人工智能大作业-蒙特卡洛树搜索树实现三子棋游戏-算法基本概念

笔记本： 我的第一个笔记本
创建时间： 2019/6/4 14:47 **更新时间：** 2019/6/7 15:02
作者： 1735512161@qq.com
URL： https://mp.weixin.qq.com/s/nbTkr0PImlXUSYl91HD91Q

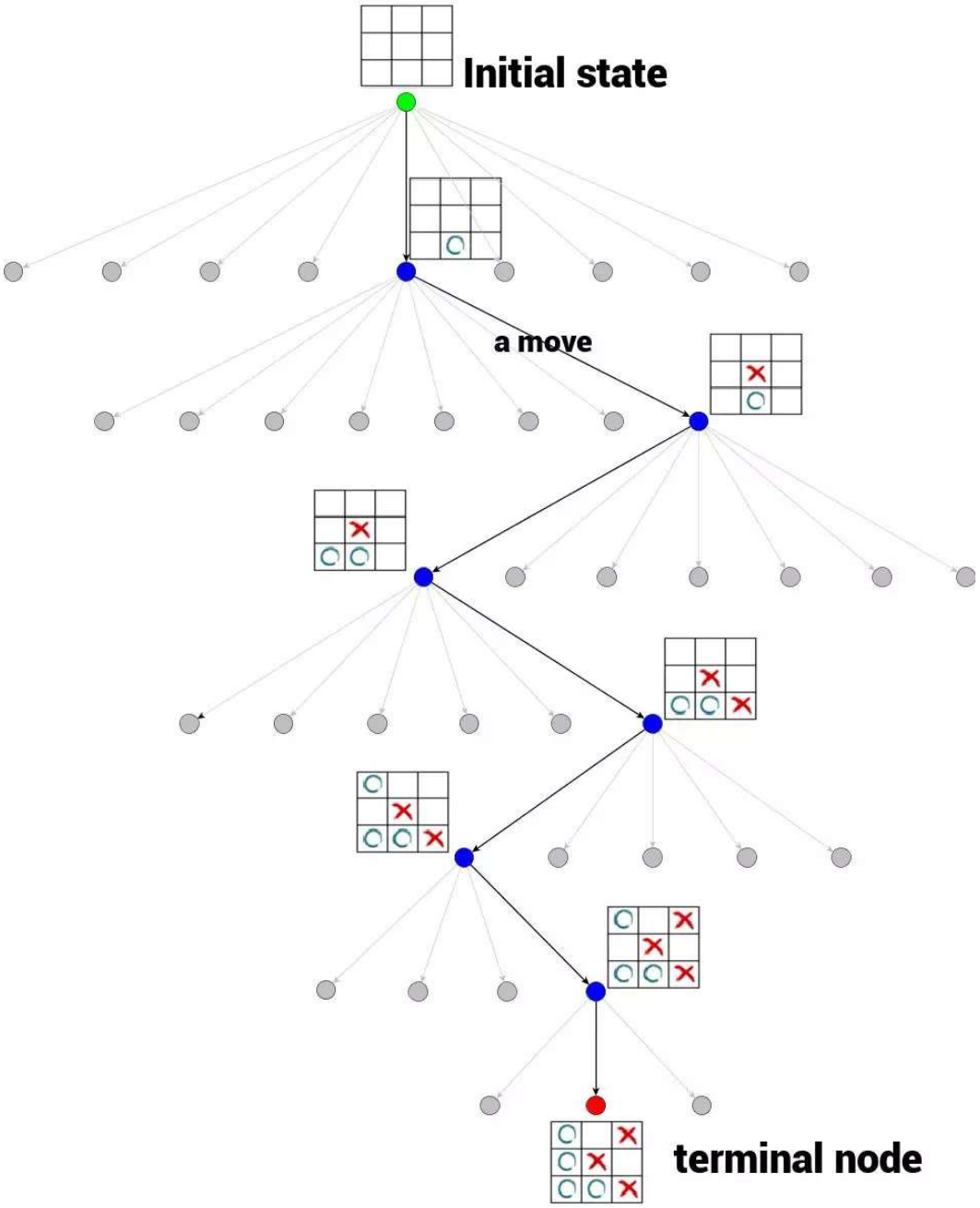
使用蒙特卡洛搜索树实现三子棋游戏-基本概念

博弈树的基本概念

博弈树

博弈树是一种树形结构，其中每一个节点表征博弈的确定状态。从一个节点向其子节点的转换被称为一个**行动**，节点的子节点数量被称为**分支因子**，树的根节点也就是博弈的**初始状态**，**端节点**也就是叶子节点，即没有子节点的节点，表示博弈的最终状

态，总结博弈结果。

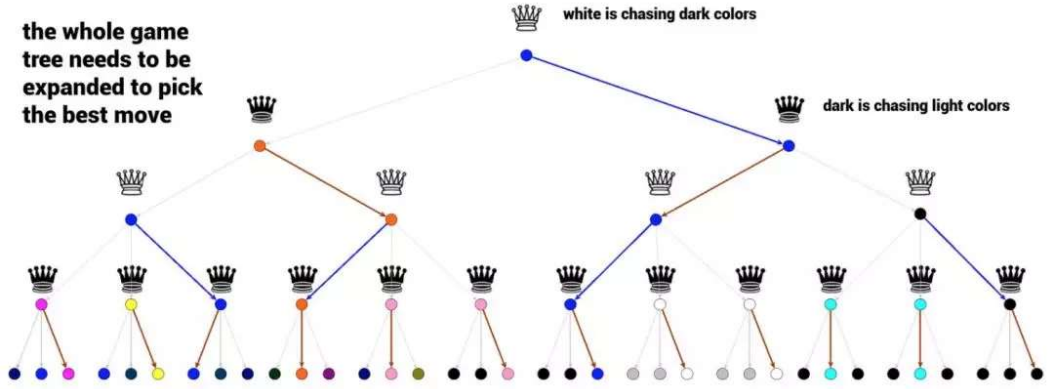


- 在顶部，你可以看到树的根节点，其表征了井字棋博弈的初始状态，即空白棋盘（标记为绿色）
- 任何从一个节点向另一个节点的转换被称为一个行动
- 井字棋的分支因子是变化的，它依赖于树的深度
- 从一个根节点到一个端节点的树遍历表征了单个博弈过程

博弈树是一种递归的数据结构，因此当你选择了一个最佳行动并到达一个子节点的时候，这个子节点其实就是其子树的根节点。因此，你可以在每一次（以不同的根节点开始），将博弈看成由博弈树表征的**寻找最有潜力的下一步行动**问题的序列。在实践中很常见的是，你不需要记住到达当前状态的路径，因为它在当前的博弈状态中并不重要。

极小极大(min-max)策略

给定状态 s，并假定**对手正在尝试最小化你的收益**，你希望找到能**最大化你的收益的动作 a_i。**这正是该算法被称为极小极大的原因。我们需要做的就是展开整个博弈树，并反向传播由递归形式的规则得到的值。



上图中的博弈树展示了极小极大算法中的最佳行动选择过程。
白皇后希望博弈的结果尽可能的黑暗，而黑皇后希望博弈的结果尽可能的明亮。
每一个层级的选择都是极小极大判断的最优结果。我们可以从底部的终端节点开始，其中的选择是很明显的。**黑皇后将总是选择最明亮的颜色，然后白皇后将寻找最大的奖励并选择到达最暗颜色的路径**，等等。这正是基础的极小极大算法的执行过程。

极小极大算法的最大弱点是它需要展开整个博弈树。
对于有高分支因子的博弈（例如围棋或国际象棋），该算法将导致巨大的博弈树，使得计算无法进行。

alpha-beta 剪枝算法

alpha-beta 剪枝是提升版的极小极大算法，它以极小极大算法的形式遍历博弈树，并避免某些树分支的展开(**最优化剪枝**)，其得到的结果在最好的情况下等于极小极大算法的结果。alpha-beta 剪枝通过**压缩搜索空间提高搜索效率**。

蒙特卡洛树搜索的基本概念

蒙特卡洛树搜索的主要概念是**搜索**，即沿着博弈树向下的一组遍历过程。
单次遍历的路径会从根节点（当前博弈状态）延伸到没有完全展开的节点，未完全展开的节点表示其子节点至少有一个未访问到。遇到未完全展开的节点时，它的一个未访问子节点将会作为单次模拟的根节点，随后**模拟的结果将会反向传播**回当前树的根节点并更新博弈树的节点统计数据。一旦搜索受限于时间或计算力而终止，下一步行动将基于收集到的统计数据进行决策。

- 什么是模拟？
- 什么是已访问节点和未访问节点？
- 什么是展开或未完全展开的博弈树？
- 什么是反向传播？
- 反向传播中的统计数据是什么，在展开博弈树结点更新的是什么？
- 在搜索过程中，向下遍历是什么意思？如何选择访问的下一个子节点？

- 最后的行动策略到底是如何选择的?
- 算法什么时候终止?

模拟

模拟即**单次博弈策略**，它是一系列从当前节点（表示博弈状态）开始，并在计算出博弈结果后结束于端节点。
在模拟中**行动**的选择方法:
rollout 策略函数:

$$\mathrm{RolloutPolicy}: s_i \rightarrow a_i$$

该函数将输入一个博弈状态si ，并产生下一次行动的选择ai 。

节点类型

已访问节点:
节点进行了至少一次的评估，就是已访问节点。
未访问节点:
节点一次的评估都没有进行过，就是未访问节点。

完全展开节点:
一个节点的所有子节点全都是已访问节点，那么该节点就是完全展开节点。
未完全展开节点:
一个节点的子节点存在未被访问的，那么该节点就是未完全展开节点。

反向传播

将模拟的结果一层层地从**模拟结束后的节点**向上传播回去，到达当前博弈树的**根节点**。

反向传播是从子节点（模拟开始的地方）遍历回根节点。模拟结果被传输至根节点，反向传播路径上的每个**节点的统计数据都被计算 / 更新**。反向传播保证每个节点的数据都会反映开始于其所有子节点的模拟结果（因为模拟结果被传输回博弈树的根节点）。

节点的统计数据

反向传播模拟结果的目的是更新**反向传播路径**上所有节点v的**总模拟奖励q** 和 **总访问次数n**。

- q(v) 即总模拟奖励是节点 v 的一项属性，在最简单的形式中是通过考虑的节点得到的模拟结果的总和。
- n(v) 即总访问次数是节点 v 的另一项属性，表示节点 v 位于反向传播路径上的次数（即它对总模拟奖励做出了多少次贡献）。

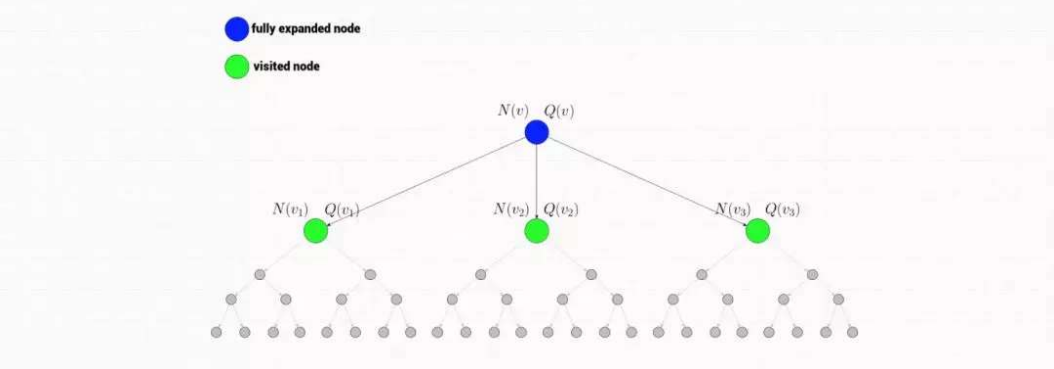
每个**被访问过的节点**都会保存这两个值，一旦确定了模拟次数，并且完成了之后，被访问的节点就保存了这两个信息。
q反应出了节点的潜在价值(总模拟奖励)
n反应出了节点被探索的程度(总访问次数)

博弈树的遍历

从根节点开始选择每个未被访问过的节点开始进行模拟，每次模拟都会反向传播，也就会更新一些节点的q\n信息，然后根节点即会被认为是经过了完全展开的。

但是接下来怎么做呢？现在我们如何从完全展开的节点导向未被访问的节点呢？我们必须遍历被访问节点的层，目前没有很好的继续进行的方式。

为了在路径中找到下一个节点，以通过完全展开的节点 v 开始下一次模拟，我们需要考虑 v 所有子节点 v_1, v_2, ..., v_k 的信息，以及节点 v 本身的信息。
现在我们来思考一下可用信息有哪些：



当前节点（蓝色）是完全展开的，因此它必须经过访问，以存储节点数据：它及其子节点的**总模拟奖励和总访问次数。这些值是为了最后一部分：树的置信上限（UCT）做准备。**

置信上限函数

通过这个函数可以从**被访问节点开始选择出下一个要继续遍历的节点**，这也就是蒙特卡洛树搜索算法的核心思想。

$$\mathbb{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\log(N(v))}{N(v_i)}}$$

找出UCT最大的节点，也就是找出当前要选择遍历的节点。

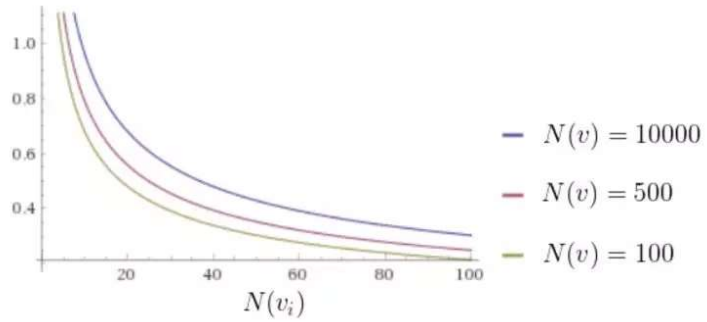
第一个组件: exploitation

$$\frac{Q(v_i)}{N(v_i)}$$

可以理解为赢 / 输率，总模拟奖励 / 总访问次数，即节点的胜率评估结果。
不可以只有前一个组件，因为该组件**只是**表示了当前已经被访问子节点的赢率，但是其实未被探索过(模拟)的节点并不一定不比它好，所以还需要有:
第二个组件: exploration

$$c\sqrt{\frac{\log(N(v))}{N(v_i)}}$$

exploration组件的形状：随着节点访问量的增加而递减，给访问量少的节点提供更高的被选中几率，以指引 exploration 探索。



参数：c
用于控制两个组件之间的权衡

终止蒙特卡洛树搜索

根据情况(游戏的数据规模、计算机的计算能力、实际应用场景)来进行终止搜索。

参考文档:

官方文档

机器之心的翻译文档

官方代码

参考代码的文档

参考代码

参考资料(理解代码)

待完成任务:

1. 继续调试bug1，再看看有没有其他bug
2. 自己写的java实现上传到github上去
3. 完成对代码解释的报告 和 需要上交的实验报告