1.用途：塑造两个**AI棋手自己对弈**，为pv网络收集训练数据

2.概述：每个树节点对应一个**棋盘状态**。

从根节点到叶节点的逐层下降，实际就是两个棋手下棋直到分出胜负或者超时。

3.树**节点属性**：有5个。

〇对应的棋盘状态**broad**

①已经被访问过几次**Ni**

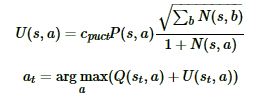
②所属棋手胜利了几次**Wi**

③属于哪一层，奇数层则为棋手1的胜率，偶数层为棋手2的胜率，**棋手方**

④对于父节点，自身的**UCT值**

4.UCT值有什么用？衡量下一步应该走哪个节点。一个朴素的衡量标准是**wi/ni**，但是只考虑了经验胜率，没有考虑到未访问的节点的潜力。

因此加上一项**c\*sqrt(lnNi/ni)**，i号子节点被访问的越多，代表它的潜力越少

然后再给UCT加上一项**Q(s,a)**，该网络预测项，是对当前棋手一方各个策略的价值的预测，**网络默认输出是白方视角**，黑方视角的Q2只需1-Q1一下就行，然后得出最应该走的策略，然后把**at**加入**epsilon-greedy**，这便是AI棋手的策略

注意：**在搜索树中，每次轮到黑棋走时，走对黑棋最有利的/概率最大的；轮到白棋走时，走对黑棋最不利的/概率最小的**

5.至此我们大致知道了蒙特卡洛树如何对弈，再来总览一下生成该树的步骤

①选择&扩展：从根节点开始，

根据上述**epsilon-greedy**动作a往下找子节点，如果节点已被充分扩展（即每种落子位置都已经被穷尽，被创建出对应子节点），那么就继续根据Q+U最大的动作a往下找子节点...这样直到**找到一个没有充分扩展的节点**，**创建**最大动作对应的子节点

等价表述：根据上述**epsilon-greedy**动作a往下找子节点，如果子节点存在，则将其设为当前节点，继续Q+U最大的动作a往下找，**直到子节点不存在**，那说明找到了新的下法，**终止循环，创建该子节点**

②模拟：把这个子节点的broad拿来，放入**真实环境下一局**，策略为**rollout**而非根据UCT，质量堪忧

③反传:得到该子节点的一次模拟结果，**根据胜负结果更新**其父、父父节点直到根节点的ni/wi/uct值