《游戏开发与实践》

第十二次作业

内容： 人工智能对战算法设计

学号： 20051406

姓名： 朱宴宽

**目录**

[一、 博弈树 3](#_Toc103775307)

[概念 3](#_Toc103775308)

[二、 alpha-beta剪枝算法 3](#_Toc103775309)

[目的 3](#_Toc103775310)

[相关概念 3](#_Toc103775311)

[算法内容 4](#_Toc103775312)

[三、 我的设计 7](#_Toc103775313)

[（1）五子棋对战棋盘的特点 7](#_Toc103775314)

[（2）alpha-beta剪枝算法的不足 8](#_Toc103775315)

[（3）棋盘状态判断与对称轴寻找 9](#_Toc103775316)

[参考资料 10](#_Toc103775317)

# 博弈树

## 概念

博弈树是指由于动态博弈参与者的行动有先后次序，因此可以依次将参与者的行动展开成一个树状图形。

博弈树在AI领域有着广泛的应用，尤其是棋类游戏AI。在进行决策前，博弈树对每种决策及其后续可能带来的各种情况进行展开，最后选择其中的最佳决策，从而达到在博弈中获取最大收益的目的。

但一般而言，如果对每种决策及其后续各种情况都进行完全的展开，在效率和尽可能减小空间复杂度上都不能达到很好的效果——就15\*15大小的棋盘而言，第一次落子有225种选择，而每种选择之下还能展开224种第二次落子的情况。光是罗列到博弈树的第三层（无落子的根节点独占一层），便已经产生了255\*224+1=50401个节点。如果将后续所有的落子选择都罗列一遍，将会产生一个非常庞大的博弈树，也会带来很大的时间与空间耗费。

而alpha-beta剪枝算法可以对这种情况带来较好的优化，下面是搜集到的相关资料。

# alpha-beta剪枝算法

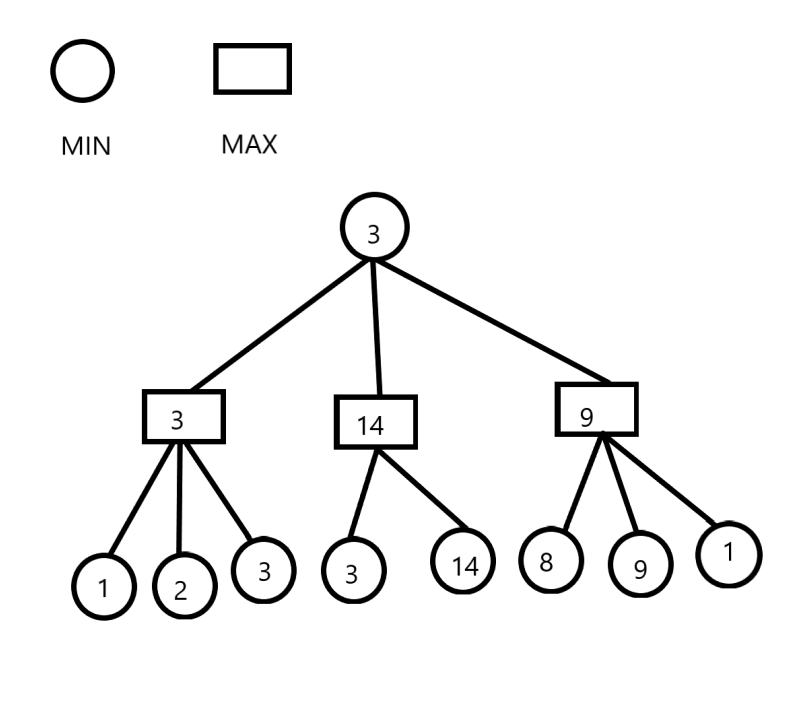
## 目的

alpha-beta剪枝算法用于裁剪博弈树中没有意义、不需要搜索的树枝，以提高运算速度。

## 相关概念

1. 极大极小树（max-min tree）

通常情况下，博弈双方轮流交替进行决策。在我方进行决策时，总是选择最有利于我方的方案，而对应在极大极小树中的这一层称为极大层；在对方进行决策时，总是选择对自身最有利的方案，而对对方本身最有利的选择也意味着是对我方最不利的选择，所以这一层称为极小层。极大层、极小层在极大极小树中轮流出现。



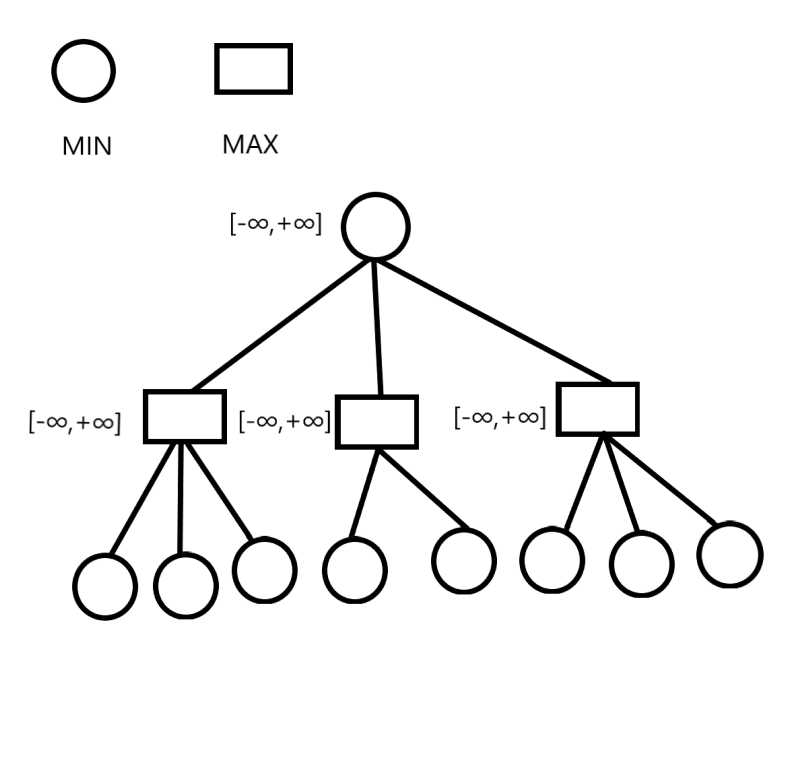
1. 评价函数

评价函数用于对当前局面进行评分，而该评分将作为alpha-beta剪枝算法中的裁剪依据对某些决策的深入搜索进行舍弃。评价标准通常根据不同游戏的规则以及人们的日常经验进行设计。

## 算法内容

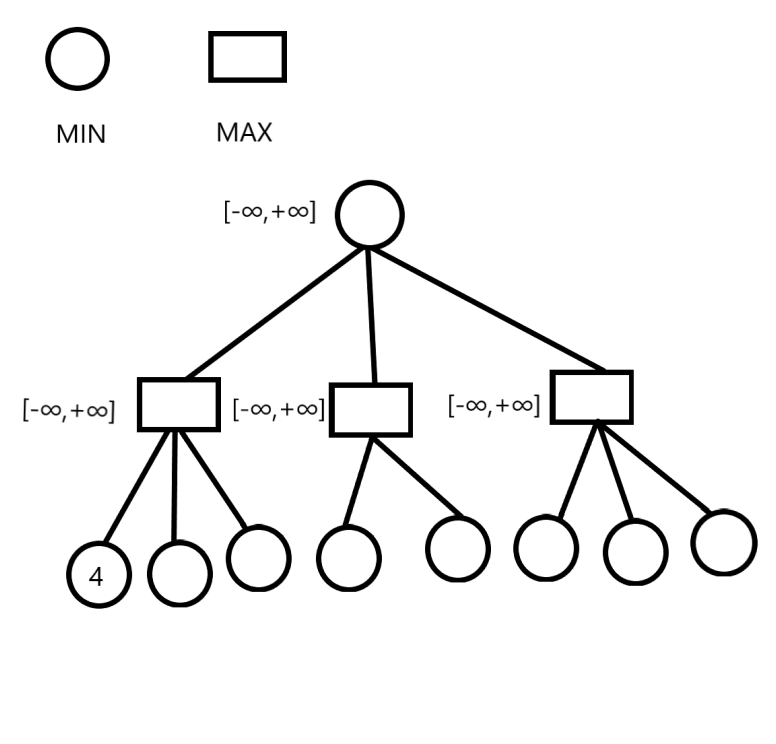
在alpha-beta剪枝算法中，对于极大极小树中的每个节点（每种决策）都会设置一个当前局面的评分区间，并假设alpha为下界，beta为上界。

每个节点（除叶子节点以外）在展开搜索其子节点前，其评分区间都会置为[-∞，+∞]，即alpha为-∞，beta为+∞。而每搜索到一个满足其评分区间更新条件的子结点，区间的alpha值与beta值都会被更新。更新值为评价函数对该子节点的评分。更新规则的设置依据极大层和极小层的区分也有所不同。

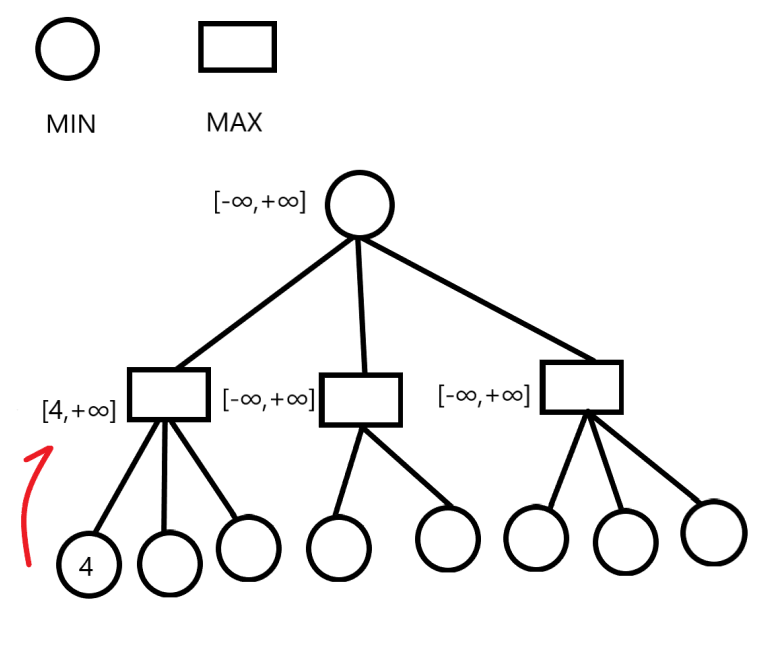


首先，从根节点往下搜索，直至到达第一个叶子节点（基于dfs深度优先搜索算法）；

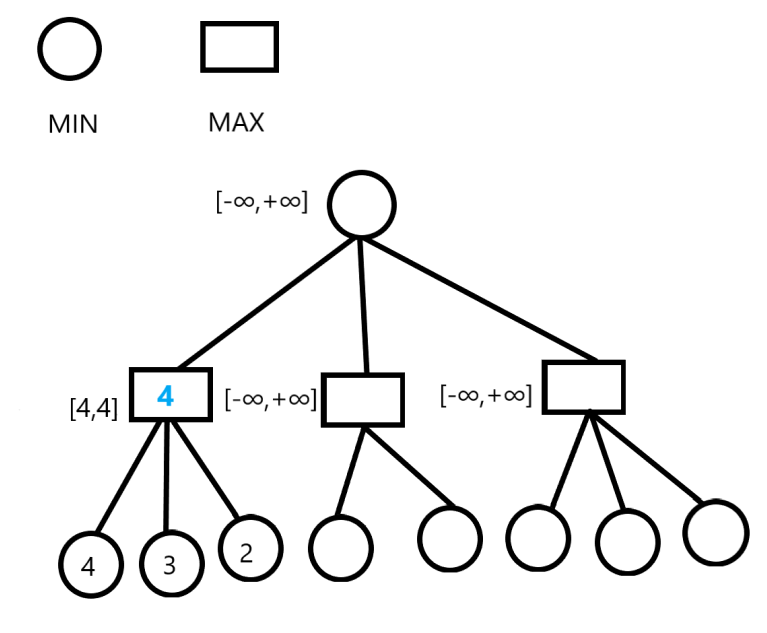
调用估值函数，获得该叶子节点的评分；



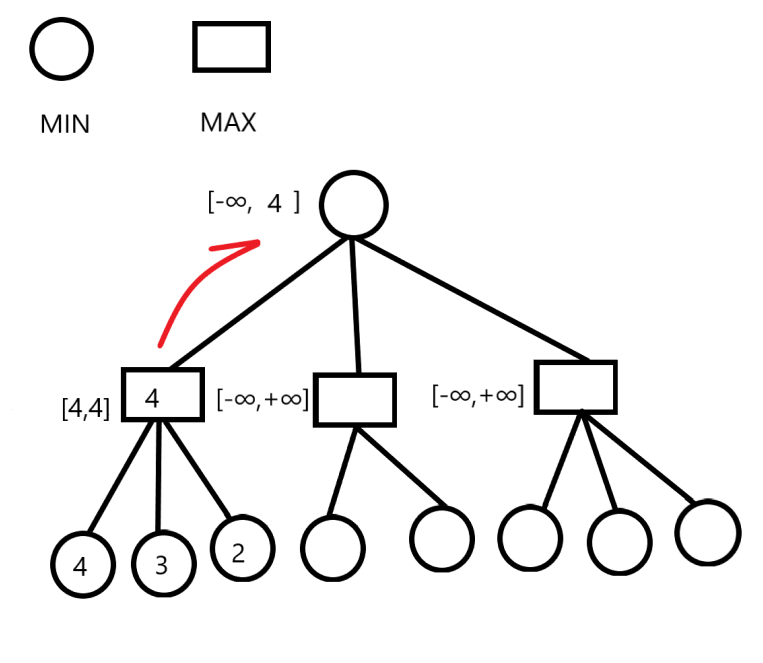
如果叶子节点的父节点位于极大层，更新父节点评分区间的下界为当前叶子节点的评分，而后继续搜索该叶子结点的其他兄弟节点，如果兄弟叶子节点评分大于父节点评分区间的下界，则更新；



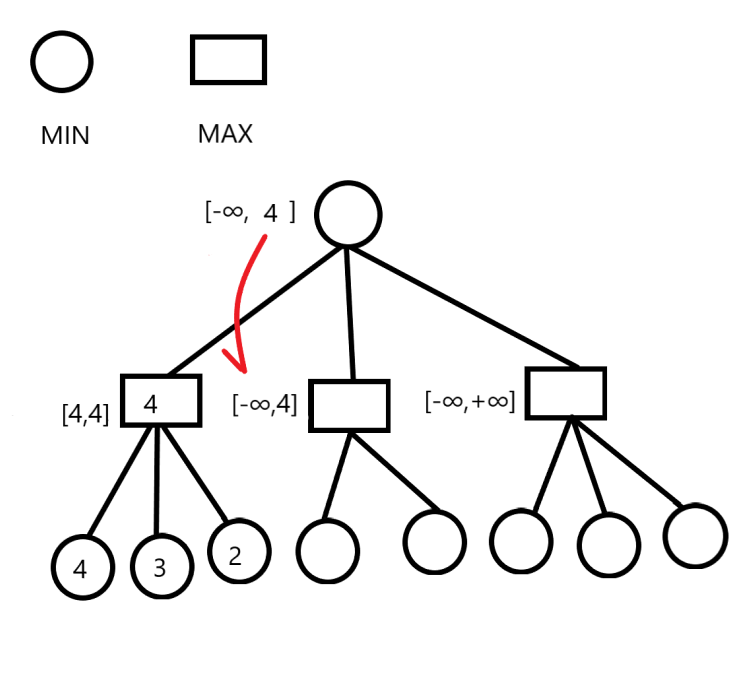
当父节点所有的子节点都搜索完毕，可将父节点搜索区间的上界更新为子节点评分的最大值（等于下界值），此时可确定父节点的评分也为该值（因为上界值beta等于下界值alpha）；



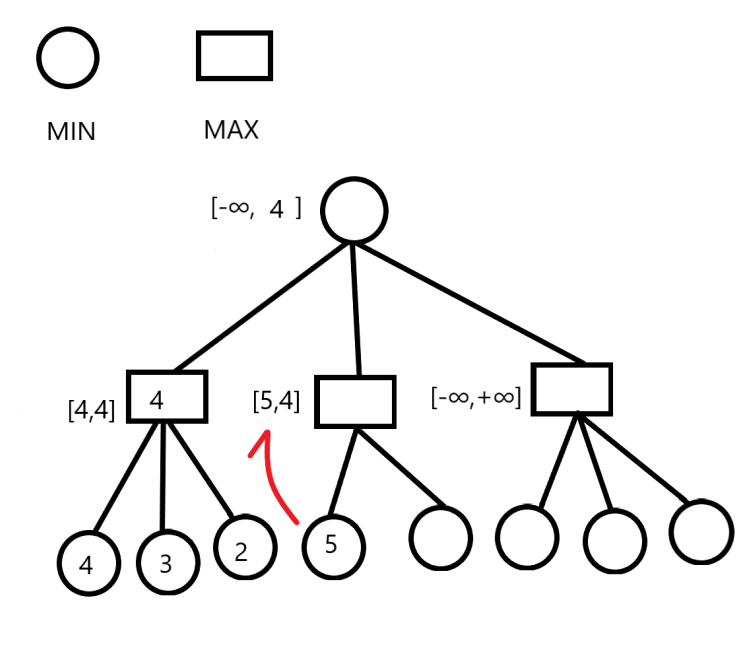
将当前父节点的评分向上传递。因为已经假设当前父节点位于极大层，根据极大极小树中极大层和极小层交替出现的特点，当前父节点的父节点（暂且称为grandpa节点）必定位于极小层。此时，可以将grandpa节点的评分区间中的上界更新为当前父节点的评分。



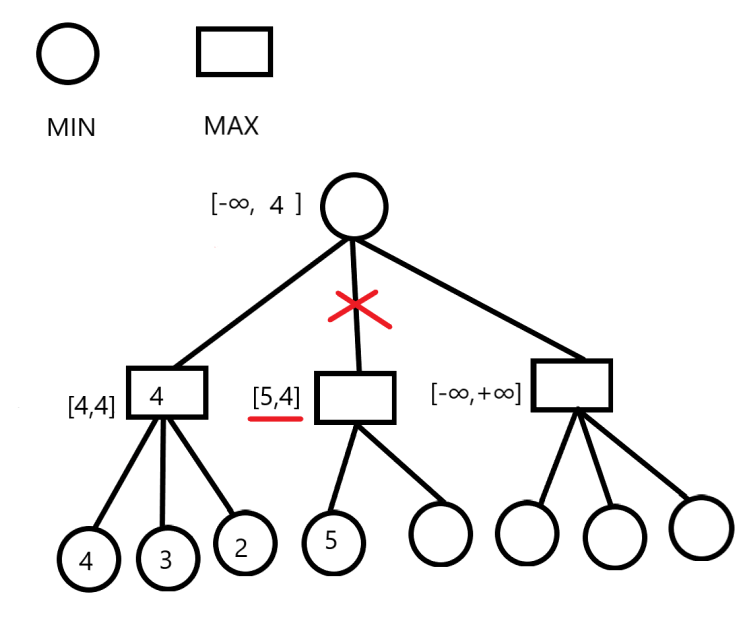
将grandpa节点的上界传递至当前父节点的兄弟节点（暂且称为uncle节点），uncle节点与当前父节点同时位于极大层，将传递的上界值作为uncle节点评分区间的上界。



随后，对uncle节点的子节点进行搜索，并更新uncle节点的下界；



每次更新uncle节点的下界值（alpha），都进行alpha值与beta值的比较：如果出现alpha>beta的情况，则舍弃对uncle节点的展开搜索，从而达到“剪枝”的目的，提高运算与选择最优决策。



# 我的设计

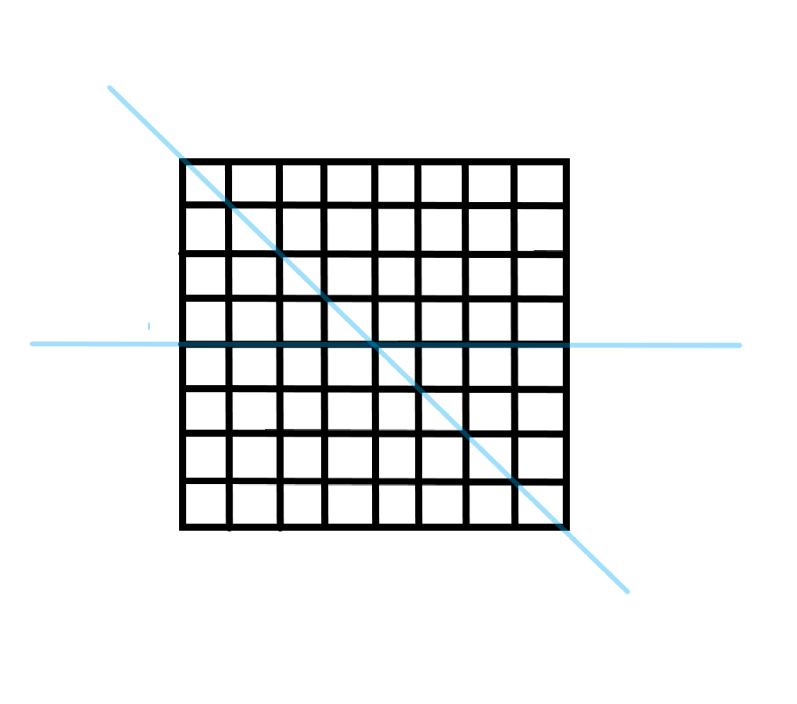
我决定在alpha-beta剪枝算法的基础上，设计一个针对于五子棋的AI对战算法。算法目的主要在于提升决策效率，可以在此过程增加一定的空间支出。

以下是我的思考点：

## （1）五子棋对战棋盘的特点

棋盘是一个n\*n的二维平面，n条横线与n条竖线垂直相交，各横线间距离相等，竖线亦然；而玩家落子时落在线与线之间的交点处。在这样规整的五子棋棋盘上有一个很明显的特点：对称性。可以利用对称性这一特点对alpha-beta裁剪算法做出一定的改进。

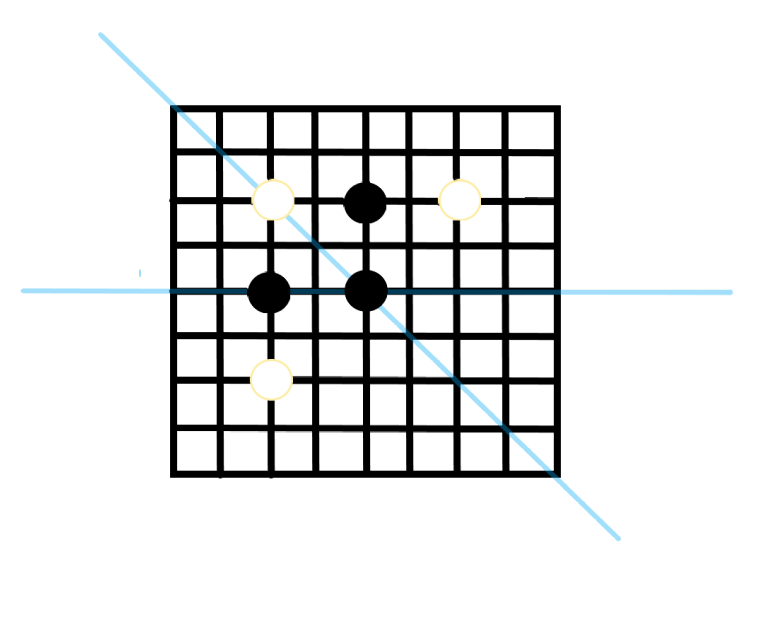
如下是我所绘的一副9\*9简单棋盘，两条蓝色的线为棋盘的其中两条对称轴。



## （2）alpha-beta剪枝算法的不足

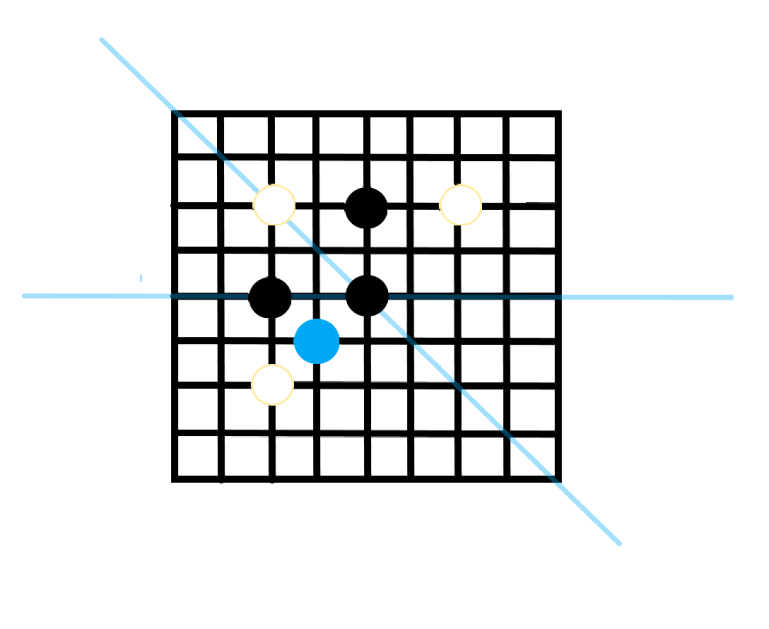
alpha-beta剪枝算法主要聚焦于对评分区间边界有矛盾的节点进行裁剪，减少了展开搜索的操作；但对于那些评分区间没有矛盾、位置与搜索过的兄弟节点（其子节点都已搜索过、确定了没有矛盾的评分区间上界与下界）有一定的相似性、但还没有被搜索的节点，没有裁剪的操作——这是alpha-beta剪枝算法可以继续改进的空间。

而对于兄弟节点间的“相似性”，是根据五子棋棋盘的“对称性”特点进行定义的。即，如果对于对局双方当前的落子情况，存在一条对称轴，使得棋盘上的各子都能够绕该轴对称（黑子对黑子、白子对白子），那么棋盘状态便呈现的是对称状态；此时在对称轴一侧进行落子，和在对称轴另一侧进行落子的结果评分将是一样的。如下图所示，棋盘上的落子在此时呈现出的是对称状态，对称轴为棋盘的一条对角线。

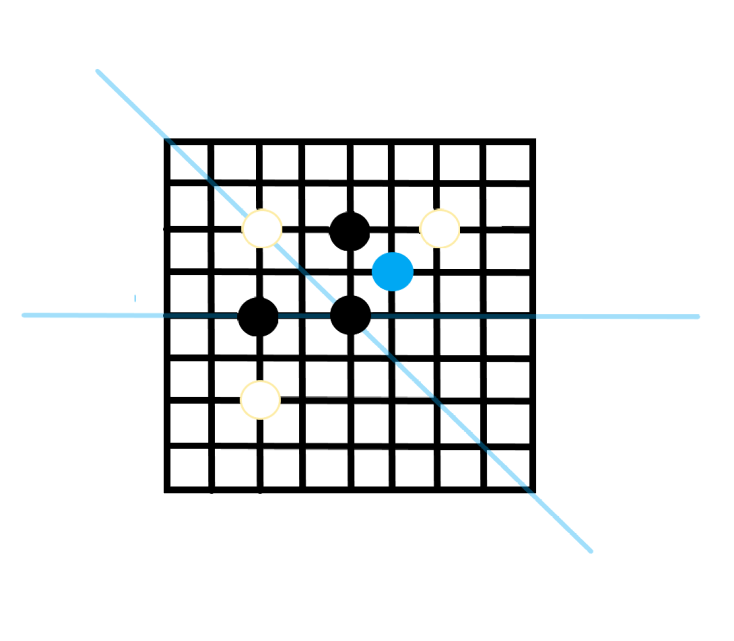


由此而言，如果在搜索过程中，棋盘呈现的是对称状态，那么对节点的搜索还可以“裁剪”掉一半。在对称轴一侧的某个位置进行搜索，和在对称轴另一侧的对应位置进行搜索，所得结果是一样的。

假设蓝色的棋子是我将要探索的落子位置，如下图。



那么这点的评分与在对称轴另一侧对应点的评分是一致的，此时便可以对另一侧的探索进行“剪枝”。



那么关键便在于如何判断当前棋盘是否处于对称状态，以及找到这条对称轴。

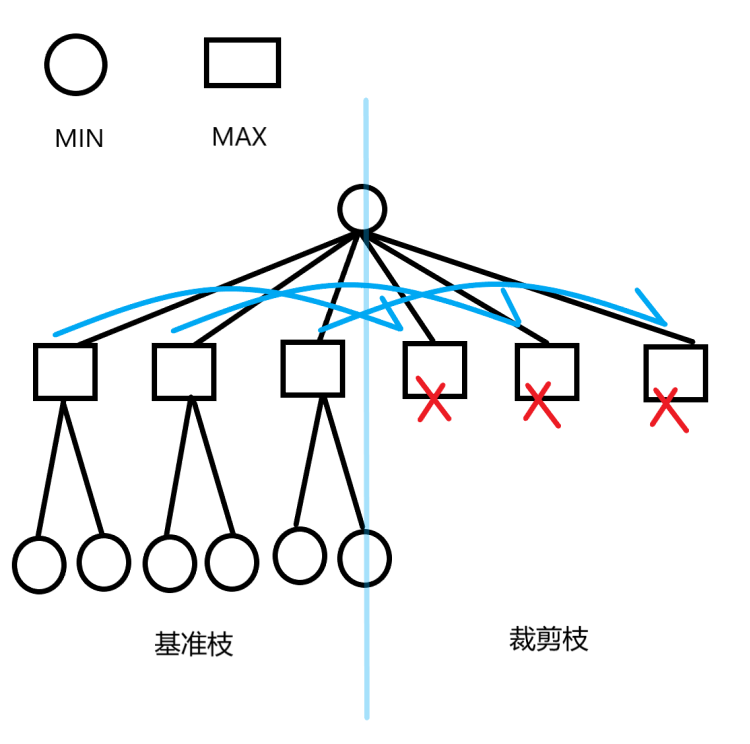
## （3）棋盘状态判断与对称轴寻找

对于n\*n尺寸棋盘上的落子信息，可以利用一个二维数组arr[n][n]保存。由于棋盘上[a,b]处的位置对应二维数组中的arr[a][b]，所以这个二维数组其实是一张哈希表。哈希表可以帮助我们快速查询到一个位置的落子状态，比如无落子、落黑子或是落白子。可以为这些状态设置不同的对应参数，并保存在二维数组中。

而对于对称轴的寻找，棋盘上的落子可能存在多种多样的对称情况，如果将每种对称情况都一一罗列出来，会带来不小的工作量。所以为了便捷快速起见，可以提前规定好棋盘上的四条对称轴：水平居中对称轴、垂直居中对称轴以及两条斜对称轴。当棋盘上的落子以这四条对称轴中的其中任意一条对称时，棋盘状态才可算作处于对称状态。

因为已经提前规定好了对称轴，所以无需进行计算对称轴的操作，只需要让二维数组中的信息与各条对称轴一一对比、检查是否满足对称要求即可。

此外，还可以先提前定义某条对称轴某一侧的位置作为“基准枝”，另一侧则作为“裁剪枝”。如果棋盘处于对称状态、且“基准枝”已经经过了搜索，那么与“基准枝”相对应的“裁剪枝”便无需搜索，达到了剪枝的目的。



# 参考资料

博弈树（game tree）- MBA智库·百科

[博弈树 - MBA智库百科 (mbalib.com)](https://wiki.mbalib.com/wiki/%E5%8D%9A%E5%BC%88%E6%A0%91)

博弈游戏的AI设计（一）：极大极小树 - 折射的文章 - 知乎

<https://zhuanlan.zhihu.com/p/55749861>

博弈游戏的AI设计（二）：博弈树与α-β剪枝 - 折射的文章 - 知乎

<https://zhuanlan.zhihu.com/p/55750669>

Alpha-Beta剪枝(Alpha Beta Pruning) - 急流 - CSDN

<http://t.csdn.cn/11JnF>

博弈树搜索算法 – Notzuonotdied - CSDN

<http://t.csdn.cn/qqI0s>