

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 人工智能 |
| 姓 名： | 王俊 |
| 学 院： | 海洋学院 |
| 专 业： | 海洋工程与技术 |
| 学 号： | 3170100186 |
| 指导教师： | 吴飞 |

2021年 5 月 9 日

**Lab3—黑白棋AI算法**

课程名称： 人工智能 实验类型： 综合

实验项目名称： 黑白棋AI算法

学生姓名： 王俊 专业： 海洋工程与技术 学号： 3170100186

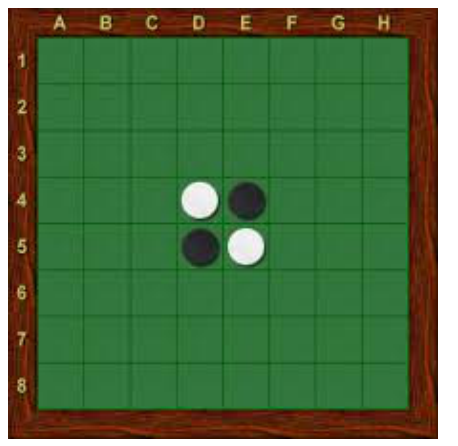
同组学生姓名： None 指导老师： 吴飞

实验地点： 曹四-503 实验日期： 2021 年 5 月 9 日

1. 问题重述

* 背景：

黑白棋 (Reversi)，也叫苹果棋，翻转棋，是一个经典的策略性游戏。一般棋子双面为黑白两色，故称“黑白棋”。因为行棋之时将对方棋子翻转，则变为己方棋子，故又称“翻转棋”(Reversi) 。棋子双面为红、绿色的称为“苹果棋”。它使用 8x8 的棋盘，由两人执黑子和白子轮流下棋，最后子多方为胜方。



* 具体实验要求：

A. 使用“最小最大搜索”、“Alpha-Beta 剪枝搜索”或“蒙特卡洛树搜索算法”实现

miniAlphaGo for Reversi（三种算法择一即可）。

B. 使用 Python 语言。

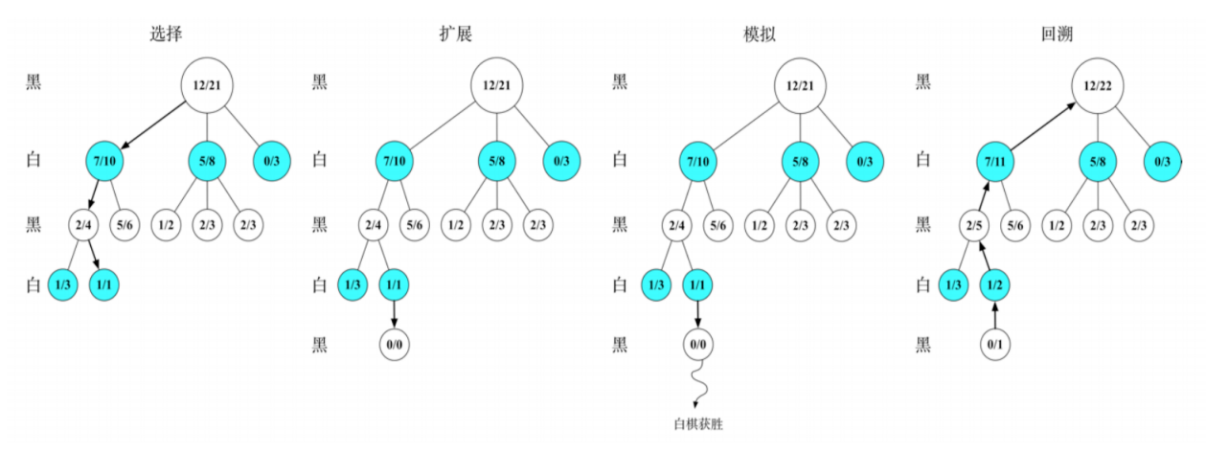
C. 算法部分需要自己实现，不要使用现成的包、工具或者接口。

1. 设计思想和代码内容

**2.1蒙特卡洛树搜索**

蒙特卡洛树搜索分为 4 个部分：选择、扩展、模拟、回溯。

* 选择：指从根节点开始，选择连续的子节点向下至叶子节点 L。
* 扩展：指除非任意一方的输赢导致游戏结束，否则 L 会创建一个或多个子节点。
* 模拟：从 L 的子节点中随机布局。
* 回溯：使用布局结果更新从 L 到根节点路径上的节点信息。



对于任一节点的信息，使用字典格式 dictionary 存储，

key ：从根节点到当前状态的路径 route，格式为字符串（如”A1B7C3D5”表示行棋顺序为 A1-B7-C3-D5）

Value：list [a，b] a 为已获得的奖励 reward，b 为该状态的总访问次数。

|  |  |
| --- | --- |
| **S** | **状态集** |
| A(s) | 在状态s能够采取的有效行动的集合 |
| S (v) | 节点v所代表的状态 |
| a (v) | 所采取的行动导致到达节点v |
| f：S x A -> S | 状态转移函数 |
| N(v) | 节点v被访问的次数 |
| Q(v) | 节点v所获得的奖赏值 |
| △(v,p) | 玩家p选择v得到的奖赏 |

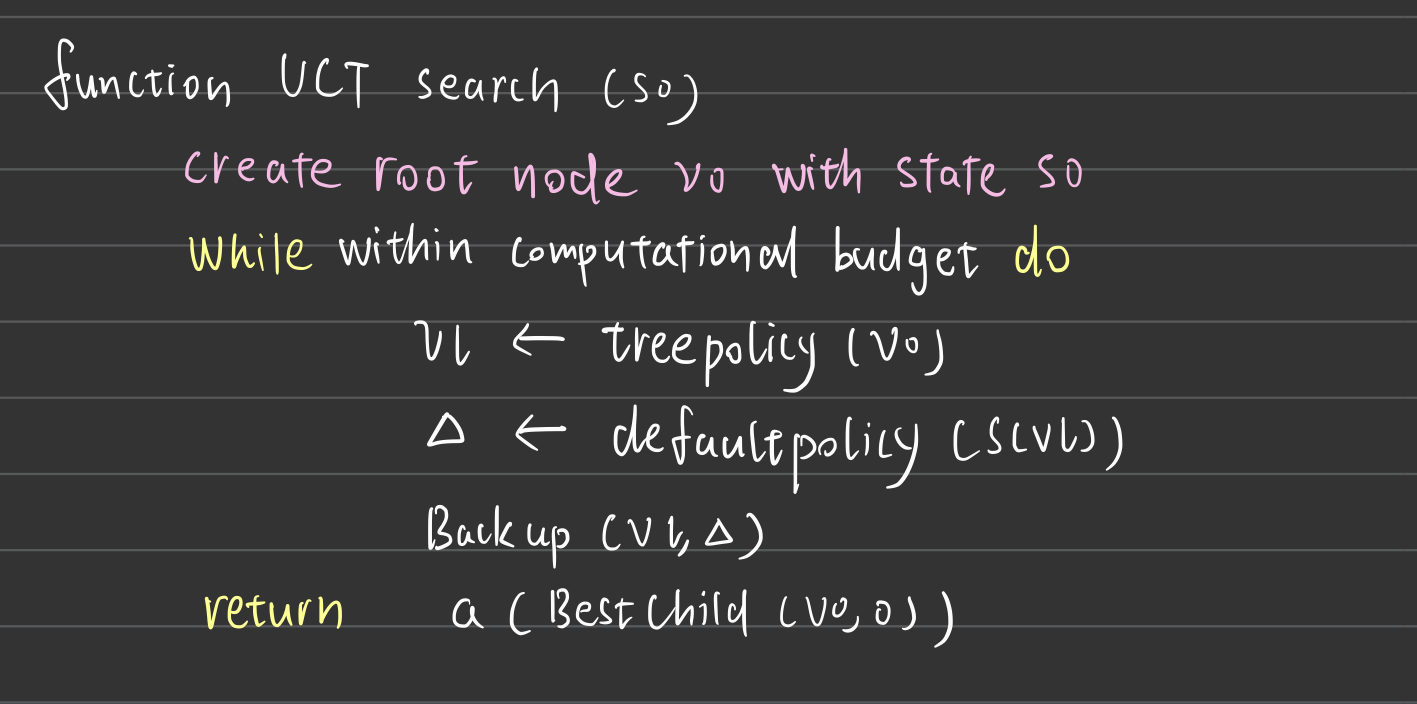
以下表格是算法伪代码的符号：

**2.2 搜索树算法 UCTSearch**

首先是实现整个 MCTS 算法的主体算法 UCTSearch，此外我还增加了一些跳出循环的条件: ①选举出来的节点已经是叶节点

②已经遍历到预先设定的最大层级

③时间已消耗至 20s 以上(一局时间为 60s)。

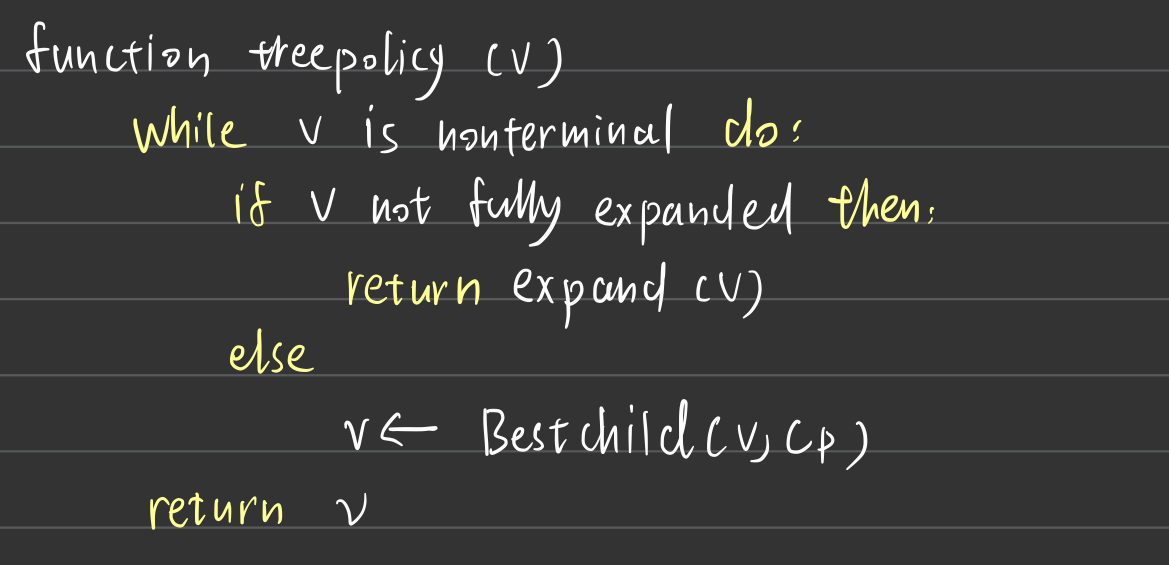
伪代码如下： 

* **选择算法 tree\_policy**

首先获取目标棋局的所有合法招式，如果某一个招式对应的棋局未被搜索过，则调用

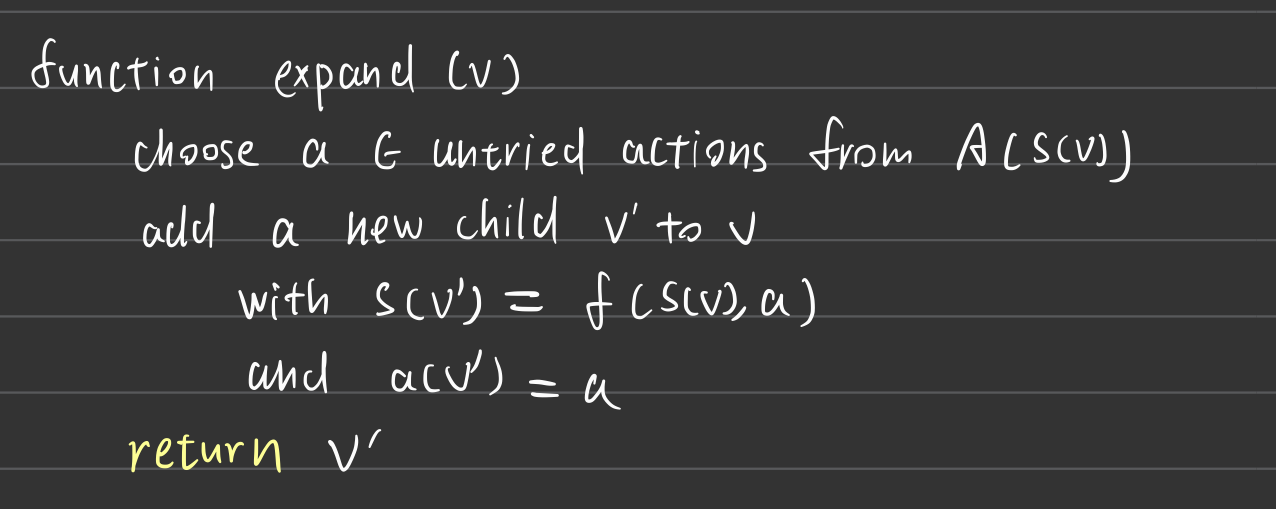
expand()并返回该招式对应的棋局，否则选择所有招式中当前最优的招式对应的棋局，继续

调用 tree\_policy()，进行递归，直到达到预先设定的深度。选择算法伪代码如下所示：



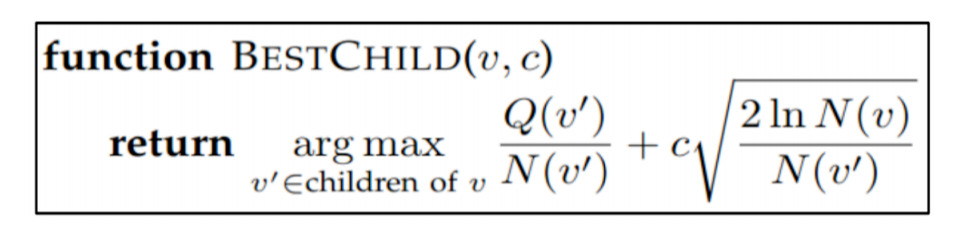
* **扩展算法 expand**

扩展节点，先在子节点中选择一个未被尝试过的，将这一新的节点更新至搜索路径中， 并为这一新的状态赋予初始值[0,0]，添加到字典 dic 中，伪代码如下：

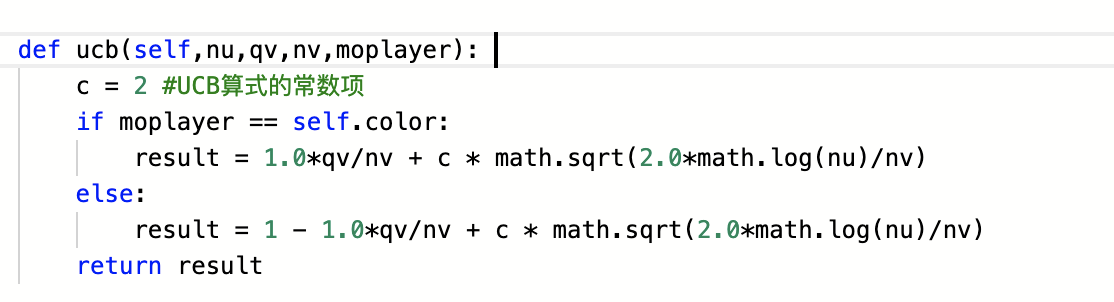


* **最佳子节点算法 best\_child**

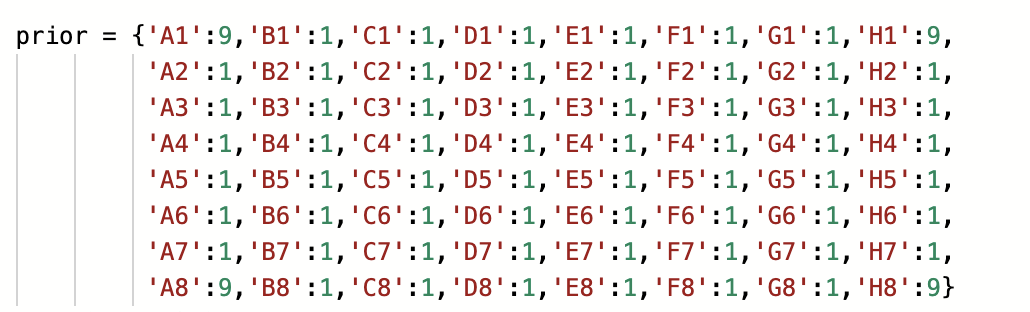
**实现计算 UCB 得出估值最高的子节点的算法，伪代码如下。这里考虑了己方节点和对方节点对于估值选择的不同决策：己方节点将选择子节点中估值最大的(Max)，对方节点将选择子节点估值最小的(Min)。**



**Code：**



对于最终根节点选择行棋的策略，有另一个函数 best\_action。创建了一个优先度矩阵 prior，标记了棋盘上各个位置的优先程度，默认四个顶点为 9，其余为 1。在其中增加了对行棋后果的判断：若行这步棋之后能够促使对方占领四个顶点，则降低该位置的优先度至 0.1。最后计算子节点的 UCB 并乘上优先度，得到最适合的行棋策略。



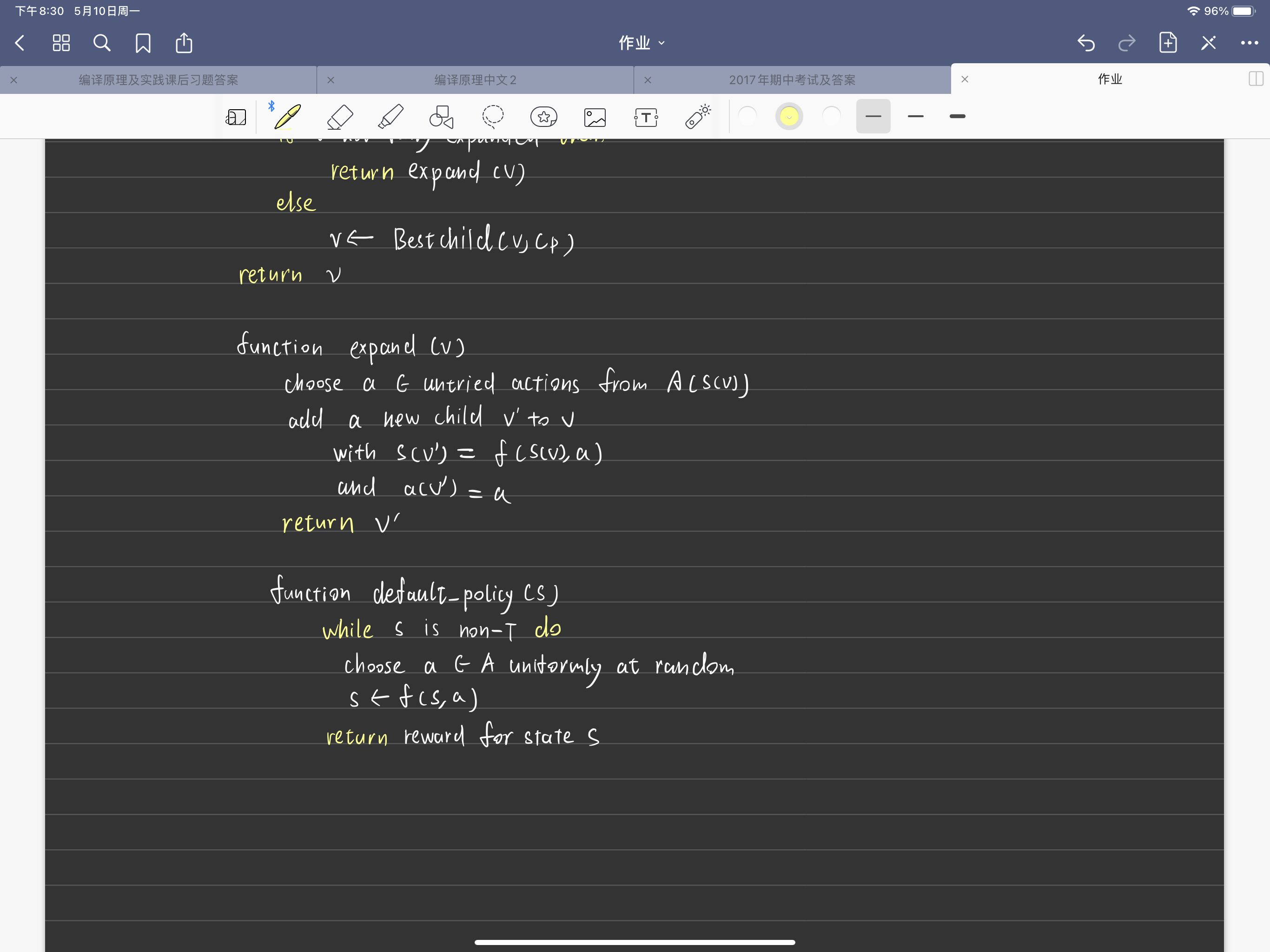
**2.3MCT代码实现：**

MCTS 的核心代码结构如下：



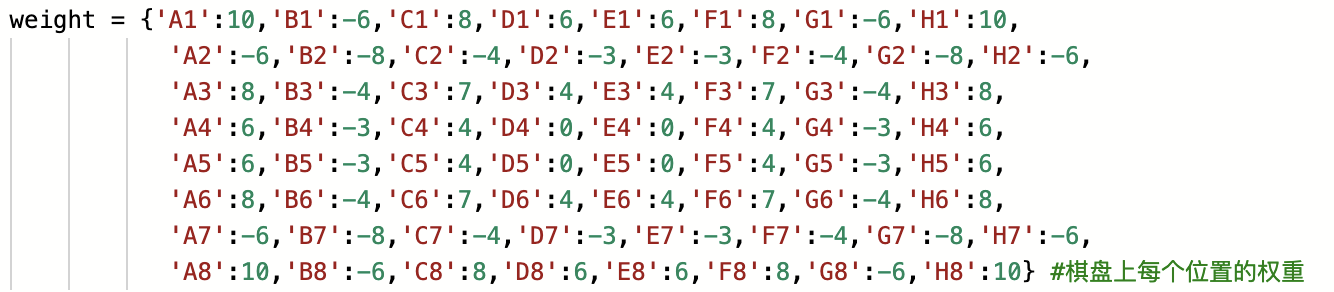
**2.4 模拟算法 default\_policy**

**伪代码：**

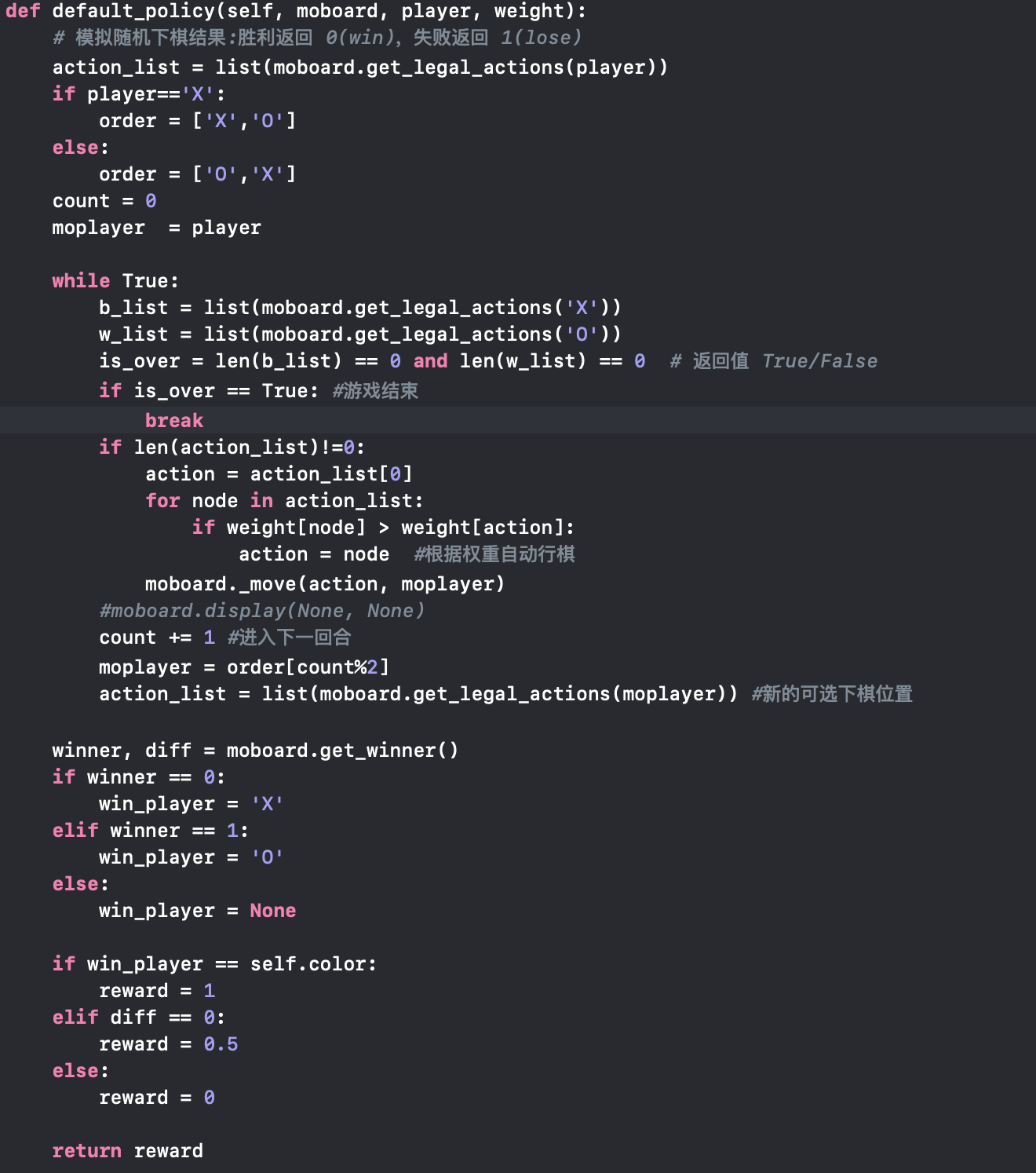


在本算法中，模拟过程的每一步决策，抛弃了随机策略，而采用参考当前棋盘每个位置

的权重，选择权重最大的有效棋步来下，如此能够获得更为准确的终局结果。将棋盘各个位

置的权重设置如下： 

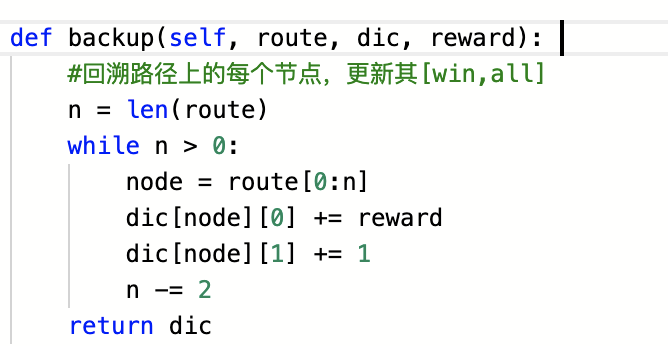
Code：



**2.5 回溯算法**

对于模拟终局的结果，作为 reward 需要回溯更新每一个祖先节点，遍历方法时每次把

route 减去一个位置作为 key 更新字典的值：



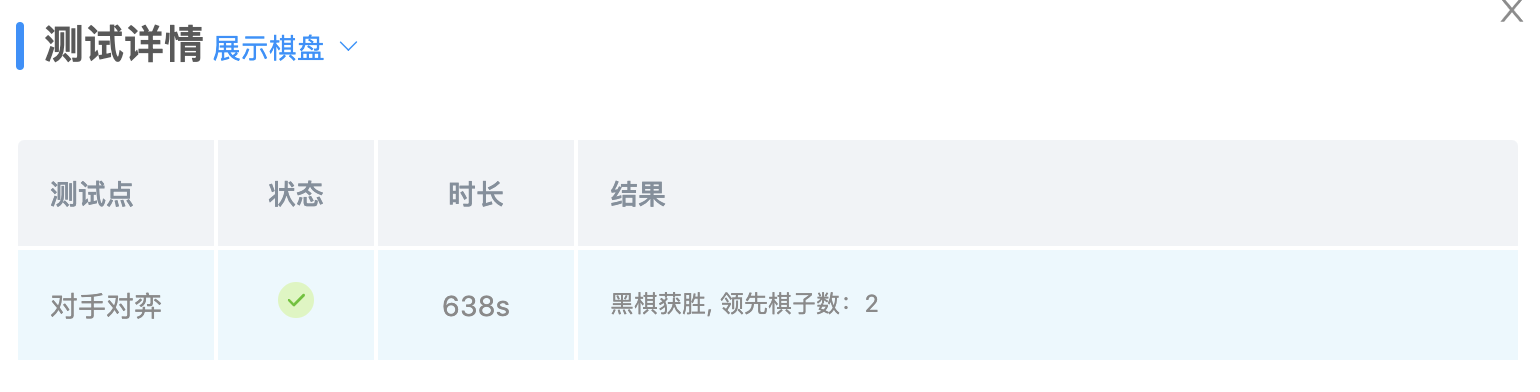
1. 实验结果

3.1. 对战初级，黑棋先行：



3.2. 对战中级，黑棋先行：



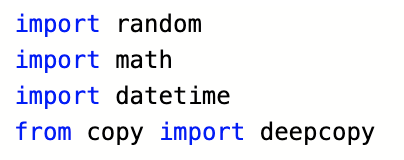
3.3. 对战高级，黑棋先行：

可以看出，此时的时间较长，还有优化的空间

1. 总结

**4.1. import random：**

需要记住在前面import包的时候import自己需要的包：



**4.2. 模型进一步优化**

模型明显训练的时间过长，不够简单，应该想办法近一步优化，使得其实时反应

时间更短，从而增加其可用性。

**4.3. 算法评价**

优点：蒙特卡洛树搜索算法比 alpha-beta 剪枝搜索和 minimax 更复杂，效果较好。

缺点：优化程度还有很大的改善空间，甚至可以利用强化学习来优化参数。