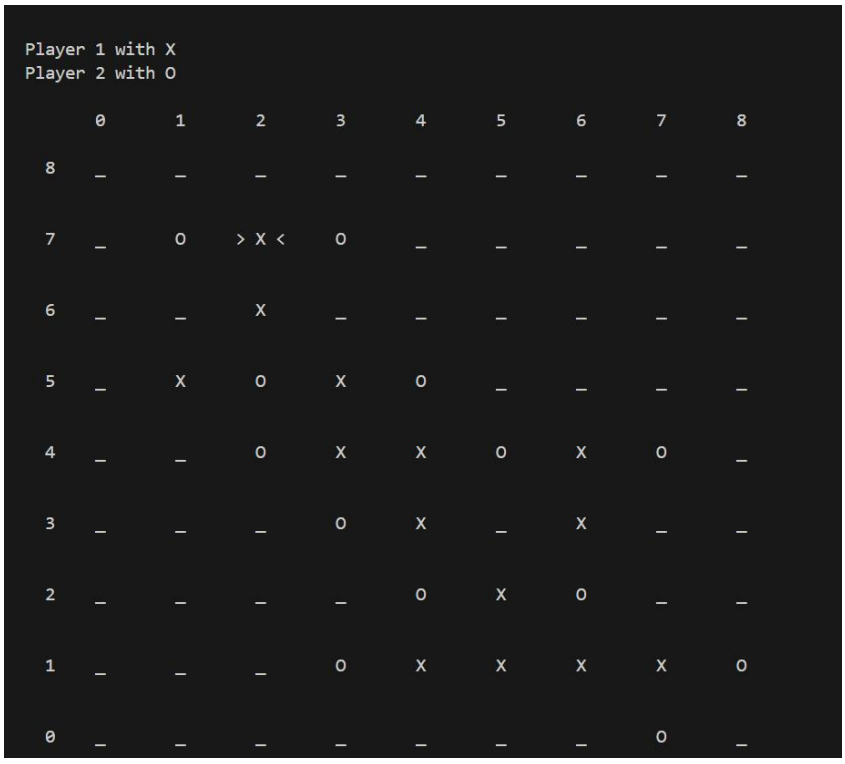
```
    # 为不同的棋型设置不同的得分
    scores = {
        "live_four": 1000,    # 活四
        "four": 200,       # 冲四
        "live_three": 200, # 活三
        "three": 10,      # 眠三
        "live_two": 5,  # 活二
    }


    # 评估当前玩家的棋型得分
    for pattern, pattern_score in scores.items():
        score += info[player][pattern] * pattern_score


    # 扣除对手的棋型得分
    for pattern, pattern_score in scores.items():
        score -= info[opponent][pattern] * pattern_score

    score=score/1500
```

```
Player 1 with X
Player 2 with O

      0     1     2     3     4     5     6     7     8

  8   _     _     _     _     _     _     _     _     _

  7   _     _     _   > X <   _     _     _     _     _

  6   _     X     O     O     _     _     _     O     _

  5   _     O     O     O     X     _     _     X     _

  4   O     X     X     X     X     O     O     X     O

  3   _     _     O     O     X     X     O     X     _

  2   _     _     _     _     O     X     X     X     _

  1   _     _     _     O     X     X     X     O     _

  0   _     _     _     _     _     _     _     O     _
```

这把更是快速输掉

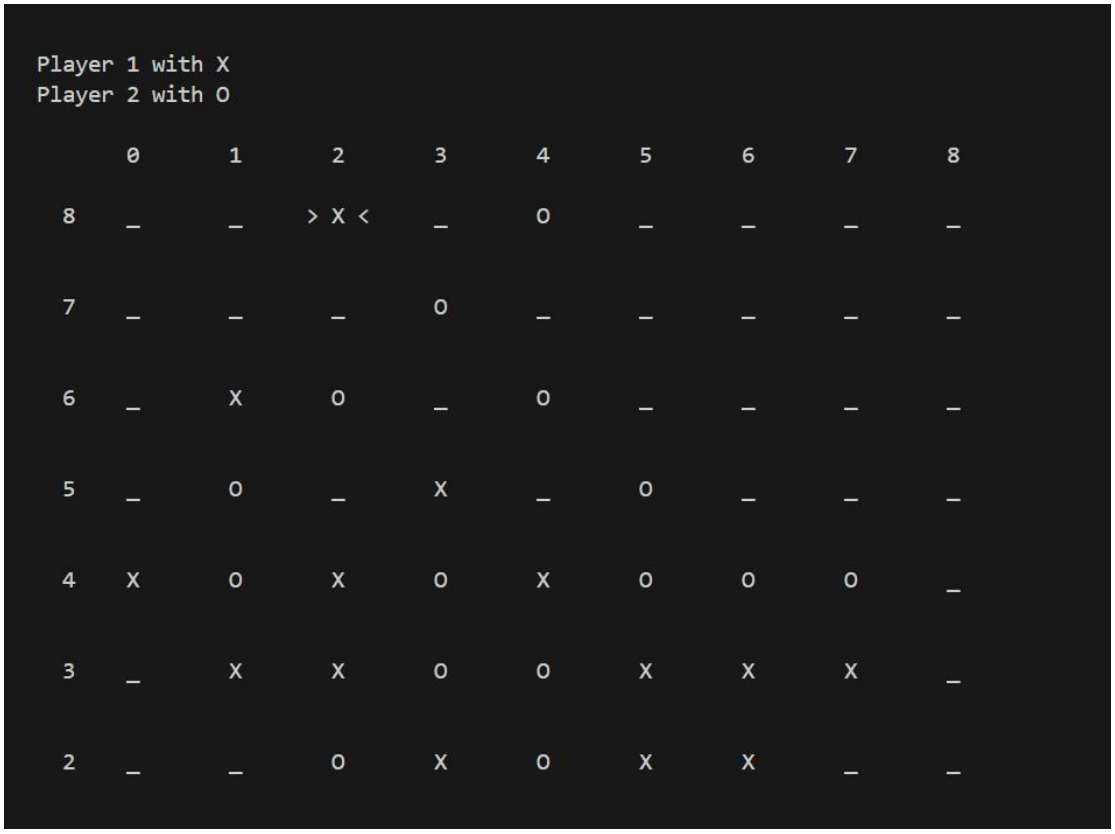


这把没一点机会

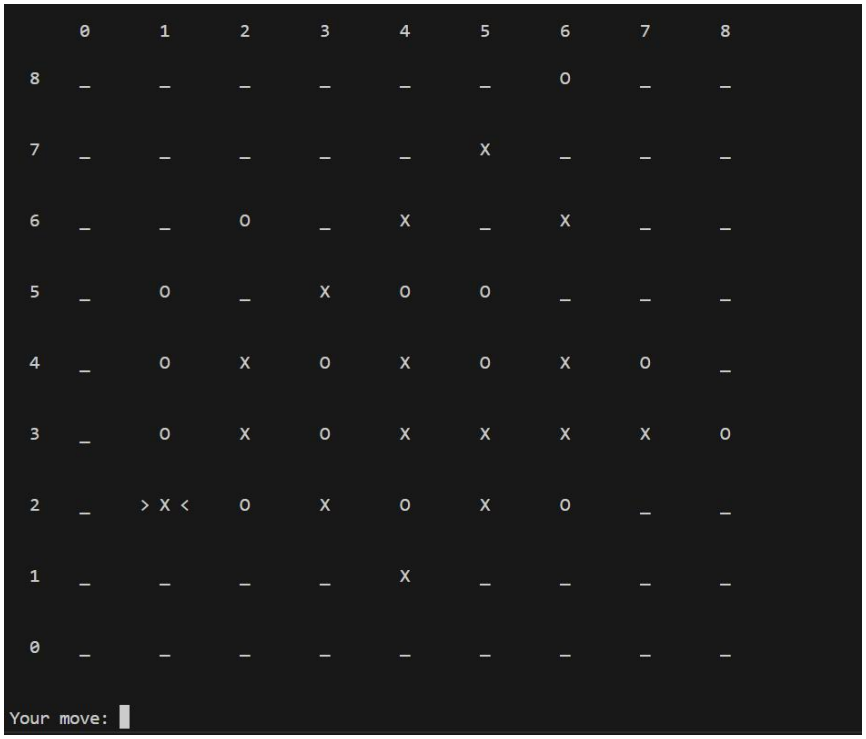


、

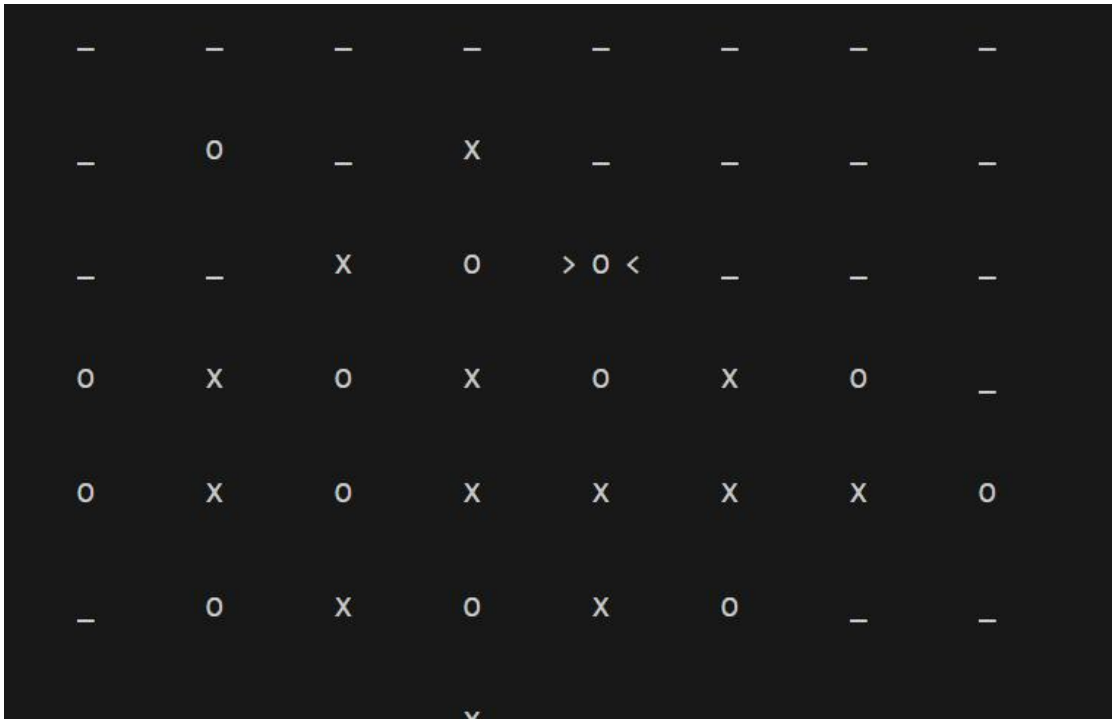
堵冲 4

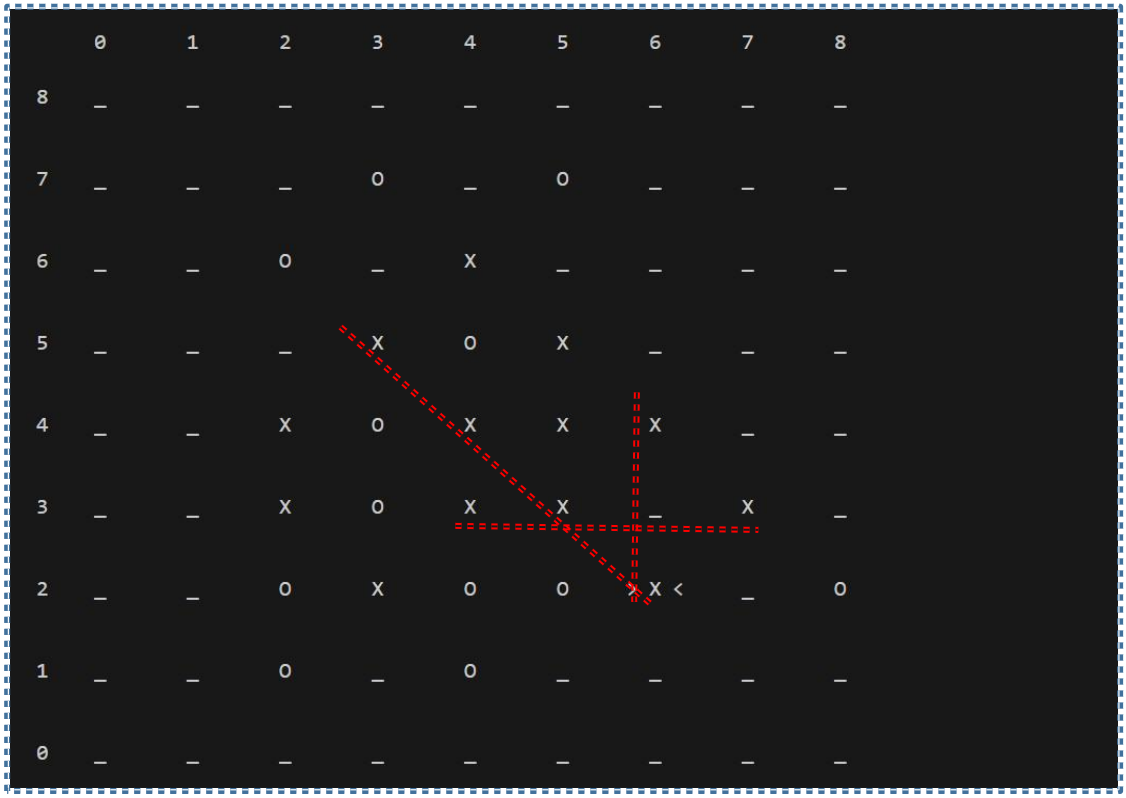

Player 1 with X
Player 2 with O
0 1 2 3 4 5 6 7 8
8 _ _ > X < _ O _ _ _ _
7 _ _ _ O _ _ _ _ _
6 _ X O _ O _ _ _ _
5 _ O _ X _ O _ _ _
4 X O X O X O O O _
3 _ X X O O X X X _
2 _ _ O X O X X _ _

堵活 3

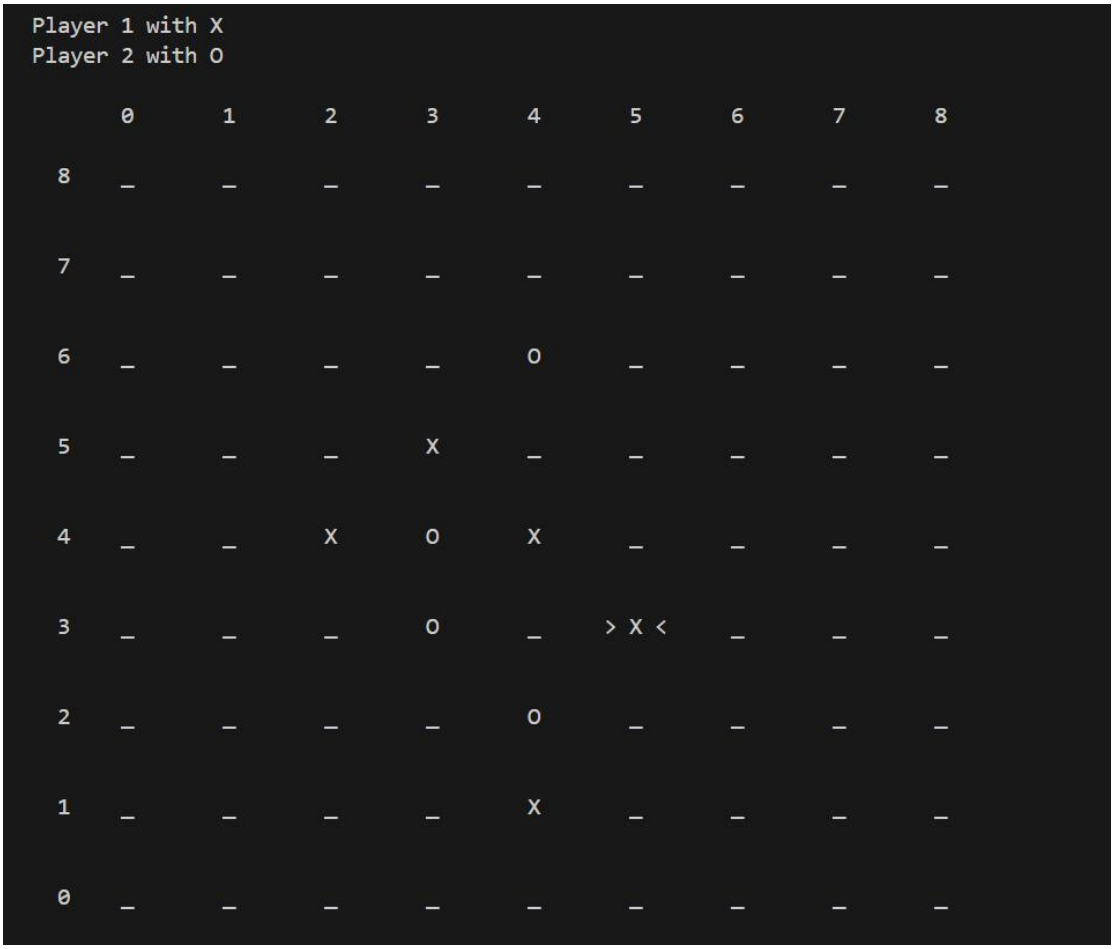

0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ O _ _
7 _ _ _ _ _ X _ _ _
6 _ _ O _ X _ X _ _
5 _ O _ X O O _ _ _
4 _ O X O X O X O _
3 _ O X O X X X X O
2 _ > X < O X O X O _ _
1 _ _ _ _ X _ _ _ _
0 _ _ _ _ _ _ _ _ _
Your move:

造间隔活 3、三管齐下：

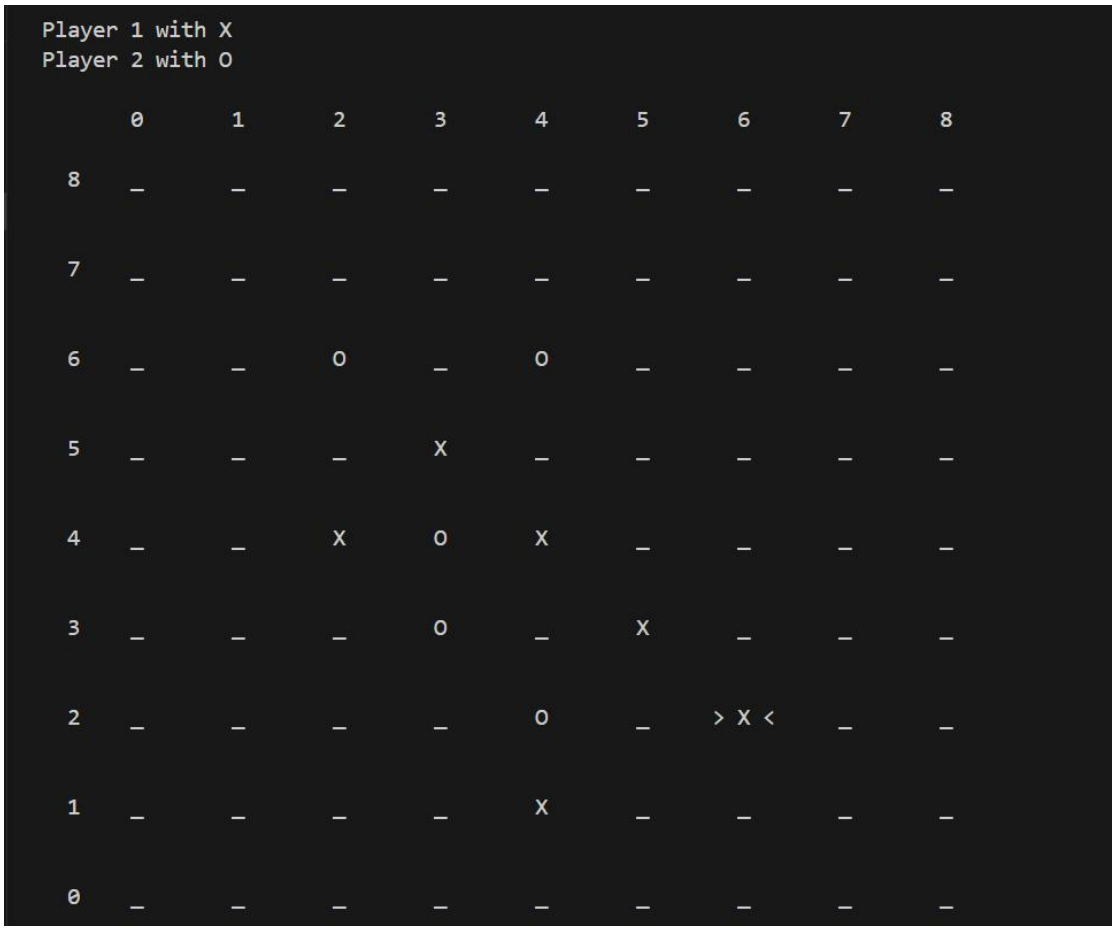

_ _ _ _ _ _ _ _
_ O _ X _ _ _ _
_ _ X O > O < _ _ _
O X O X O X O _
O X O X X X X O
_ O X O X O _ _
X


0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ _ _ _
7 _ _ _ O _ O _ _ _
6 _ _ O _ X _ _ _ _
5 _ _ _ X O X _ _ _
4 _ _ X O X X X _ _
3 _ _ X O X X _ X _
2 _ _ O X O O X < _ O
1 _ _ O _ O _ _ _ _
0 _ _ _ _ _ _ _ _ _

主动进攻：


Player 1 with X
Player 2 with O
0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ _ _ _
7 _ _ _ _ _ _ _ _ _
6 _ _ _ _ O _ _ _ _
5 _ _ _ X _ _ _ _ _
4 _ _ X O X _ _ _ _
3 _ _ _ O _ > X < _ _ _
2 _ _ _ _ O _ _ _ _
1 _ _ _ _ X _ _ _ _
0 _ _ _ _ _ _ _ _ _

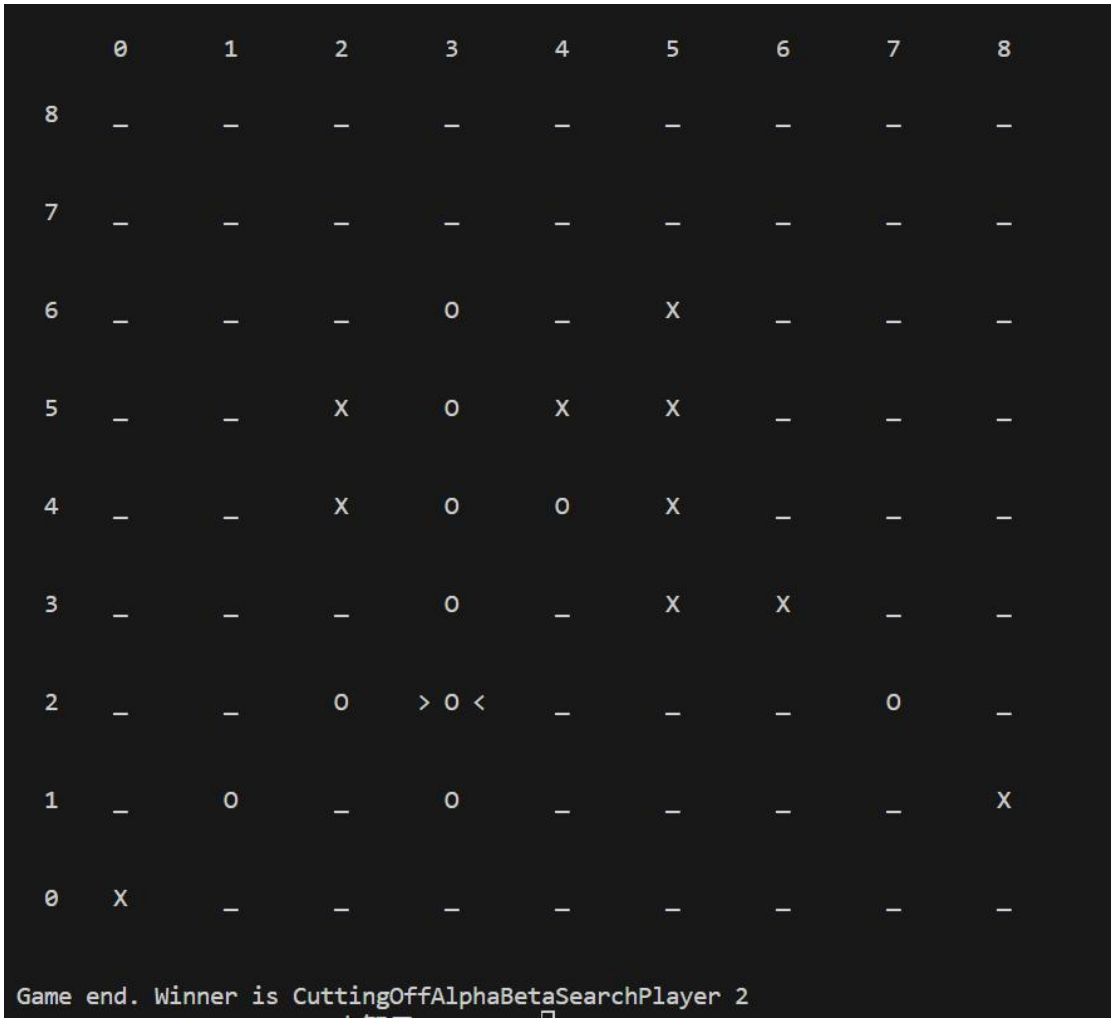
连续进攻：


Player 1 with X
Player 2 with O
0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ _ _ _
7 _ _ _ _ _ _ _ _ _
6 _ _ O _ O _ _ _ _
5 _ _ _ X _ _ _ _ _
4 _ _ X O X _ _ _ _
3 _ _ _ O _ X _ _ _
2 _ _ _ _ O _ > X < _ _
1 _ _ _ _ X _ _ _ _
0 _ _ _ _ _ _ _ _ _

问题 4

试验 1：MCTS 先手，最后 MCTS 赢了，但赢得比较抽象，两个人菜鸡互啄了，本来打得好好的，eval（cutting off alpha beta）有个双三，一直不继续下。

试验 2：（再改进 EVAL，将其活 4 拉高后），EVAL 压倒性地击败了 sample 数为 700 的 mcts.



问题 5

评估函数的基本思想是将己方棋型评分减去对手棋型评分。而根据经验，棋型评分中，己方 4 子是必胜情况，分值高；活 3 和冲 4 对于对手而言都是需要封堵的（一般情况），应较高，而其他棋型有衍生为上述的可能性，分值相对较低。

最开始的 alphazero 初始行棋很高超，各种造 3 造 4，但我发现不具备杀死比赛的能力，表现为已经连成四个了他就是不填第五个，单独的 mcts 和单独的 evaluation 都没有这个问题。

后来我发现了问题所在我写的时候把活 4 和冲 4 直接 return1，导致它认为这个局面就是最好的，反而不觉得比赛胜利（连成 5 个）是最好的了，于是加入了胜负的判断，并将其设为 1，而将活 4 等必胜情况和各种棋型和距离相应调低。

改进后如下：

```
if end:
    if winner == -1:
        return 0  # 平局得分为0
    else:
        return (1 if winner == player else -1)  # 获胜得分为1，失败为-1
if(info[player]["live_four"] or info[player]["four"]):return 0.5    #0只有0.5是为了与正儿八经胜利区分开
# 为不同的棋型设置不同的得分
scores = {
    "live_four": 1000,    # 活四
    "four": 200,       # 冲四
    "live_three": 200, # 活三
    "three": 10,      # 眠三
    "live_two": 5,  # 活二
}
# 评估当前玩家的棋型得分
for pattern, pattern_score in scores.items():
    score += info[player][pattern] * pattern_score
# 扣除对手的棋型得分
for pattern, pattern_score in scores.items():
    score -= info[opponent][pattern] * pattern_score
score=score/10000           #一般来说，单纯棋型的累加最多到0.2，明显劣势于执棋有4、赢棋的情况，但优于距离

# 考虑棋子距离棋盘中心的距离
center_distance_score = -info[player]["max_distance"] + info[opponent]["max_distance"]
# 距离中心越近越好
score+=center_distance_score*0.0015
return score
```

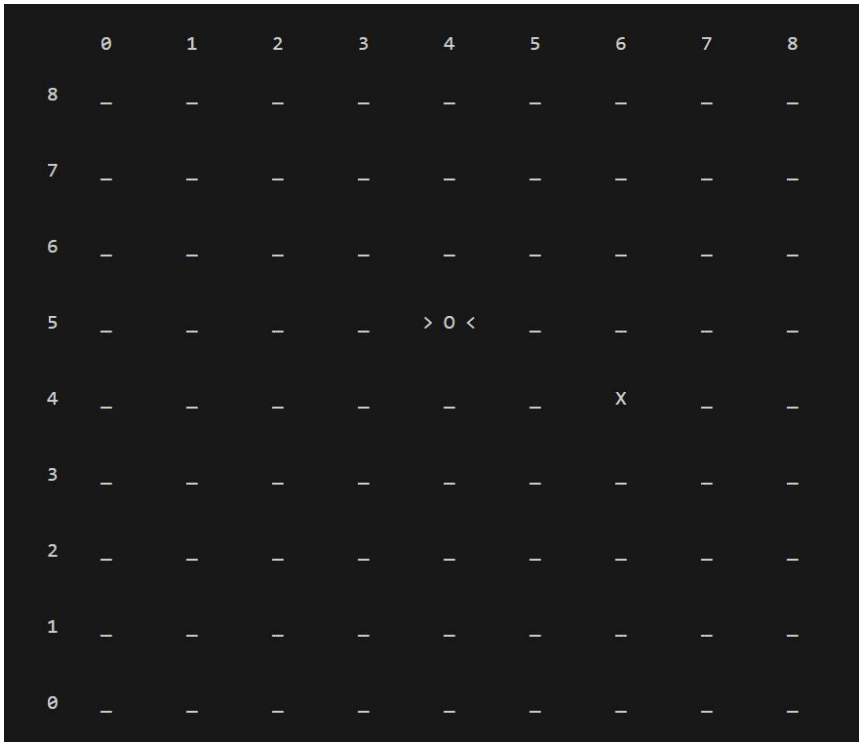
但发现 AI 攻击很厉害，尤其一开始咄咄逼人，不断进攻，让对方只能被迫防守，不过到了后面它的防守偏弱，对手要活 4 的情况都不防了，检查代码后发现，可能是过于强调自己冲 4 了，而忽略了防守，于是进一步修改如下：

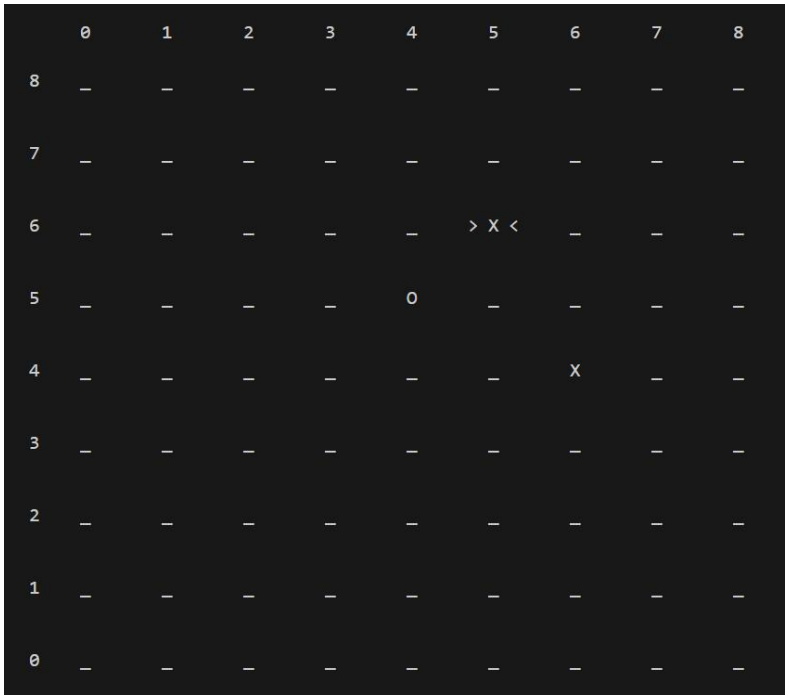
```
if(info[player]["live_four"] or info[player]["four"]):return 0.5    #0只有0.5是为了与正儿八经胜利区分开
elif (info[opponent]["live_four"]): return -0.4   #一般来说，避免对方活4（除非自己有4）

# 为不同的棋型设置不同的得分
scores = {
    # "live_four": 1000,    # 活四（已经无需讨论）
    "four": 200,        # 对手冲四
    "live_three": 200, # 活三
    "three": 20,       # 眠三
    "live_two": 5,  # 活二
}
# 评估当前玩家的棋型得分
for pattern, pattern_score in scores.items():
    score += info[player][pattern] * pattern_score
# 扣除对手的棋型得分
for pattern, pattern_score in scores.items():
    score -= info[opponent][pattern] * pattern_score
score=score/1000           #一般来说，单纯棋型的累加最多到0.2，明显劣势于执棋有4、赢棋的情况，但优于距离

# 考虑棋子距离棋盘中心的距离
center_distance_score = -info[player]["max_distance"] + info[opponent]["max_distance"]
# 距离中心越近越好
score+=center_distance_score*0.015
return score
```

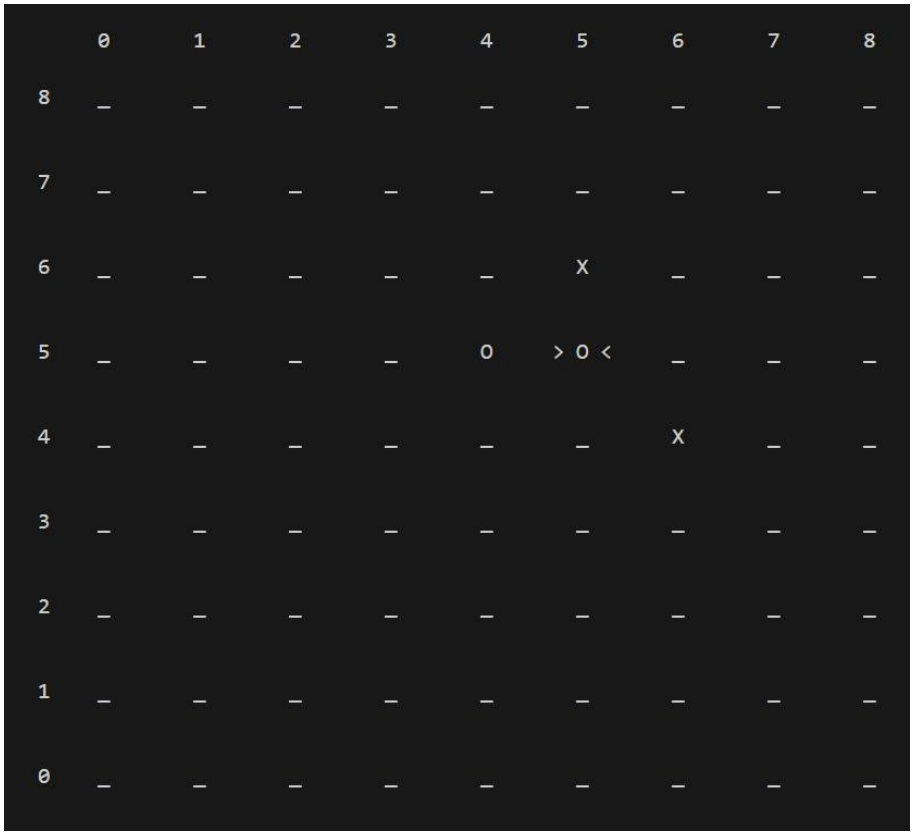
这样修改后，AI 进攻性略微减弱（尤其初期没有进攻灵性），但是防守大幅加强，而且后期很厉害，攻守兼备。

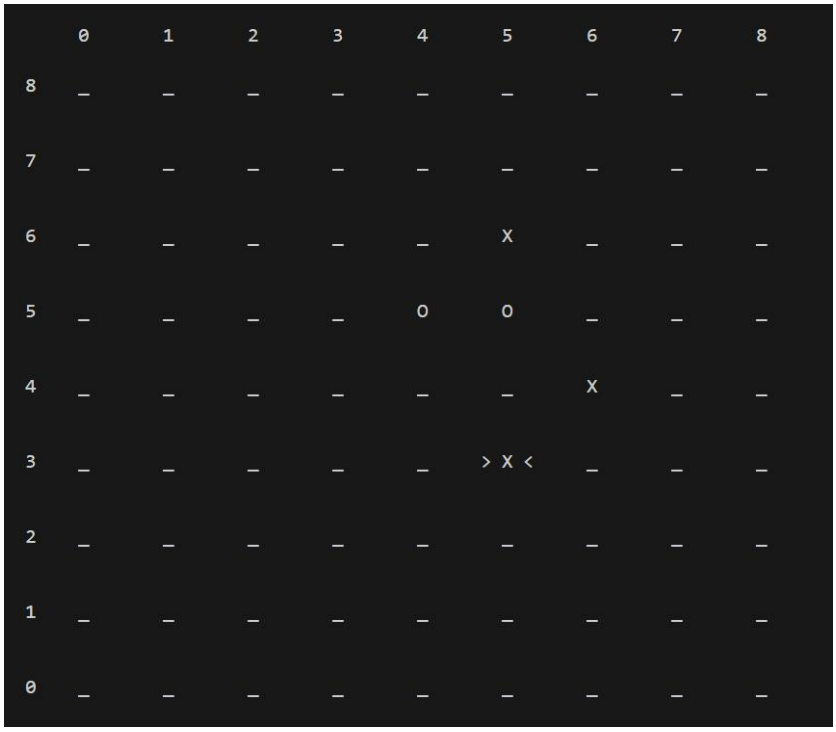
case1:alphazero vs mcts(O 为 alphazero)

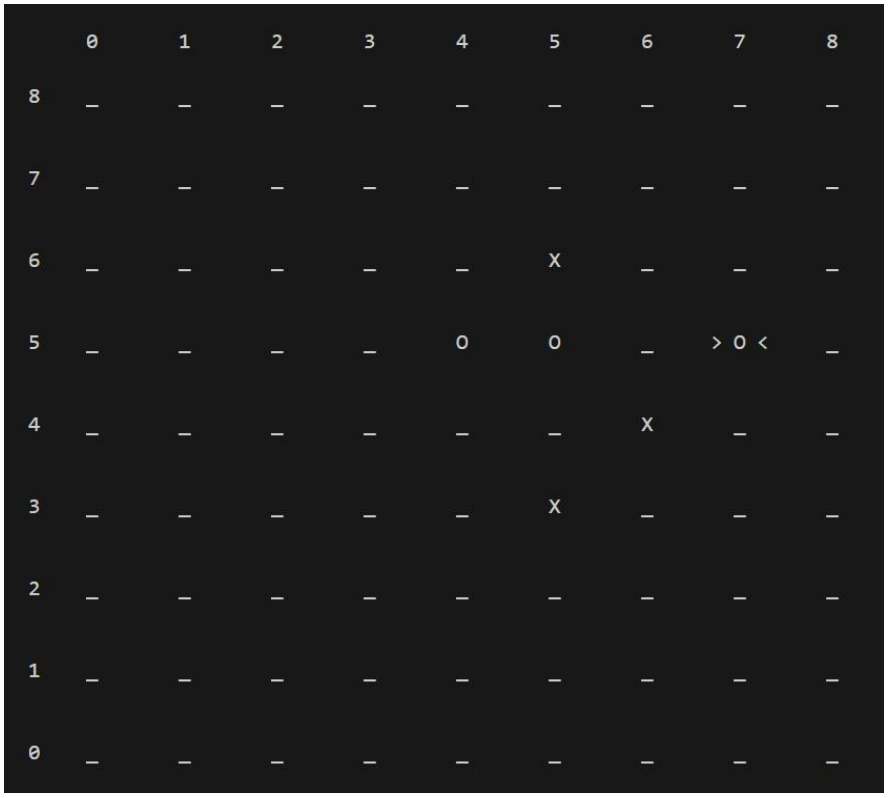
这里的 mcts 有点菜

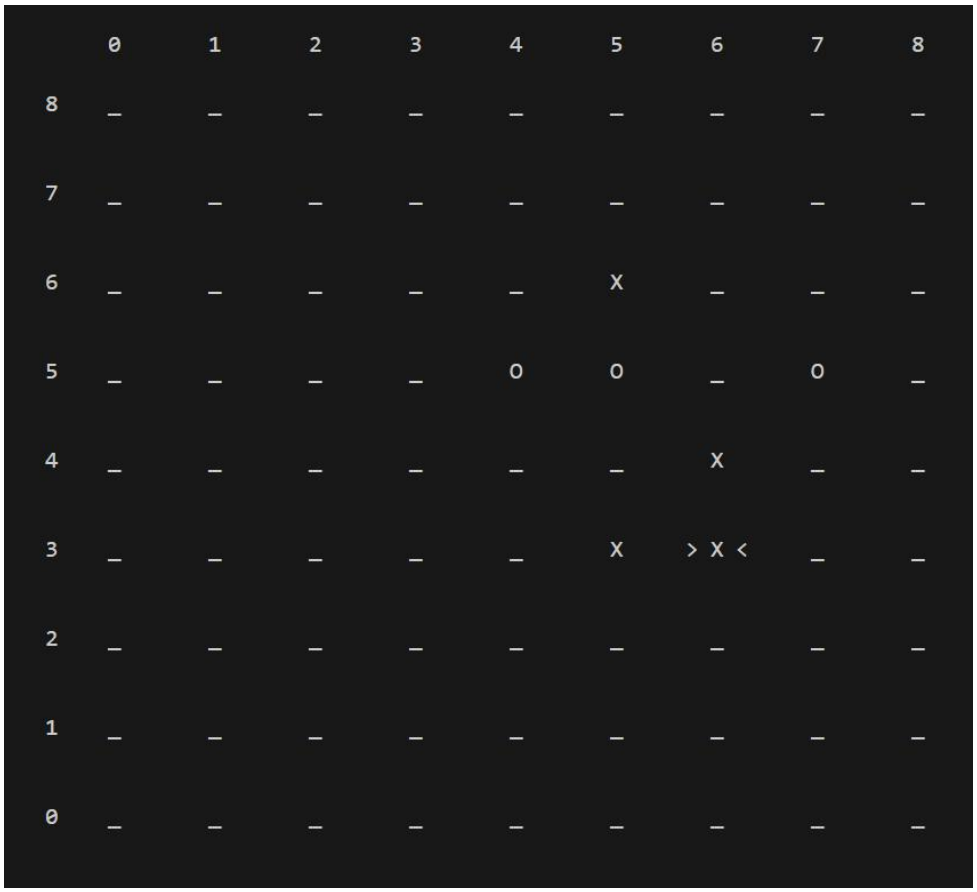

0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ _ _ _
7 _ _ _ _ _ _ _ _ _
6 _ _ _ _ _ _ _ _ _
5 _ _ _ _ > O < _ _ _ _
4 _ _ _ _ _ _ X _ _
3 _ _ _ _ _ _ _ _ _
2 _ _ _ _ _ _ _ _ _
1 _ _ _ _ _ _ _ _ _
0 _ _ _ _ _ _ _ _ _

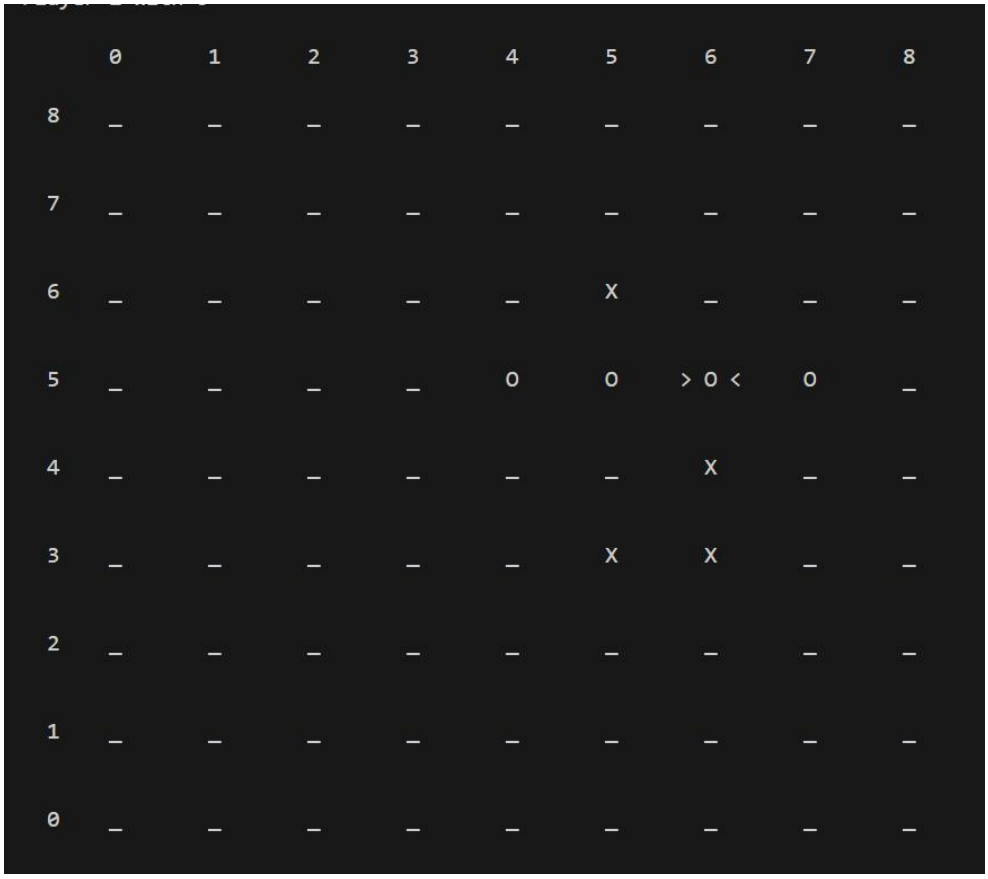

0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ _ _ _
7 _ _ _ _ _ _ _ _ _
6 _ _ _ _ _ > X < _ _ _
5 _ _ _ _ O _ _ _ _
4 _ _ _ _ _ _ X _ _
3 _ _ _ _ _ _ _ _ _
2 _ _ _ _ _ _ _ _ _
1 _ _ _ _ _ _ _ _ _
0 _ _ _ _ _ _ _ _ _

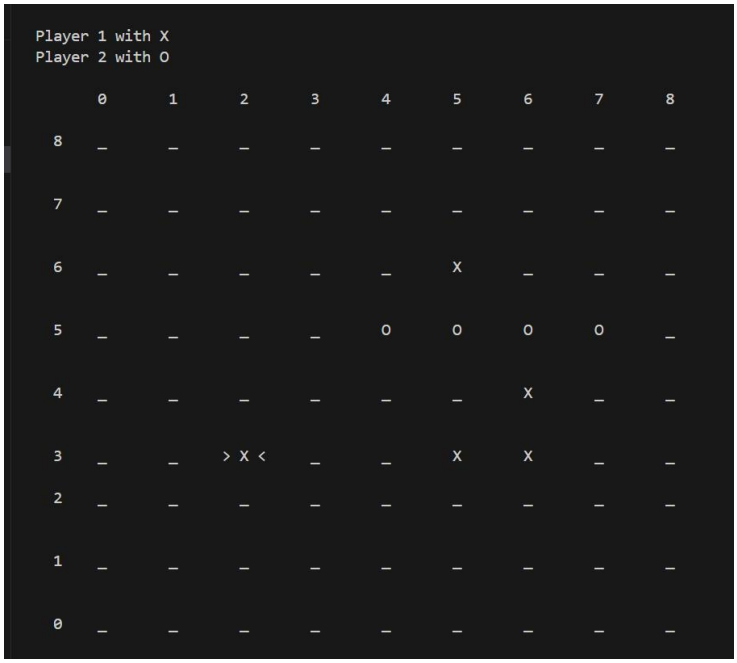

0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ _ _ _
7 _ _ _ _ _ _ _ _ _
6 _ _ _ _ _ X _ _ _
5 _ _ _ _ O > O < _ _ _
4 _ _ _ _ _ _ X _ _
3 _ _ _ _ _ _ _ _ _
2 _ _ _ _ _ _ _ _ _
1 _ _ _ _ _ _ _ _ _
0 _ _ _ _ _ _ _ _ _


0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ _ _ _
7 _ _ _ _ _ _ _ _ _
6 _ _ _ _ _ X _ _ _
5 _ _ _ _ O O _ _ _
4 _ _ _ _ _ _ X _ _
3 _ _ _ _ _ > X < _ _ _
2 _ _ _ _ _ _ _ _ _
1 _ _ _ _ _ _ _ _ _
0 _ _ _ _ _ _ _ _ _


0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ _ _ _
7 _ _ _ _ _ _ _ _ _
6 _ _ _ _ _ X _ _ _
5 _ _ _ _ O O _ > O < _
4 _ _ _ _ _ _ X _ _
3 _ _ _ _ _ X _ _ _
2 _ _ _ _ _ _ _ _ _
1 _ _ _ _ _ _ _ _ _
0 _ _ _ _ _ _ _ _ _


0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ _ _ _
7 _ _ _ _ _ _ _ _ _
6 _ _ _ _ _ X _ _ _
5 _ _ _ _ O O _ O _
4 _ _ _ _ _ _ X _ _
3 _ _ _ _ _ X > X < _ _
2 _ _ _ _ _ _ _ _ _
1 _ _ _ _ _ _ _ _ _
0 _ _ _ _ _ _ _ _ _

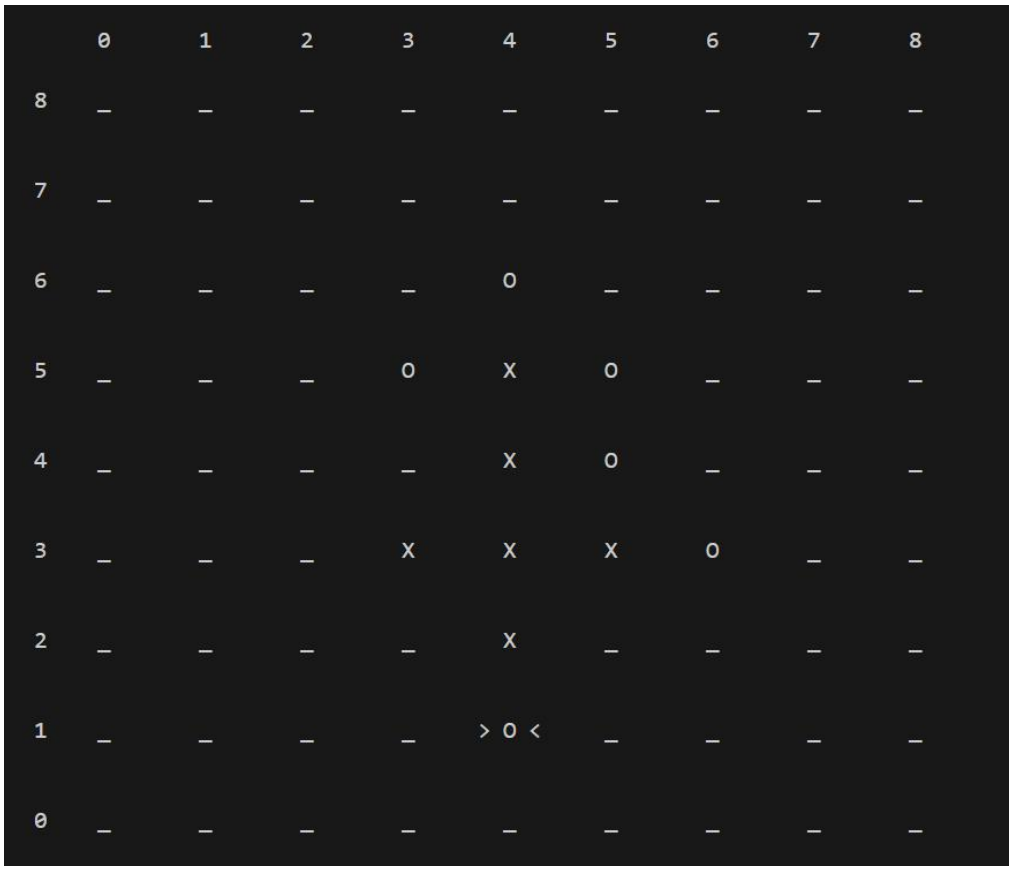

0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ _ _ _
7 _ _ _ _ _ _ _ _ _
6 _ _ _ _ _ X _ _ _
5 _ _ _ _ O O > O < O _
4 _ _ _ _ _ _ X _ _
3 _ _ _ _ _ X X _ _
2 _ _ _ _ _ _ _ _ _
1 _ _ _ _ _ _ _ _ _
0 _ _ _ _ _ _ _ _ _

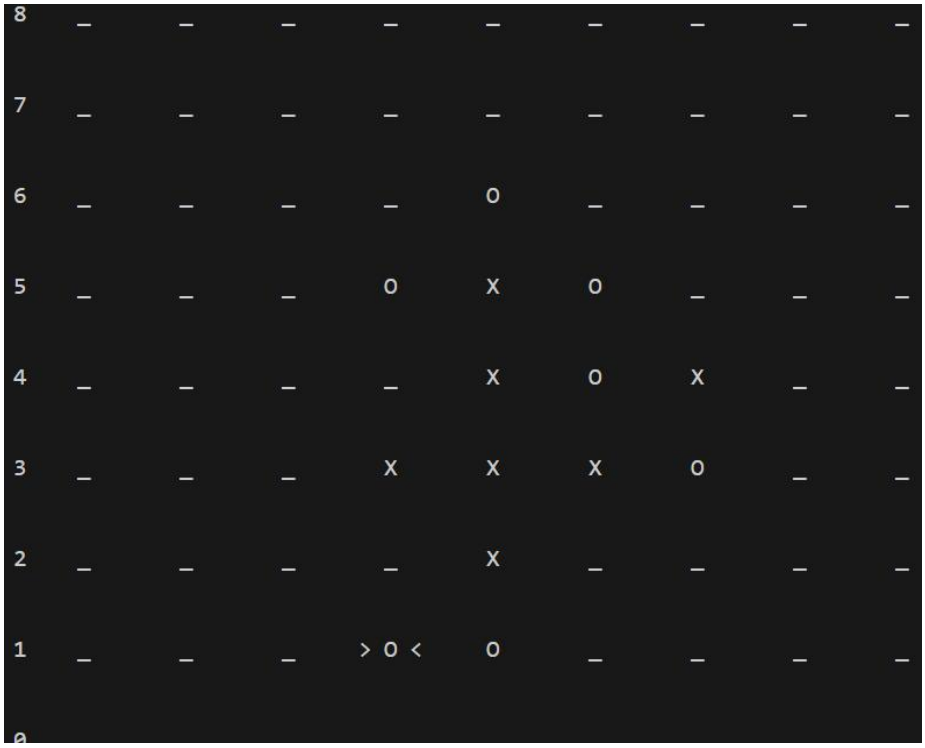

Player 1 with X
Player 2 with O
0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ _ _ _
7 _ _ _ _ _ _ _ _ _
6 _ _ _ _ _ X _ _ _
5 _ _ _ _ O O O O _
4 _ _ _ _ _ _ X _ _
3 _ _ > X < _ _ X X _ _
2 _ _ _ _ _ _ _ _ _
1 _ _ _ _ _ _ _ _ _
0 _ _ _ _ _ _ _ _ _

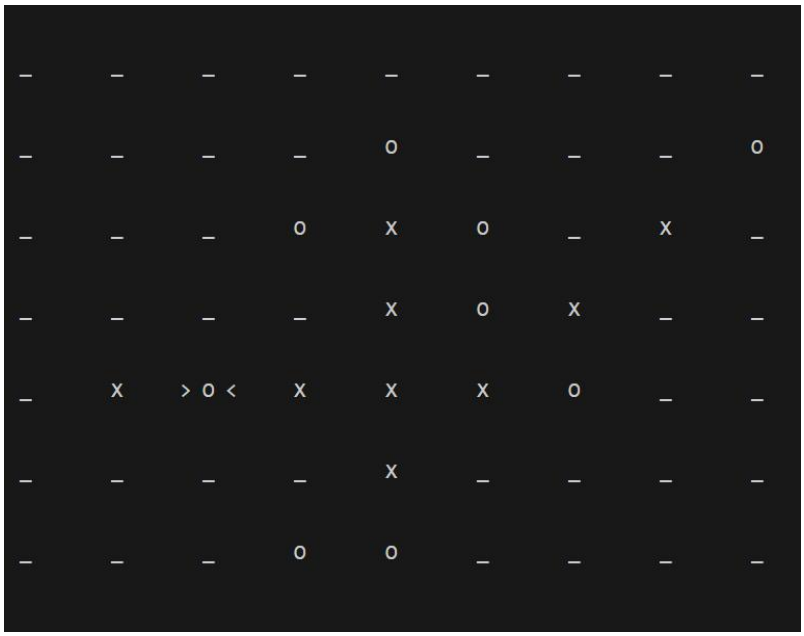

0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ _ _ _ _
7 _ _ _ _ _ _ _ _ _
6 _ _ _ _ _ X _ _ _
5 _ _ _ > O < O O O O _
4 _ _ _ _ _ _ X _ _
3 _ _ X _ _ X X _ _
2 _ _ _ _ _ _ _ _ _
1 _ _ _ _ _ _ _ _ _
0 _ _ _ _ _ _ _ _ _

case2:alphazero vs Cuttingoff(O 为 alphazero)

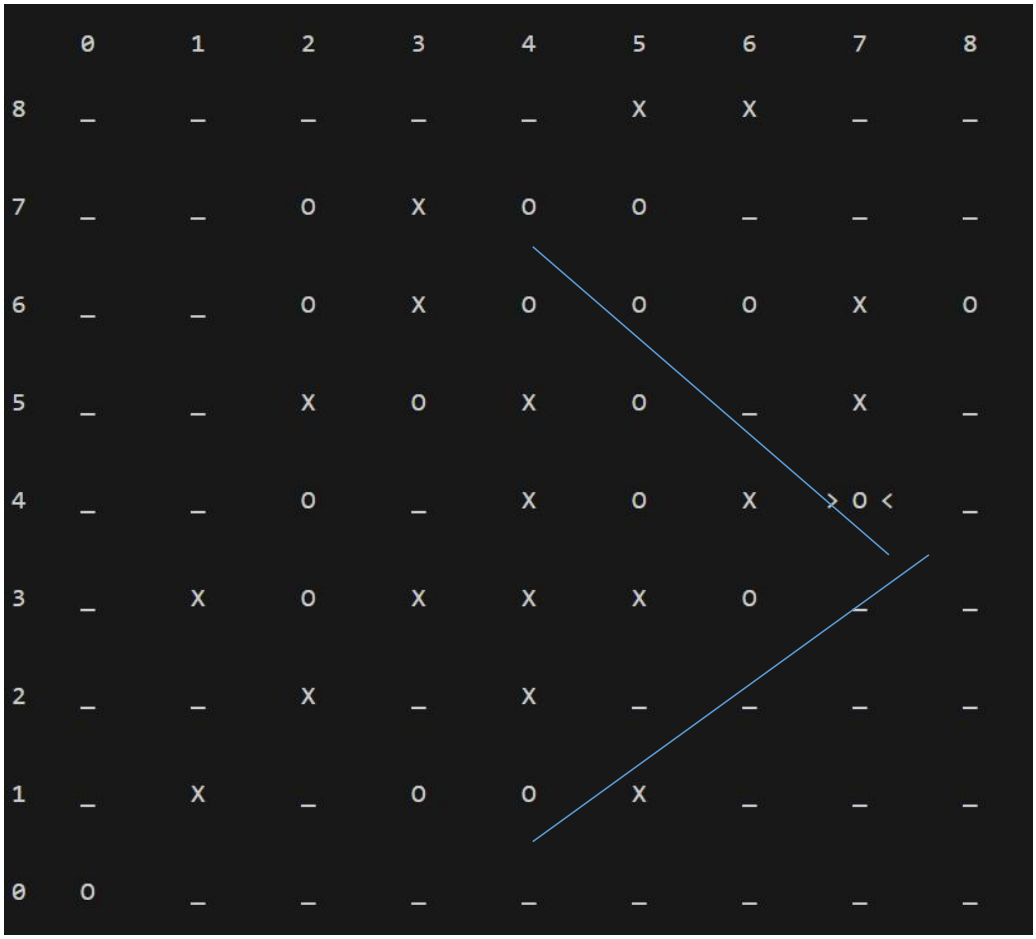
看高手过招。






_ _ _ _ _ _ _ _ _
_ _ _ _ o _ _ _ o
_ _ _ o x o _ x _
_ _ _ _ x o x _ _
_ x > o < x x x o _ _
_ _ _ _ x _ _ _ _
_ _ _ o o _ _ _ _

关键双 **3**


0 1 2 3 4 5 6 7 8
8 _ _ _ _ _ X X _ _
7 _ _ o X o o _ _ _
6 _ _ o X o o o X o
5 _ _ X o X o _ X _
4 _ _ o _ X o X > o < _
3 _ X o X X X o _ _
2 _ _ X _ X _ _ _ _
1 _ X _ o o X _ _ _
0 o _ _ _ _ _ _ _ _

```
Player 1 with X
Player 2 with O

      0     1     2     3     4     5     6     7     8

  8   _     _     _   > O <   _     X     X     _     _

  7   _     _     O     X     O     O     _     _     _

  6   _     _     O     X     O     O     O     X     O

  5   _     _     X     O     X     O     O     X     _

  4   _     _     O     _     X     O     X     O     _

  3   _     X     O     X     X     X     O     _     _

  2   _     _     X     _     X     X     _     _     _

  1   _     X     _     O     O     X     _     _     _

  0   O     X     _     _     _     _     _     _     _

Game end. Winner is AlphaZeroPlayer 2
```

用 **stats.py**（关闭棋盘输出）自动打了 **20** 场，都是 **alphazero** 胜利

```
PS D:\Documents\THU3_2\人智导\search> python -u "d:\Documents\THU3_2\人智导\search\stats.py"
Results after 20 matches:
AlphaZeroPlayer wins: 20 (100.00%)
MCTSPlayer wins: 0 (0.00%)
```