**程序报告**

学号： 3180105438 姓名：贺情怡

1. **问题重述**

（简单描述对问题的理解，从问题中抓住主干，必填）

====================================================================

**利用给出的board.py，使用蒙特卡洛树搜索算法来完成黑白棋AI。**

**AI需要完成的功能：**

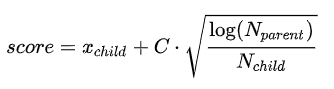
1. **在当前棋盘状态下，选择一个合法且在算法上最优的落子位置，作为返回值**
2. **搜索及决策时间不超过一分钟，若无合法位置则返回None**
3. **在游戏结束（即双方均无合法落子位置）时，尽量最大化己方与对方的棋子数量差**
4. **设计思想**

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

====================================================================

**采用了蒙特卡洛树搜索算法。**

**该算法利用上限置信区间算法的思想对搜索过程加以改进，其中有超参数C可以影响搜索的效果。**

**对于一个子节点，其奖励期望的上界可以写作****其根号下的常数2被合并到C里面了。可以看出，C越大，程序越倾向于探索访问次数偏小的节点。**

**在搜索过程中，每一次“采样”都有四个步骤：选择，扩展，模拟和反向传播：**

1. **其中选择主要受到UCB函数中C值的影响**
2. **扩展完全随机**
3. **模拟时由于黑白棋合法落子位置与当前局面的相关性非常大，没有找到随机以外的合适方式进行落子（基于当前局面的贪心甚至不如随机算法）**
4. **反向传播时更新的收益分数也是可以人为影响算法效益的部分。由于board.py提供了分数差的信息，可以用分数差的相关函数作为奖励收益（此处采用了分数差\*k，k为人为规定的系数）**

**但是根据UCB的score函数组成，其实能发现k如果只是作为乘上去的系数，本质上就是C，不过添加一个k可以方便调整也更直观而已。**

**然后就是搜索次数可以对搜索效果产生影响了，由于给出了一分钟的落子时限，在提交的时候应该会顶着时间上限吧（笑）**

1. **代码内容**

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

====================================================================

选择：

扩展：

模拟：

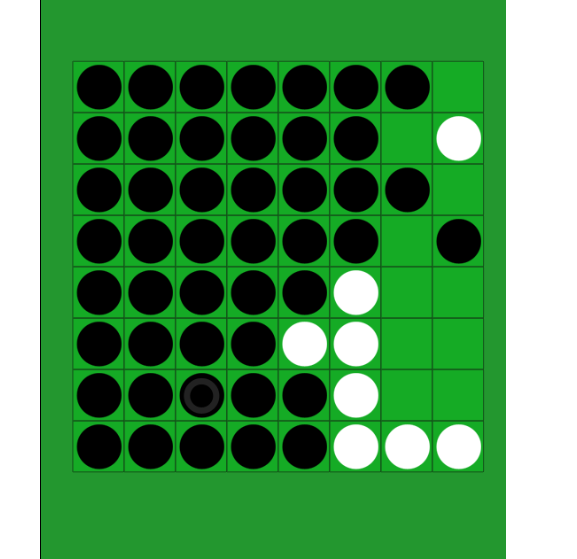
Back Propogation：

1. **实验结果**

（实验结果，必填）

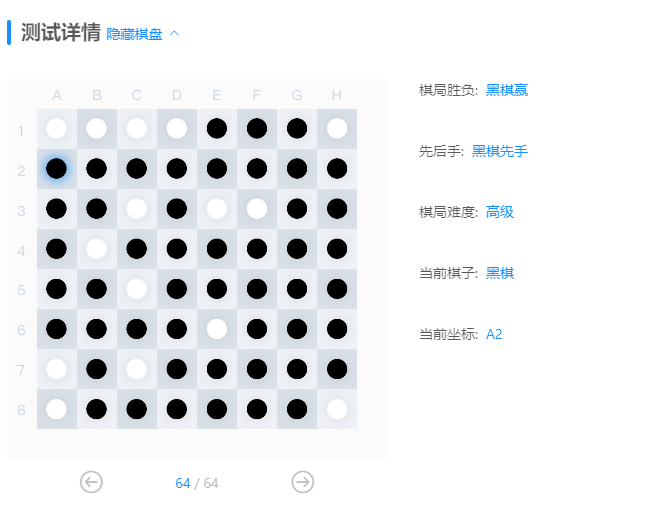
====================================================================

下图为与网站（http://www.7k8k.com/h5/3906\_swf.html）的黑白棋AI对战结果，白子为网站AI，黑子为蒙特卡洛树搜索结果：



====================================================================

下面是采样时间25秒，超参数C为1.4，k为0.4的AI与测试平台上提供的棋局对战的结果，可以看出效果很好且胜负情况较为稳定：

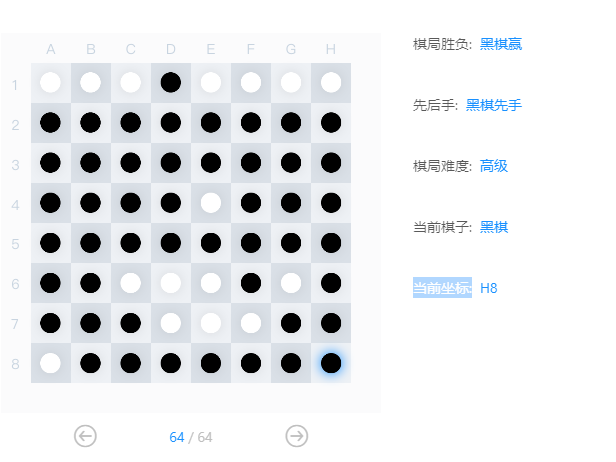


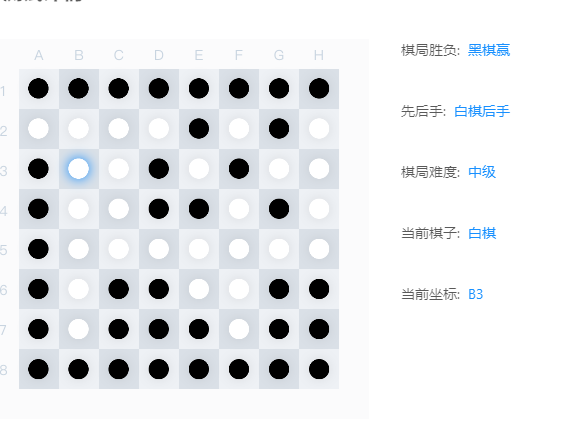




====================================================================

k=0.2，运行时间55秒则不那么稳定：





1. **总结**

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

====================================================================

因为很少使用python写长代码，这次试验挑战还挺大的（尤其是需要记录当前棋局状态作为树的节点，操作了很久才成功把棋盘hash到词典里），从vim切换到vscode之后，debug倒是方便了很多。

参考了这篇蒙特卡洛树解决五子棋的blog（<https://www.cnblogs.com/xmwd/p/python_game_based_on_MCTS_and_UCT_RAVE.html>），它的树节点是（color，position），并没有记录棋局状态，只是记录了落子位置，想必会有一定误差，但对于五子棋的大棋盘来说应该更加合理。

这份代码也没有把采样的四个步骤分开，虽然可读性较差，但他这个写法也非常巧妙（虽然对于未探索完但探索了一部分的节点的处理有点小bug），直接找出整条从根到叶子的树链，模拟的随机可以当成select的特例，而extend只需要对第一个随机选择的节点特殊处理即可。

也参考了这个黑白棋教程的report部分（<https://github.com/MolinDeng/Othello-MCTS>），主要是对照和自己的理解是否有出入。

在最后的测试阶段中，我感受到人工选取的超参数对AI效果的影响有多大。

在选取k=0.4之前还用了很久k=0.2的算法，但哪怕运行时间提升到55秒，也很明显感受到它的稳定性较差，受随机下棋影响很大，猜测是采样结果分数的影响过小，相当于C的取值太大，导致其一直无法收敛到较优的解上。