|  |
| --- |
| **计算机专业类课程** |
| **实验报告** |
| **课程名称： 数据结构与算法I**  **学院专业：计算机科学与工程学院**  **学生姓名：幸子豪**  **学　　号：2024080907009**  **指导教师：俸志刚** |
| **日　　期：2024年 12月 16日** |
|  |
| 电子科技大学计算机学院实验中心 |

**电 子 科 技 大 学**

**实 验 报 告**

**实验一**

# 一、实验室名称：

电子科技大学清水河校区品学楼C222

# 二、实验项目名称：

基于Minimax博弈树和α-β剪枝的五子棋AI（Gobang AI）

# 三、实验目的：

通过设计算法，实现能与人进行五子棋对弈的AI

# 四、实验主要内容：

通过Qt设计UI界面；通过Minimax算法在博弈树上搜索，并通过α-β剪枝等优化提升搜索效率，实现AI内部算法

实现五子棋人机对弈

# 五、实验器材（设备、元器件）：

1. 硬件环境：
   1. 处理器：AMD Ryzen 7 8845HS w/ Radeon 780M Graphics 八核
   2. 内存：24 GB ( 英睿达 DDR5 5600MHz 12GB x 2 )
2. 操作系统：Windows 11 家庭中文版 64位
3. 编程语言：C++ 17
4. 开发工具：Visual Studio Code，Qt Creator
5. 编译器：MinGW-w64

# 六、实验步骤：

1. 问题描述：设计AI实现五子棋人机对弈

2. 算法分析与概要设计

* 1. 极小化极大算法（Minimax 算法）：基于贪心的底层搜索思路
  2. α-β剪枝：利用已知信息剪枝，提高搜索效率
  3. 棋局局面估价函数：AI鉴定局面的方法，直接影响AI的棋力
  4. 启发式搜索：基于α-β剪枝的底层逻辑，通过调整搜索顺序提升剪枝的效果
  5. Zobrist-Hashing：通过局面的哈希值实现记忆化，优化估价函数的效率，且哈希冲突概率极低
  6. AC自动机：提高多模式串匹配的效率，进而影响估价函数的效率
  7. Qt-UI框架：绘制棋盘棋子等，获取鼠标事件，提升交互体验

3. 核心算法的详细设计与实现

a) 博弈树与Minimax算法

对于博弈游戏，我们可以将双方选手交替操作后的局面顺次链接，具象为链状结构，那么链上相邻两个节点所代表的局面一定是由不同选手操作后的局面。

而未来多种可能的操作情况可以具象为多条链的合并，即一颗树，我们称其为博弈树。同理，树的相邻两层的节点所代表的局面亦为由不同选手操作后的局面。

在博弈时，一方选手必然会选择对自己最有益（也即对对手最不利）的下法。如果我们将局面表示为一个具体的值（称为估值），估值越大表示对 AI 越有利，越小表示对玩家越有利，那么我们就可以将博弈树的每层划分为Min层和Max层。

Max层代表AI操作，此时AI会选择操作后估值尽可能大的局面；

Min层代表玩家操作，此时AI会认为玩家聪明，即选择操作后估值尽可能小的局面；

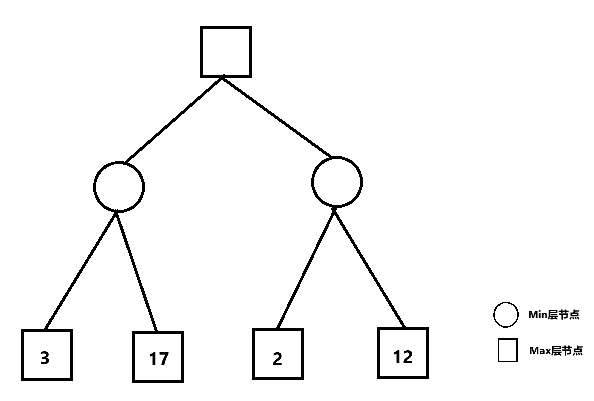
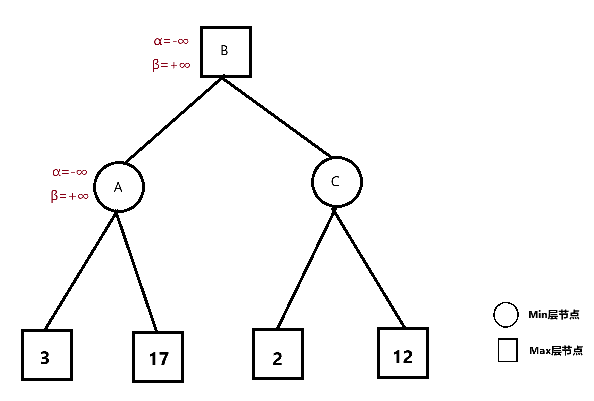
这就是Minimax算法，是AI进行搜索的基本思路

b) α-β剪枝

在博弈树上搜索时，当我们进入一棵子树时，我们可以根据已经搜索过的子树，得知当前估值的范围，即 ，那么对于估值不在范围内的子树，我们可以不必搜索，从而达到剪枝的目的。

下面，我们通过一个具体的例子来了解α-β剪枝的具体流程。

例如我们有博弈树如下，节点内的数字为对应局面的估值：

初始时，递归时下传，回溯时更新。

图示

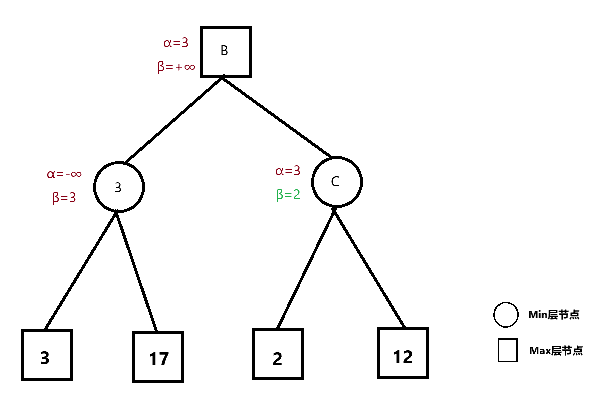
描述已自动生成在搜索完估值为“3”的节点后，回溯至其父节点A。注意到节点A是Min层的一个节点，于是A节点最后选择的局面的估值不会大于3，因而可以将其β值更新为3。

同理，搜索完A的整棵子树，得到A返回的估值是3，由于A的父节点B是Max层的一个节点，于是B节点最后选择的局面的估值不会小于3，因而可将其α更新为3，并下传给C节点 。

图示

描述已自动生成

在搜索完估值为“2”的节点后，同理将C节点的β更新为2。



此时注意到C节点的α>β，说明B无论如何也不会选择节点C，所以可以直接停止搜索，那么估值为“12”的节点直接被“剪掉”了，于是优化了搜索的效率。

具体实现可以参考如下代码：

1. step dfs(vector<chess> &chesses, bool color, int alpha, int beta, int dep, int maxdep) // 分别表示：所有落子的集合，当前层的颜色，α值，β值，当前搜索深度，最大搜索深度

2. {

3. if (dep == maxdep) // 达到最大搜索深度，返回当前的估值与对应的决策

4. return step{chesses[chesses.size() - maxdep], Board\_Evaluate(chesses)};

5. vector<chess> vec = Get\_Next\_Steps(chesses, color); // 获取下一步可能的走法

6. step mx, mn;

7. mx.score = -inf, mn.score = inf;

8. for (auto x : vec) {

9. chesses.emplace\_back(x);

10. step res = dfs(chesses, !color, alpha, beta, dep + 1, maxdep);

11. chesses.pop\_back();

12. if (dep & 1) { // Min 层，更新 beta 值

13. mn = min(mn, res);

14. beta = min(beta, mn.score);

15. }

16. else { // Max 层，更新 alpha 值

17. mx = max(mx, res);

18. alpha = max(alpha, mx.score);

19. }

20. if (alpha >= beta) break; // alpha-beta 剪枝

21. }

22. return dep & 1 ? mn : mx; // 根据当前是 Min 层还是 Max 层返回估值

23. }

c) 启发式搜索

注意到，α>β的情况出现得越早，剪枝的效果越好。因此我们可以在搜索下一步前，对可能的下一步按照估值排序。

在Max层，先搜索估值大的下一步，使更新的α尽可能大；在Min层，先搜索估值小的下一步，使更新的β尽可能小。

启发式搜索可以极大地提升α-β剪枝的效果，同时避免极端情况导致效率低下。

d) AC自动机

在估价函数中，我们有一个映射表（例如：五连映射一个很大的值，活四映射的值次之，活三再次之，等等），在估价时，我们需要将映射表中的字符串作为模式串与局面情况作为主串进行匹配。

对于多模式串匹配问题，可以使用AC自动机算法进行优化，提高估价函数的效率。

但受算法本身常数因子的影响，只用当落子数较大时，算法的优化效果才明显。

# 七、实验数据及结果分析：

图表, 气泡图

描述已自动生成 1. 运行效果展示：

运行效果展示

2. AC自动机优化效率测试：

测试方法：

一次测试包含：随机生成N步合法落子（黑白各N/2步），执行下面操作3次取平均值作为本次测试的结果：删去记忆化，调用整体局面估价函数1000次，将估价函数运行时间累加。

取三次测试的结果求平均值作为最终结果（四舍五入至两位小数）。

测试结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | 50 | 100 | 150 | 200 |
| 使用AC自动机 | 546.56ms | 1053.55ms | 1520.67ms | 2015.00ms |
| 未使用AC自动机 | 690.56ms | 1314.33ms | 1812.22ms | 2328.55ms |

AC自动机优化效率测试

从测试结果可以看出，AC自动机在一定程度上提升了估价函数的效率。

3. 棋力测试：

1) 测试1

测试网站：https://game.higrid.net/wuzi/

测试难度：高手水平

蓝色的球

低可信度描述已自动生成图片包含 图表

描述已自动生成 测试详情：本项目AI执黑，网站AI执白，本项目AI胜。

测试1网站截图

测试1本地截图

2) 测试2

测试网站：http://www.ku51.net/wuziqi/

测试难度：default

图表, 气泡图

描述已自动生成图表, 气泡图

描述已自动生成 测试详情：本项目AI执黑，网站AI执白，本项目AI胜。

测试2网站截图

测试2本地截图

# 八、总结及心得体会：

1. 该项目AI搜索效率依然有待提升，在6层搜索中每步仍需搜索1~3s
2. 该项目AI估价函数欠佳，当局面比较复杂时，AI有时会错过决胜的机会
3. 由于初次学习并制作UI，UI界面仍不够完

# 九、对本实验过程及方法、手段的改进建议及展望：

1. 增加贪心算法算杀
2. 添加悔棋功能

**报告评分：**

**指导教师签字：**