**中山大学数据科学与计算机学院**

**计算机科学与技术-人工智能**

**本科生实验报告**

**（2018-2019学年秋季学期）**

课程名称：**Artificial Intelligence**

|  |  |  |  |
| --- | --- | --- | --- |
| 教学班级 | 教务二班 | 专业（方向） | 计算机科学与技术超算 |
| 学号 | 16337107 | 姓名 | 金柏汇 |

* **实验题目**

博弈树搜索

* **实验内容**

**minimax搜索：两名玩家交替行动，而利益关系对立，一名玩家试图让评估分数更大，另一名玩家试图让分数更小，两个玩家均采取最优策略，进行多层搜索返回最优解**

**αβ剪枝：由于minimax搜索时，随着深度的加大叶子节点的数量成指数级增长，严重限制了搜索性能，而minimax的目标是寻找最优解,在搜索节点满足一定条件时可以提前预测到这个行动不会被选择而进行剪枝。当极小层返回的值小于alpha值，此时由于极大值节点要选择最大的值，所以肯定不会走这种对自己不利的选择，故发生剪枝，同理极大值节点返回的值大于beta值发生也要剪枝。**

* **伪代码**

function minimax(node, depth, a, b)

if node is a terminal node or depth = 0

return the heuristic value of node

if the adversary is to play at node

let b := +∞

foreach child of node

b : = min(a, minimax(child, depth - 1, a, b))

if b <= a

return b

return b

else { we are to play at node }

let a := -∞

foreach child of node

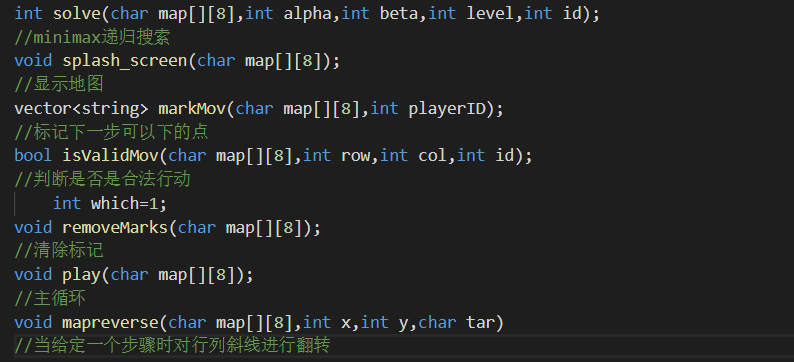
a : = max(b, minimax(child, depth - 1, a, b))

if a >= b

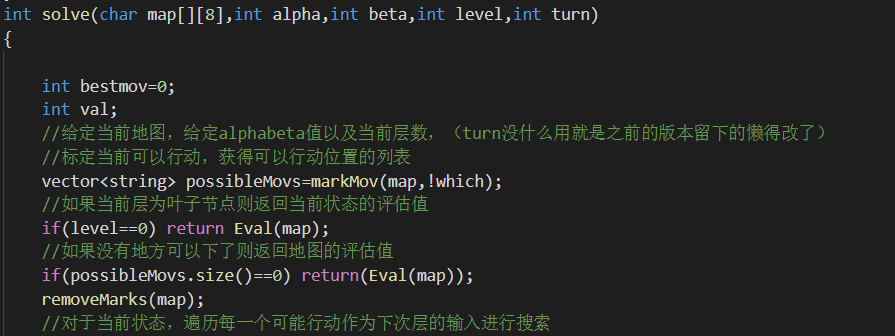
return a

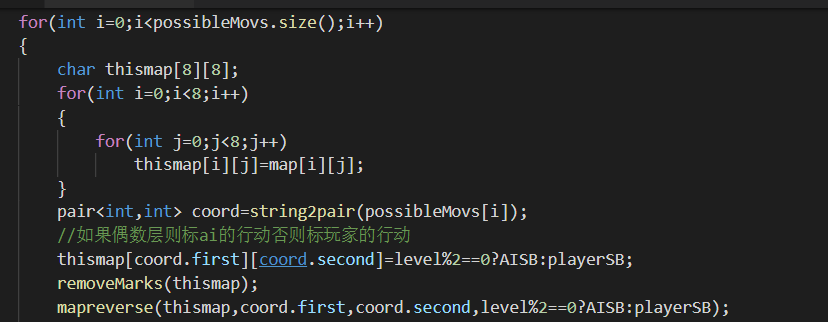
* **关键代码截图**

本次实验的关键部分是alphabeta剪枝以及minimax搜索算法，至于黑白棋的实现代码较长如果全部贴上来会影响对于核心部分的表达，因此只给出函数的声明没有定义部分。关键代码截图部分将只包括minimax的递归搜索以及评估函数部分。

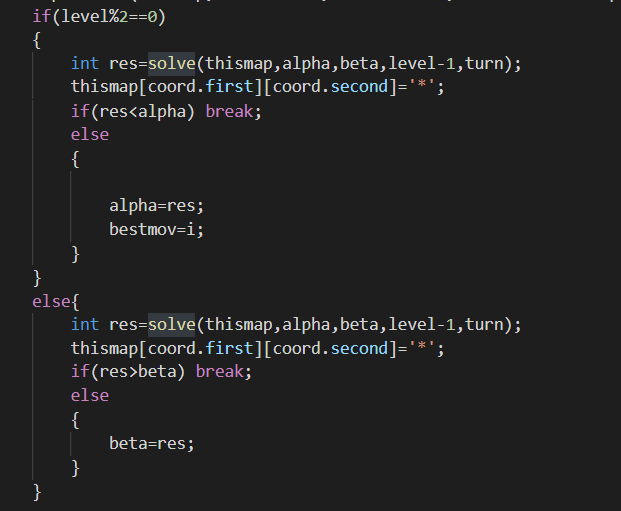


搜索的策略：根节点记为极大值层，输入为一张地图，首先判断当前AI方的可行棋状态，然后对于每一个状态递归的在下一个极小层搜索，取几个可能性中最大的取值，极小值层进行相反的操作，当到达搜索深度或者无路可走的时候返回当前棋局的估值，在根节点层选最有利的层标记地图并返回。

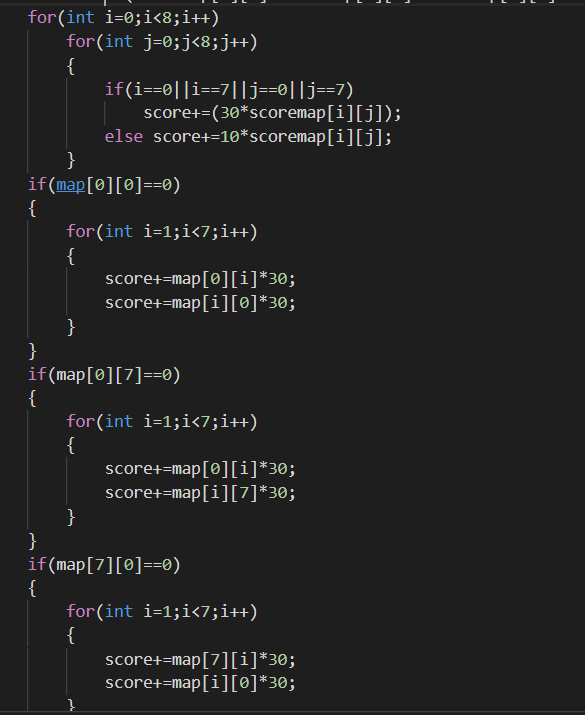
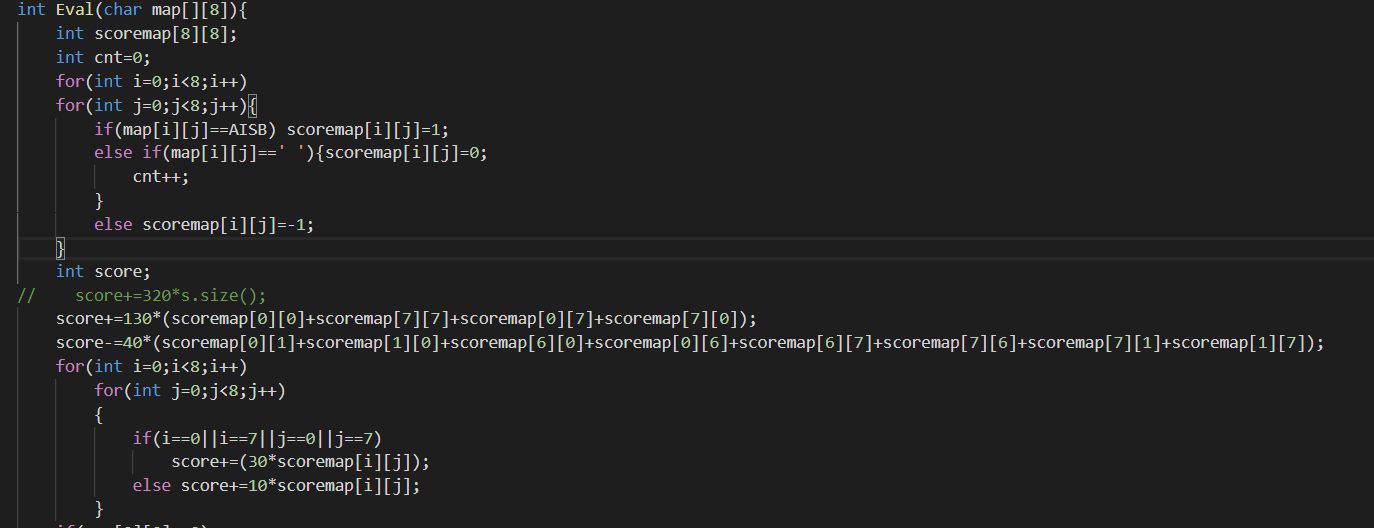




ALPHABETA剪枝

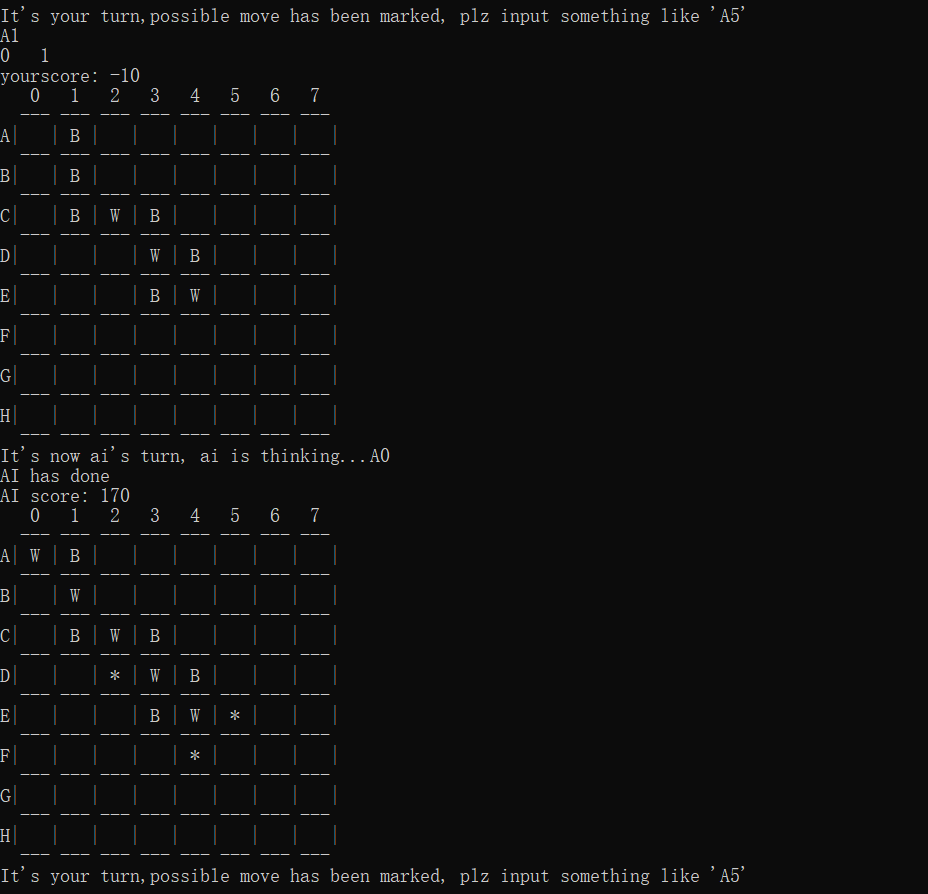


评估函数的思路是角的权值设的比较大，如果角没被占领边的权值也比较大，对于恶劣的位置设置负权值。因为四个角落很明显如果占领就不会被改变并可以以此为根基扩展其他的点，可以大大提高自己的胜率，而某些位置如A1，1A，B1如果某一方占领了这些点很大的概率对方会占领角落所以是恶劣的点置负权值，如果一方已经控制了某个角落那么己方在相邻的边上的控制权就会大大提高，而对于对方来说继续争夺这条边上的点的意义就不是很大了，因为对方有很大的概率会将这个点翻转过来，因此当角落被占领时就忽略一部分边的影响。

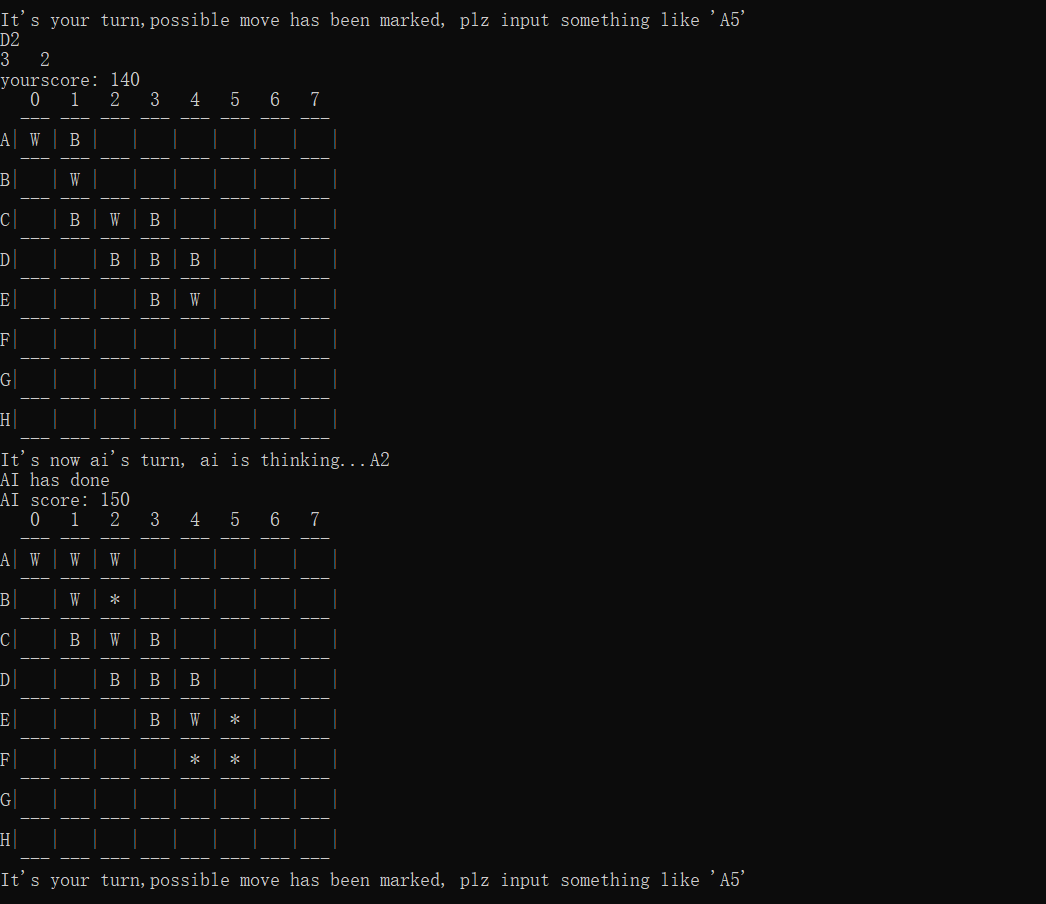


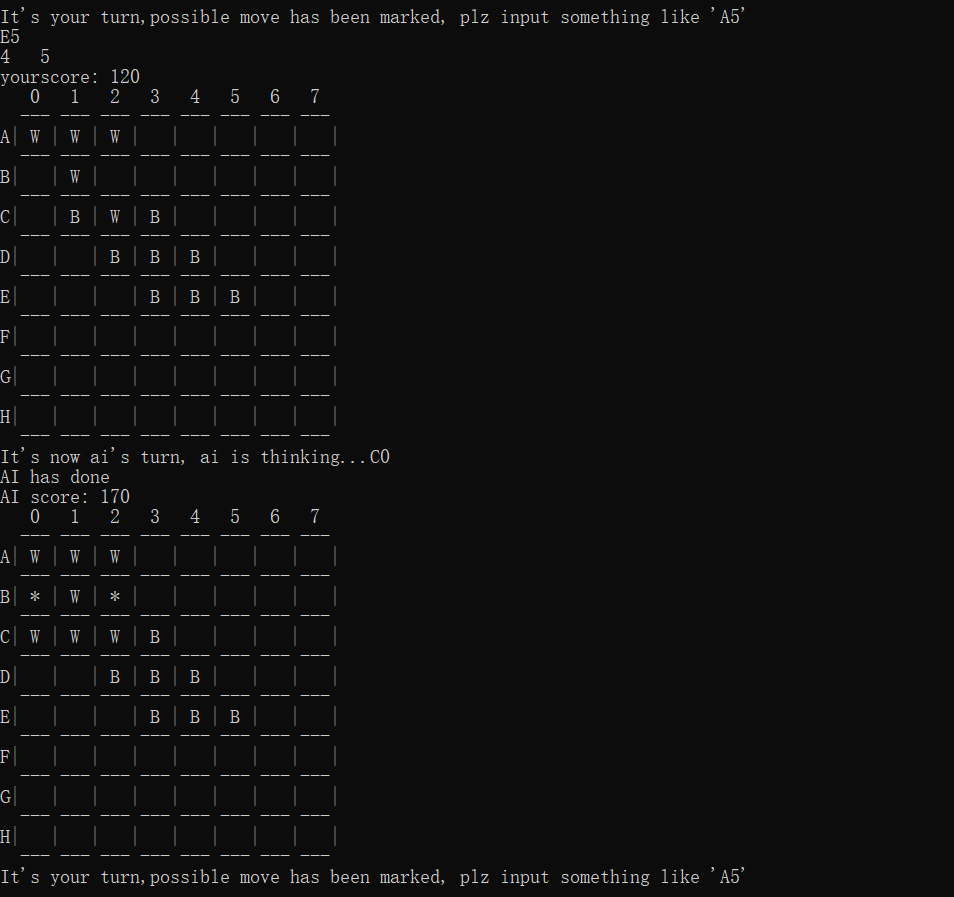
结果分析（三步）：

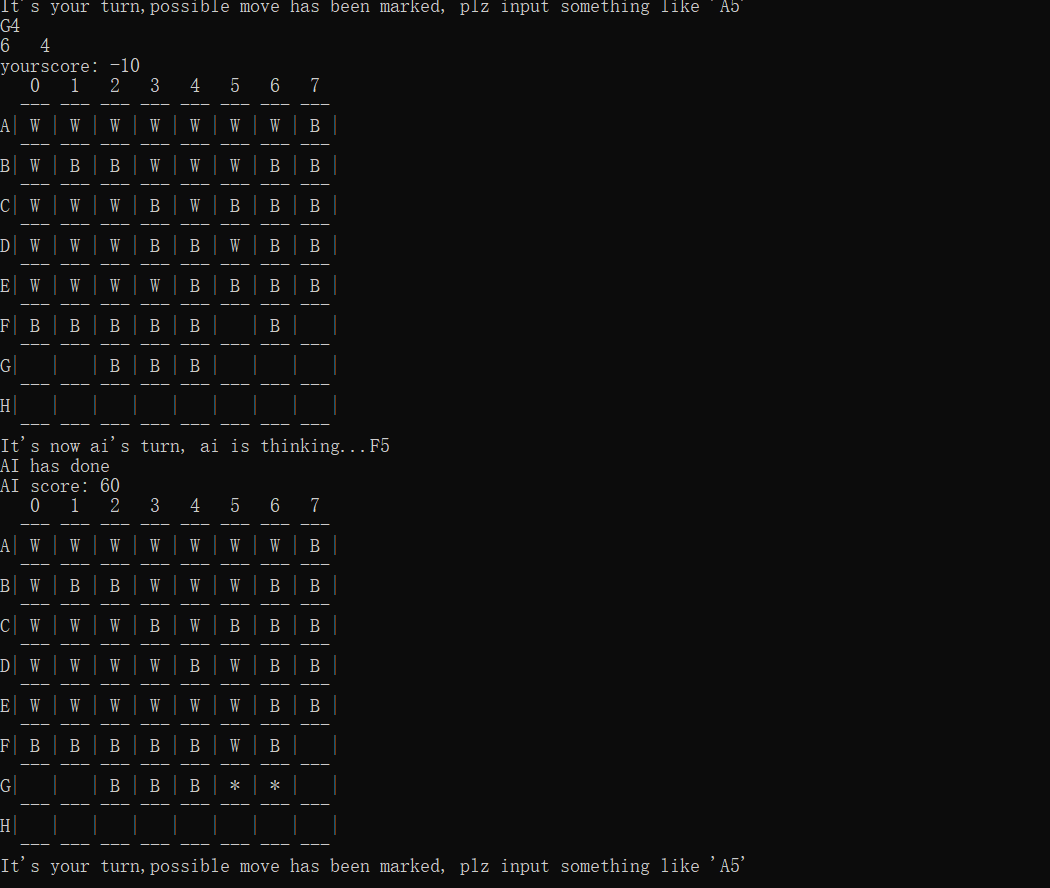
我是黑棋电脑为白棋，我下了A1估值为-10电脑占据了角落并觉得自己优势很大



我在中间下了一步削弱了一点电脑的优势但是它又加大了这个优势





n轮交锋之后

又n轮过后即便我占据了下面三个角还是输了。

