人机对弈的五子棋 AI

摘要

在这篇文章中，我们将描述人机对弈五子棋程序的设计和算法的相关说明.

关键词

五子棋 minimax alpha-beta 剪枝

1. 介绍

五子棋 AI 对弈是在人机对弈过程中，计算机根据棋盘上所有棋子的点位来通过算法搜索到一个最优的落子点，我们使用了判断多种局势的评估函数来对某种走法进行数值评估，并使用搜索算法 minimax，利用每一步的估值来找到一个最佳点。在这个程序中我们给 minimax 算法添加了一个简单的 alpha-beta 剪枝操作，在一定程度上提升了搜索效率，支持更深层次的搜索空间。

2. 函数，变量和模块设计

2.1 数据结构及逻辑

数据结构设计:

- 全局变量 chessBoard[][]用来存储棋盘落子
- Point 结构存储坐标、落子类型、估值。
- vector<point> 用来存储所有合法走法。

游戏逻辑设计:

选择好游戏设置并开始游戏后通过传入当前玩家落子 point，随后电脑执行走法时，调用 createmoves 生成所有合法走法 vector<point>返回值，并对所有 point 进行估值。随后通过 minimax 函数进行搜索和剪枝操作，选出估值最大的走法进行实现。在每次落子后都会进行游戏是否有人（或电脑）胜出的判断。

2.2 createmoves

原则上，五子棋棋盘上所有的空点都是一个合法的走法，但在很多情况下，有大量的走法是无意义的，比如某一个走法离其他棋子的距离都过远，那么这个走法就起不到任何作用。因此，我们在筛选出棋盘上的空位的同时，将所有三步以内有棋子的位置都视为合法走位。suit_position 函数的作用是判断此点位的 8 个方向的 3 步以内是否有棋子。

2.3 gameover

在电脑和玩家每下一步棋之后，都以此步棋子为中心，寻找其 8 个方向的前后四步之内，如果相反方向相同颜色的连续棋子数加起来大于等于 5，就判游戏结束。

2.4 searchmove

这个模块实现了搜索的核心部分。主要包括决策函数 mkdecision 和 minimax 算法和 alpha-beta 剪枝优化函数。

在 mkdecision 函数中，调用 minimax 搜索函数得到决策结果，并执行走法，并将决策得出的走法返回。

2.4.1 minimax 算法

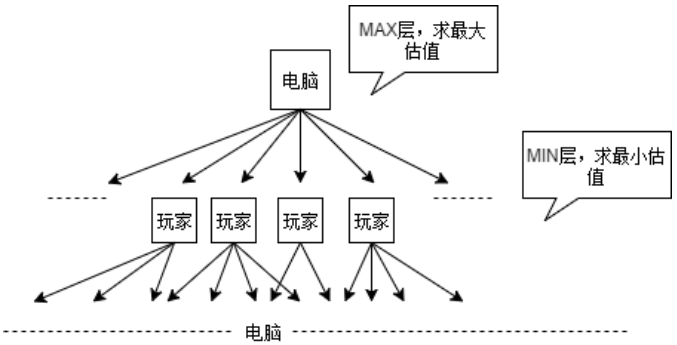
Minimax 算法的思想比较简单。在电脑选择决策的时候，我们的搜索过程其实是遍历一棵很大的树。假设电脑先手，那么第一层就是电脑所有的走法，第二层就是玩家所有的走法，以此类推。假设我们平均每一步都有 n 步走法，那么从根节点开始，每一层节点的数量都是上一层的 N 倍。在遍历的过程中，我们的决策方案是:

电脑走棋的层为 MAX 层，在这一层电脑要保证自己的走法最具优势，就要选择估分最高的节点。

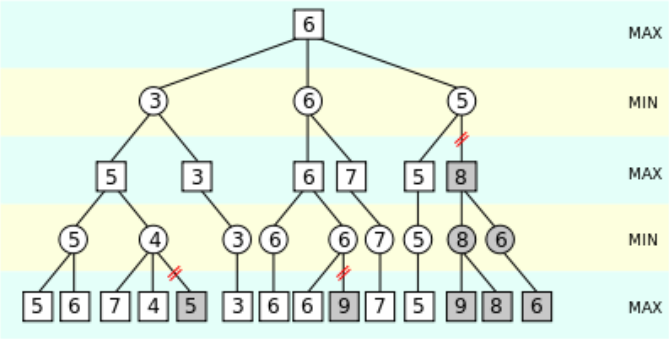
玩家走棋的这一层为 MIN 层，要保证玩家的利益最大化，就要选择估分最低的节点。

估算 MAX 层时调用的函数为 search_max，估算 MIN 层时调用函数为 search_min。

最后，在 minimax 主函数中，调用 createmove 函数计算出所有的合法走法，然后在这些走法中选取一个估值最大的走法，作为最终要执行的走法。



2.4.2 alpha-beta 剪枝优化



简单来说，alpha-beta 剪枝的大体思想就是，如果电脑向下一层发现了一个走法对玩家更有利，就不会在这一层上继续向下搜索，反之，如果玩家发现电脑的走法更不利，也不会继续向下搜索而是直接放弃这一步，并直接放弃搜索后面的所有节点。这样的算法有时并不能得出最优的结果，但是可以得到一个可以接受的结果。

大体思路如图，在 MAX 层，假设当前层已经搜索到一个最大值 best，如果发现下一个节点的下一层会产生一个比 best 还小的值，那么就直接剪掉此节点。在 MIN 层，假设当前已经搜索到了一个最小值 min, 如果发现下一个节点的下一层会产生一个比 min 还大的值，就直接剪掉此节点。

2.5 evaluate 估值算法

调用 getline 函数获取对应目标棋子周围棋子的状况。

对于几种不同的棋型，分别给予不同的估值，同时，对于同种棋型的估值选取，采用防止对方得分>自己得分的策略，例如同为 0111*的棋型，当*方为 1 时估值为 300000，为 2 时估值为 299500。（0 为空白，1 为黑子，2 为白子）。对于某一位置的估值为两种策略得分之和。

同时对于不属于棋型，但是很有意义的棋子布局也进行了相应估值，例如某一子周围存在较多数的相连二子的情况给予了 3000 的估值。使得下棋看起来更加自然。

3. 实验

对比 minimax 算法和 minimax+alpha-beta 剪枝算法的运行时效。

实验条件:

Cpu: i5-8300H, 2.3Ghz

IDE：VS

Mode: Debug X64

每一层测试次数均测多次取平均值

算法/搜索层数(层)/时间(ms)	2	3	4
Minimax	210	3972	136721
Minimax+alpha-beta	210	7157	237147

根据实验结果，搜索 2 层的时候，剪枝优化的效果不大，主要是因为两层的时候需要剪枝的情况比较少，其次是剪枝后的影响不大，因为 2 层往下就没有其他更多的节点了。

搜索 3 层时，剪枝优化效果达到了 44%，搜索 4 层时，优化效果达到了 42%。两者非常相近且趋于 50%。这是因为 alpha-beta 剪枝时在约平均达到一半节点的时候，发现了可剪枝的节点并剪枝，因此趋于 50%。Alpha-beta 效果十分明显。

4. 额外功能说明

4.1 悔棋

我们在人机对战中才用保留变量的办法，可以使玩家选择连续悔棋一步的操作。

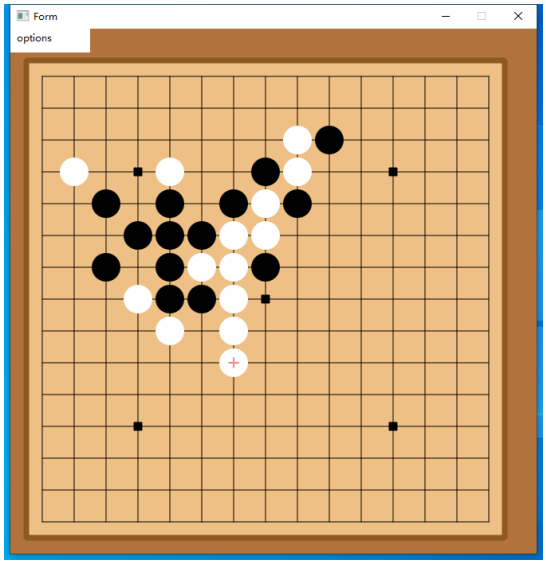
4.2 复盘

我们使用了 vector 向量数组来保存了玩家和电脑的所有的走法执行结果和顺序，并保存了先手者。在一局结束后，在我们的 UI 界面上，点击 replay，可以每 800ms 一步的频率将此局完整过程复现出来。

5. UI 展示

我们为此游戏专门使用 qt 做了一个 UI 界面，其中人机对战时的菜单栏中有:

- Back: 后撤一步
- Restart：重新开始
- Replay: 游戏结束后进行复盘。



6. 实验体会

本次人工智能算法实验主要考察了我们对于一个五子棋搜索过程的理解和运用，并在此基础上添加了剪枝优化。这个类似学习的过程告诉我们设计一个复杂算法的基本思路和优化算法的方法，要从实际感受出发，先具象化算法的思想，再从这些具象的过程中寻求优化的过程。整体来说在理解算法思想的基础上难度中等。

7. 参考文献

[1] https://www.cnblogs.com/maxuewei2/p/4825520.html

[2] https://blog.csdn.net/lihongxun945/article/details/50625267

[3] https://blog.csdn.net/lihongxun945/article/details/50668253