UESTC 五子棋项目实验报告

——2023080903010 黄雪锋

注：以下内容中，用1表示白子，0表示空位，2表示黑子。

项目需求

通过学习相关算法，自主设计一个**五子棋AI**程序。程序需要实现人机互弈，且AI具有**较高的棋力**。

项目设计

项目结构

- Data --存放hash数据
- Header --头文件
- Resource --源文件

项目模块

- main
- evaluate --棋局评估
- generator --位置生成
- scrap --其余辅助函数
- search --搜索树
- ui --界面相关功能
- variable --全局变量
- hash --置换表和棋型hash

main为主函数，是调动其他模块的主控制台。

ui模块构建了游戏窗口，通过读取variable中的全局变量来绘制棋盘中的棋子，同时帮助玩家在棋盘落子。

search模块是项目的核心。该模块通过MaxMin算法，结合alpha-beta剪枝，启发式搜索，搜索节点个数限制等优化方法，实现了八层搜索（release模式 O3优化的前提下，每步平均计算不过7s）。evaluate模块为该模块的搜索树的节点打分提供依据，generator函数为搜索树提供有序的节点。搜索结果将会反映在全局变量中，供ui模块读取，实现可视化。

hash模块是专注于优化evaluate的模块。hash模块里综合了zobrist和经典棋型的hash表。zobrist是全局棋盘的hash，而经典棋型的hash则专注于局部。其中，经典棋型的hash表不仅应用在

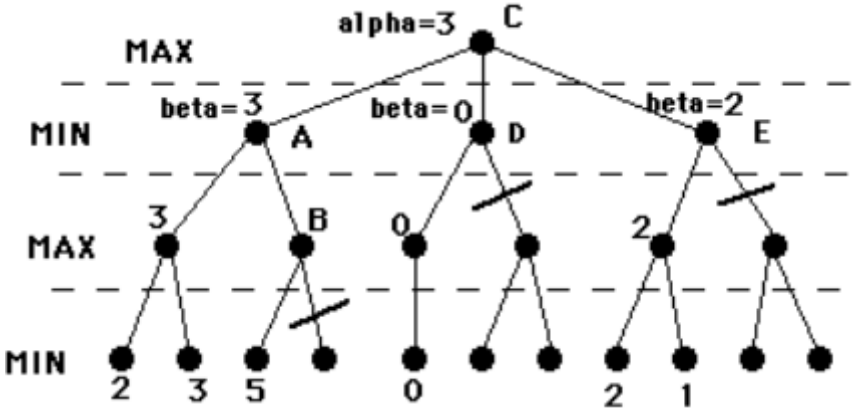
evaluate函数中，也为generator模块的节点排序优化做了很大贡献。

scrap模块里，综合了一些在search模块中会被反复调用的辅助函数，如落子，拿子，判断输赢。

项目优化

alpha-beta剪枝

在假定对弈双方水平极高的情况下，对方总是会在搜索树的节点中选取对他最有利的节点。基于这样的前提，我们可以在搜索过程中，在MAX层保留已搜索节点中的分数的最大值alpha，MIN层保留最小值beta。若MAX层的子节点一搜出比保留的值还要小的值，则直接放弃搜索该子节点。MIN层同理。



例如在上图中，在搜索完A节点之后，C保留了alpha值为3。继续搜索D节点时，D节点的子节点搜索出了值为0的节点，那么D的beta值就为0。那么基于假设，在D所在的MIN层，D不可能选择出比0的值还要大的节点，而我们已经有一个值为3的选择了，那么D就没有必要继续搜索了，故剪去D之后所有的子节点。

启发式搜索

如果在搜索子节点的过程中，搜到的第一个子节点就是预期中最好的节点，那么后面的节点很有可能都会被alpha-beta剪枝剪掉。启发式搜索的功能就是对待搜索节点进行预评估，并将所有待搜索节点进行排序，这样可以极大地提高alpha-beta剪枝的优化效能。

搜索节点个数限制

上述generator函数中，最初步的生成位置是棋盘上距任意棋子距离不超过两格的位置。这样的函数会使得搜索树的子节点非常多，降低search模块的效率。

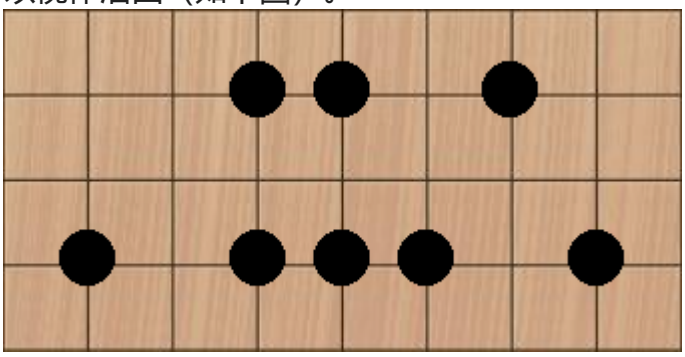
搜索限制指的是在进行节点排序的前提下，只对待搜索节点队列中的前几个节点进行搜索，后续节点全部舍弃。这样做的合理性在于，因为节点排序已经将可能分数最高的节点排在了前面，后面节点选择的可能性就会很低，不如抛弃掉这些可能性低的节点以提升效率（在众多节点分数接近的情况下优化效果明显，特别是对局前期）。

但是，这样的限制会对评估函数和节点排序的准确性提出更高的要求，否则棋力将会显著下降。其实由此也可见全节点搜索可以掩盖掉或弥补评估函数和节点排序一些准确性的不足。

hash

hash模块里综合了两个功能，一是zobrist，一是棋型的hash。前者是用来优化evaluate模块的；后者一方面提升了evaluate和generator的效率，另一方面提高了二者评估的准确性。

引入棋型hash的原因在于，一个能完美评估所有棋型且复杂度低的函数难以构建。之所以难构建，是因为 010110 1011101 等棋型的存在。这类棋型的特征是在一串棋子中有空位，却具有和经典棋型一样的意义（经典棋型指的是无空格的棋型）。比如010110可以视作活三，1011101可以视作活四（如下图）。



对经典棋型的评分是将n个子连在一起的评分视作10^n,若有一端阻隔，幂数减一，若两方阻隔，评分归零（已连五子另做讨论）。那么，1000就意味着一方已经连3子了，连3子的另一层抽象意义就是，必须进行阻隔，否则失败。同理，10000意味着连4子，抽象意义即为已经无法阻隔。

那么，我们不妨由抽象意义来进行评分，1000意味着必须阻隔，10000意味着无法阻隔成功，100意味着再下一个子就会形成必须阻隔的棋型，10同理。

由这样的抽象意义，其实评估的方法也显而易见了，那就是“更进一步”，对棋型的评估取决于，在当前棋型下，多下一个子所能取得的最大评分。例如若某棋型再下一个子就可以活四，那就视为活三。而这样的评估方法相对于hash更难实现，时间复杂度上也不见得更加有优势。故“更进一步”的方法被用作hash数据的生成，而如何生成有意义的棋型，在下面的项目实现中会详细展开。

项目实现

evaluate

在落子过程中，维护一个数组用于记录各行各列和各斜行上棋子的个数，每次进行全局评估时，只对有棋子的行、列斜行进行评估，棋局总分即为所有行列斜行分数的总和。各行各列之间的评估是相对独立的，故可用统一的一个函数对一串棋子进行分数评估。在对一串棋子进行评估时，可以对这串棋子进行棋型匹配，获取对应hash值，从而得到棋局评分。

hash

hash中有两个模块，一是zobrist，二是棋型hash

zobrist模块给棋盘上的每个格点的三个状态都生成了一个ull类型的随机数，棋盘状态对应的key就是从（1,1）对应状态的随机数一直异或运算至（15，15）的数值。每次全局评估时，都先计算key，再判断key是否有对应的value，如果有，直接返回value，如果没有，就先进行全局评估，再把key和棋盘分数进行映射。

经测试，由一串字符串作为key来表示棋型的hash速度特别慢，而用int类型作为key的hash速度相比之下快多了。故在此借鉴zobrist的思想，给一串棋子的每个位置的三种状态生成一个随机数，同理由异或结果作为key，可是实现非常高效的hash。

而hash数据如何生成呢？重点在于找到有意义的棋型。对大多数棋型进行规律总结，会发现棋型的头尾有一个规律，它们往往只有三种方式（对于白子1来说）：00 20 2。00意即表示两格以内没有阻隔，20表示第一格之外开始有阻隔，2表示直接被阻隔。我们将上述三种方式称为前后缀。将棋型前后缀去除，就会发现中间部分，只存在待评估棋子和空格0。那么中间部分就可以由1（或2）和0的排列组合进行生成（生成出来的部分不允许有连续两个0，否则其将视为前后缀部分），再和前后缀组合，就可以获得有意义的棋型。或许这样的生成方式不能囊括所有的有意义的棋型，但通过对由此方式生成的500多组的hash值的测试，发现ai的棋力并没有下降。

generator

最初生成的位置，取棋盘上所有棋子附近两格内的空格子。取两格以内的原因在于,1101这样的棋型是具有意义的，而只取一格以内的棋子的话，第三个1就有可能不会被列入生成位置的队列中，造成防守的纰漏。

随后对每一个生成位置进行局部评估。对每个位置的四个方向进行有规律的延伸，将延伸得到的棋子串进行棋型匹配，从而实现局部评估。根据局部评估得出的分数，把生成位置放入不同队列中，每个队列只存放相同类型的位置（比如能实现五子的位置，能实现双三的位置……）。最后入队进入最终的生成位置队列时，就优先入队对于己方意义更大的位置（先五子，后四子，再双三……），实现位置的排序。

search

通过深度优先搜索实现，某节点的分数由其子节点决定。每次向下延伸时，都要传递alpha和beta的值以剪枝。

scrap

scrap中有三个函数。Win函数，PlacePoint函数，TakePoint函数。

Win函数，通过对双方分值的差的判断来决定游戏是否结束。

PlacePoint函数，搜索过程中或搜索结束时，需要放下棋子，而放下棋子时需要维护相应的全局变量，PlacePoint函数的作用在于此。

TakePoint函数，同PlacePoint函数，对拿起一个子影响到的全局变量进行维护。

项目测试

在alpha-beta剪枝，启发式搜索，zobrist的优化效果下，能完全应付四层搜索。在release模式下，能完全应付六层搜索。

在以上的基础上，加上棋型hash，搜索节点个数限制的情况下，能勉强应付八层搜索，release模式下，能完全应付八层搜索。

在不加棋型hash，所有评估只针对经典棋型评估的情况下进行搜索节点个数限制，ai棋力约等于0。

在上述优化全部实现时，在O3优化和release模式下能完全实现八层搜索。