mcts\_alphaZero.py负责实现一个AI玩家，包含了它的训练过程，和面对一个确定局面时输出的动作。

AI看到一个棋面，根据直觉选出最可能赢的很多走法，并且真实+直觉模拟了2000局，再来决策下一步怎么下。

1.在一个棋盘局面下，这个玩家有一个MC树，根据策略价值网络得到两个先验知识：

①选择所有合法下一步的先验概率prior\_p,

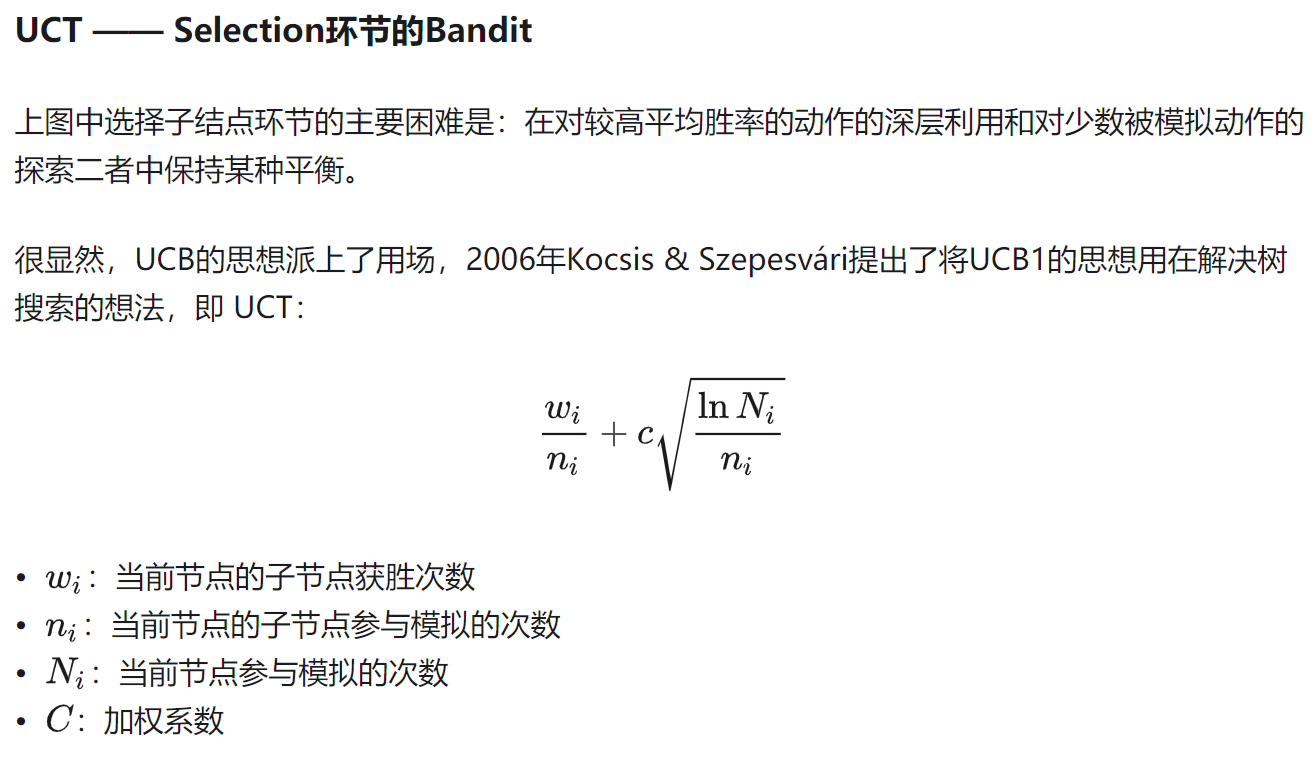
②当前节点的预测价值leaf\_value

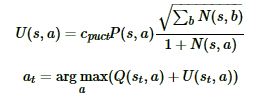
2. 进行n\_playout次的MCTS(可以视为训练蒙特卡洛树)。

其中，一次MCTS即一次playout（得到一次棋局结果）

包括：

步骤①从当前局面的树的根节点开始计算子节点的puct上限值U(s,a)+Q(s,a)。不断选择puct上限值最高的子节点（一共有boarad.available.size个节点选择，就是可选选项），直到抵达没有穷尽可选子节点的节点（没有分支的节点）

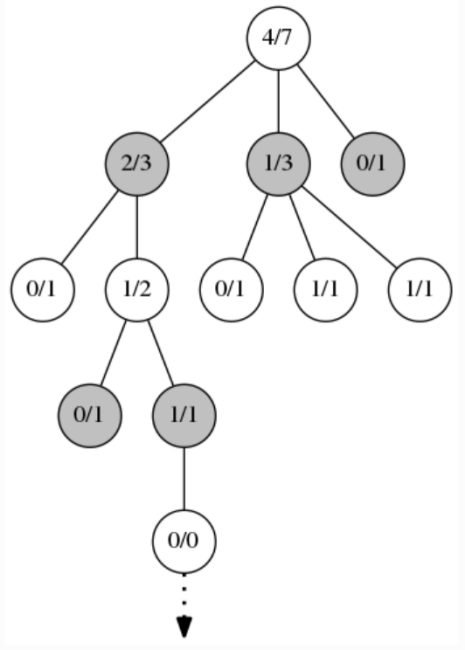
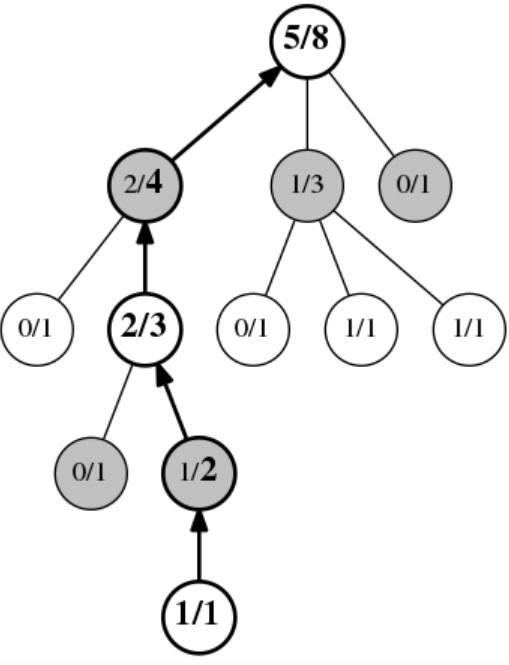




步骤②根据策略价值网络给出的prior\_p和board.availble扩展叶节点的合法子节点并赋予先验概率

步骤③如果已经到达棋局结束对应的叶节点则得到该叶节点的实际输赢价值，否则以leaf\_value作为价值估计

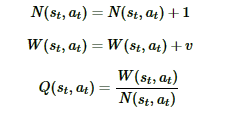
步骤④将叶节点的价值递归反向传播，更新该叶节点所属的整根枝条的属性。

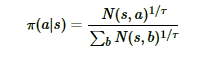
一个节点的属性：【W/N】=

【此节点对于此时player的胜利期望，或者只是该节点方的累计胜利次数/被mcts playout访问的次数】

更新公式：（其中Q为节点圆圈中的./.）



3. MCTS搜索完毕后，在MCT的根节点s基于直接分支节点的访问次数算出下一步动作的策略：



（其中τ=temperature决定探索与贪心的程度。当τ越接近于1时，策略π越接近于MCT的原始采样，是soft的策略。当τ越接近于0时，策略π越接近于贪婪策略，即只选择最大访问次数N所对应的动作）

在具体代码实现中，

===softmax())

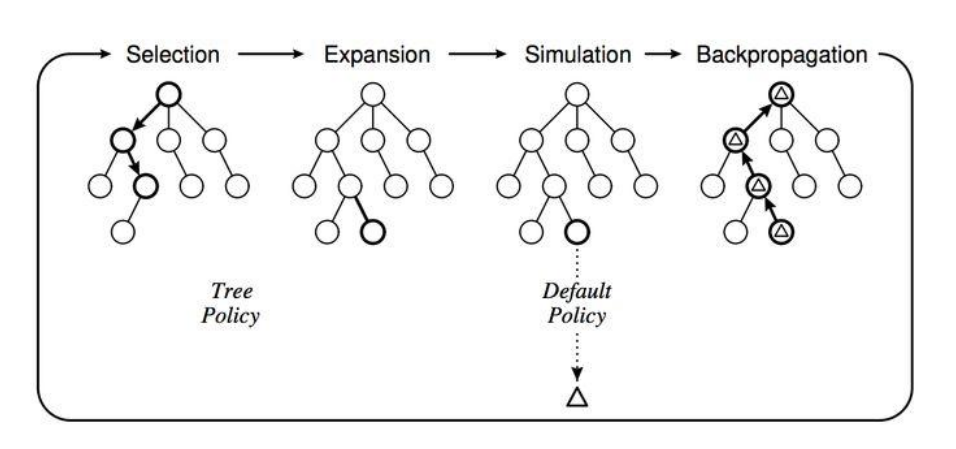
4.根据混合策略π(a|s)随机选出下一步移动move，更新蒙特卡洛树交给下一局面的Player。

情况①：如果在自我对弈，那么在π(a|s)的基础上还要加入0.25的dirichlet噪声用于继续探索。更新蒙特卡洛树只需要删去原来的根节点，把move到达的直接分支节点设置成根节点，继承这一分支节点的子树。

情况②：如果在和其他智能体下棋，不探索，直接依照π(a|s)分布选动作。更新蒙特卡洛树需要推倒重来，让下下步的自己从一个根节点从头生成整棵树，因为对手的策略分布与自己不同，数据可能需要IS才能继续用，而且分支选项的分支选项只含有很少信息，继承子树的子树没什么用了。

排除异常：如果棋盘满了，打印”board is full”然后退出。

二. 对比原始MCTS:



有了选择下一步的先验概率prior\_p、当前节点的预测价值leaf\_value。

在selection阶段puct的计算用到了p和v，

省去了一些不必要的探索

参考文章[PR Reasoning Ⅱ：Bandit问题与 UCB / UCT / AlphaGo - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/218398647)