|  |
| --- |
| 电子科技大学  **计算机专业类课程** |
| **实验报告** |
| **课程名称：程序设计基础(A班)**  **学院专业：计算机科学与工程学院**  **学生姓名：王正仁**  **学　　号：2019081308021**  **指导教师：俸志刚** |
| **日　　期：2020年 6月 5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：

家里地址：四川省遂宁市射洪县太和镇衙署街3楼1号

# 二、实验项目名称：

五子棋AI

# 三、实验目的：

利用所学C语言技能，实现五子棋AI算法。通过项目实战，提高包含语言水平，算法水平，项目水平等方面的综合程序设计水平。

# 四、实验主要内容：

利用Qt实现五子棋GUI交互界面。借助C/C++语言，和相应AI算法，完成五子棋AI。

# 五、实验器材（设备、元器件）：



# 六、实验步骤：

1. 问题描述

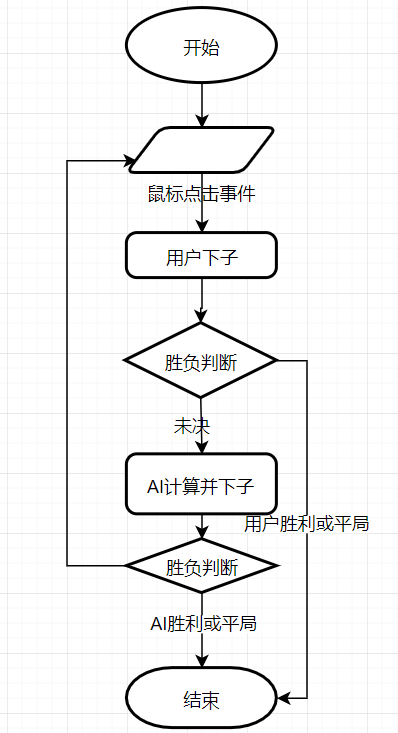
C/C++语言实现五子棋AI程序（最好附带GUI），要求五子棋程序交互性好，界面清晰美观，AI反应速度快，棋力尽可能强。

2. 算法分析与概要设计

输入：用户鼠标点击事件

输出：棋盘内电脑下子

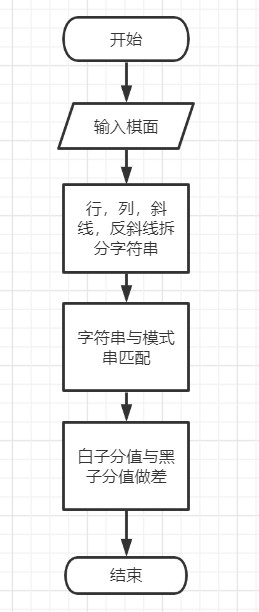
输入转化为输出：



该算法即直接模拟现实下棋规则——首先由用户点击下子，然后判断此时棋局状态，如果用户胜利或者平局即停止算法，否则AI进行计算，得出估计最优位置并下子，此时再次进行棋局状态判断，如果AI胜利或者平局，则停止算法，否则算法回到等待用户下子阶段。

3. 核心算法的详细设计与实现

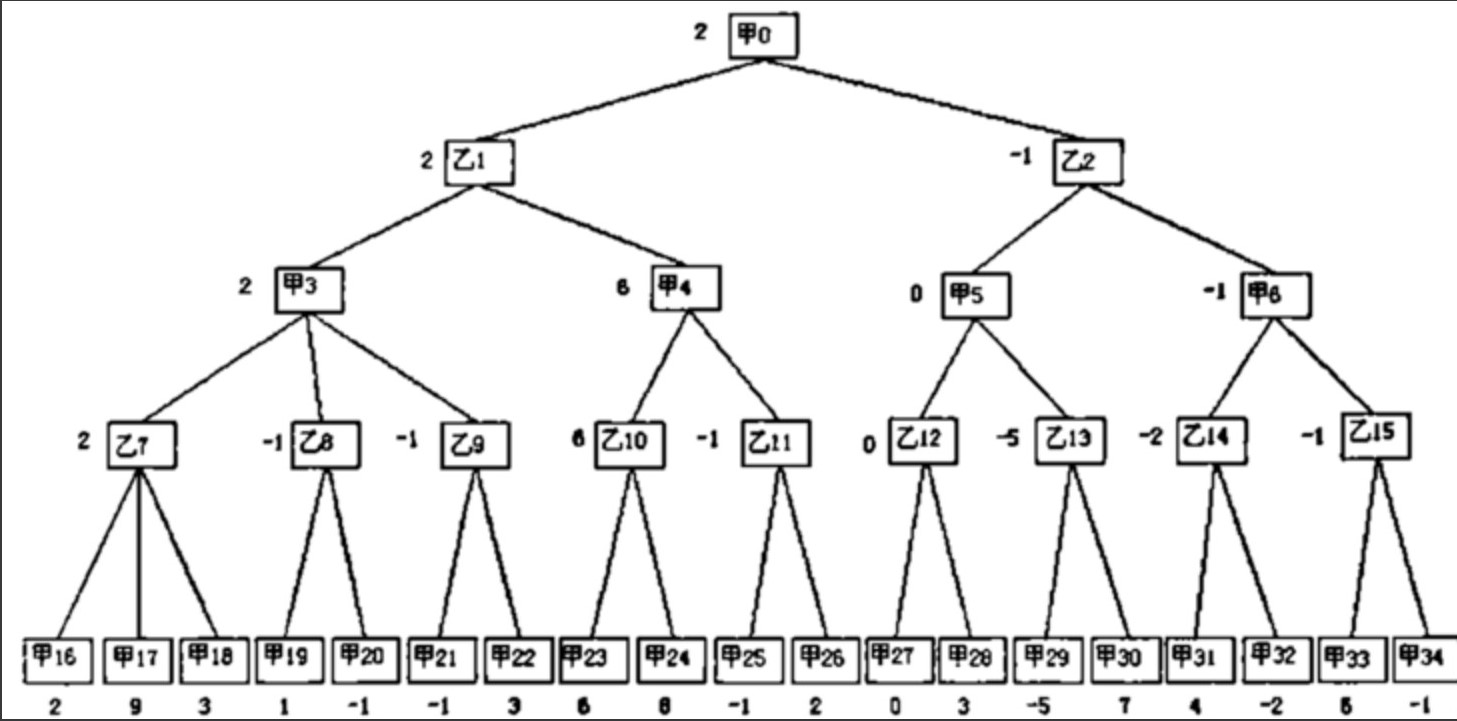
1. 评分算法：AI的基础且核心算法，该算法基于模式匹配。
2. 极大极小搜索算法：提高AI思考深度，改良AI强度的优化。
3. Alpha-Beta剪枝算法：提高AI思考速度，本质是对Brutal Force搜索的一种最优性剪枝。
4. 哈希记忆化优化：本质是空间换时间，用于优化评分算法。
5. 评分算法



评分算法的最终结果是对一个棋面的打分，该算法被人为地分为相似的两部分，即白子棋面分值与黑子棋面分值，最终结果定义为白子棋面分值与黑子棋面分值之差。

算法实现基于模式匹配，模式即指基本棋形，有连5，活4，冲4，活3，眠3等，这些都是模式串。我们不同棋子映射为不同字符，分别把每行，每列，每斜线，每反斜线上的棋子连接起来，就得到了一组被匹配串。一个被匹配串的分值就是符合匹配的模式串的权重之和，而棋面的打分即这组被匹配串的分值之和。

上文提到了权重的概念，实际上，不同的棋形对应的权重是不同的，一般遵循 的规律，没有数学上明确的要求，具有一定的随意性。

1. 极大极小搜索算法

如图所示，不妨令奇数层为白方下子，偶数层为黑方下子，并且奇数层期望棋面估分越高越好，偶数层期望棋面估分越低越好（棋面估分定义为白子估分减去黑子估分）。

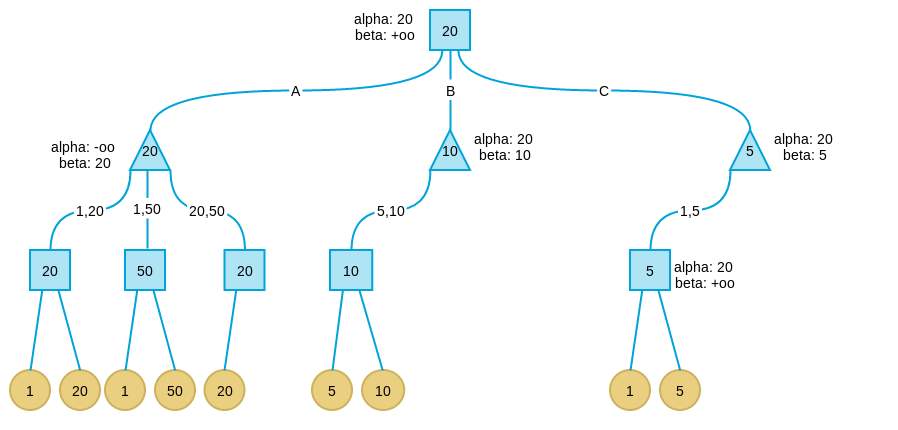
显然，白方与黑方是对立的，白方希望分值极大，黑方希望分值极小，故形象地称为极大极小搜索，其本质是一颗博弈树。

显然，任何一方的一种走法都是一个儿子节点，那么，使用DFS算法递归深搜即可，递归终点为认为设定的最高博弈层数（最低层不再博弈，改为贪心）。

伪代码如下：



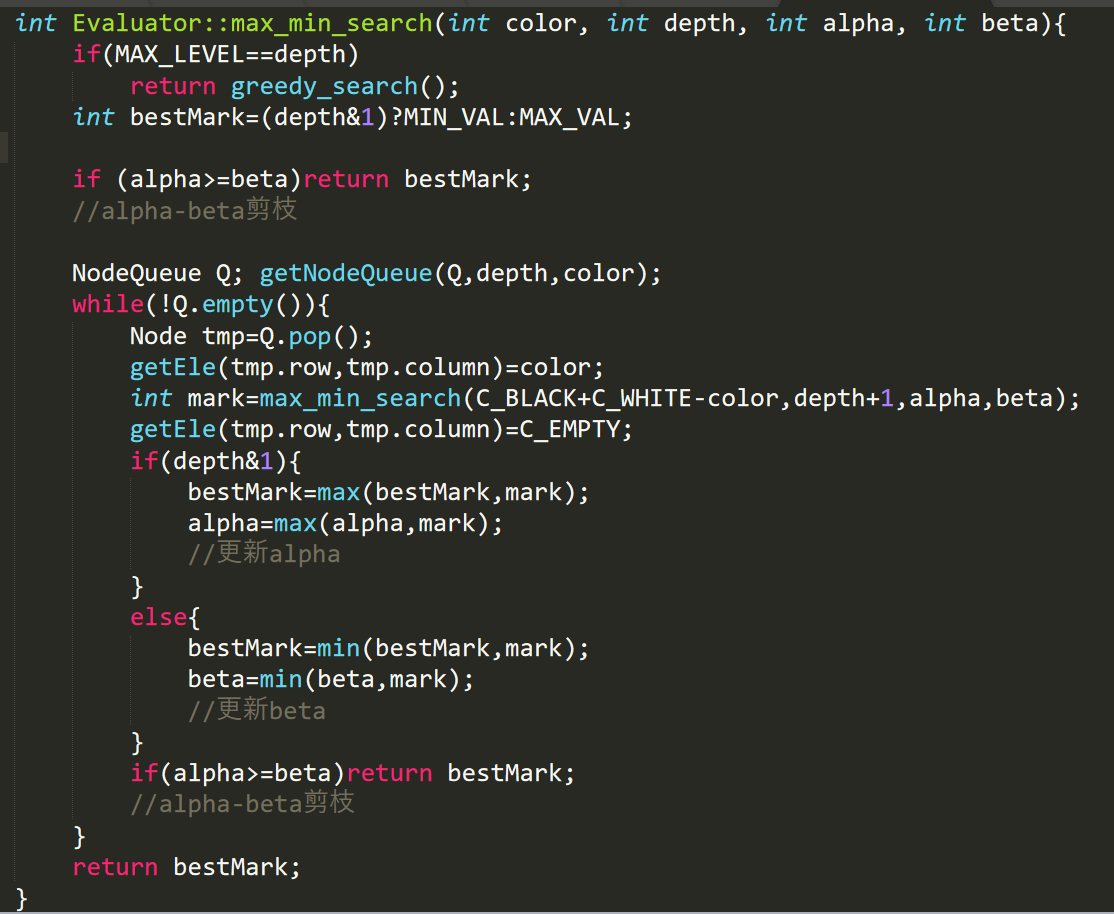
1. Alpha-Beta剪枝算法



Alpah-beta算法上是极大极小搜索的优化，本质上是一个二元博弈树上的最优性剪枝。算法为每个节点附加了两个属性alpha和beta，其根节点初始值分别为，非根节点由父节点继承而来，分别表示已经搜索到的最大值和最小值。MAX层负责更新alpha，MIN层负责更新beta，当时进行剪枝。

例子如上图所示，第二层第二个节点，alpha为20，beta为10，此时其父亲节点选择的子节点的分值一定大于等于20，而此时该节点选择的分值一定小于等于10，显然，该节点一定不会被选择，故可实行剪枝。

伪代码如下（基于上文伪代码）：

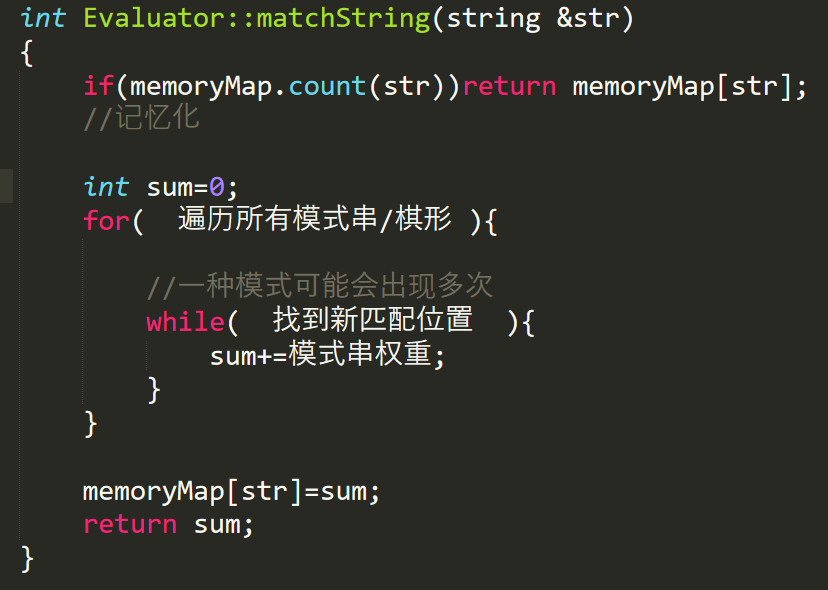


1. 哈希记忆化优化

由(1)评分算法所说，我们会频繁对每行，每列，每斜线，每反斜线的对应字符串评分，这会消耗大量的CPU资源。

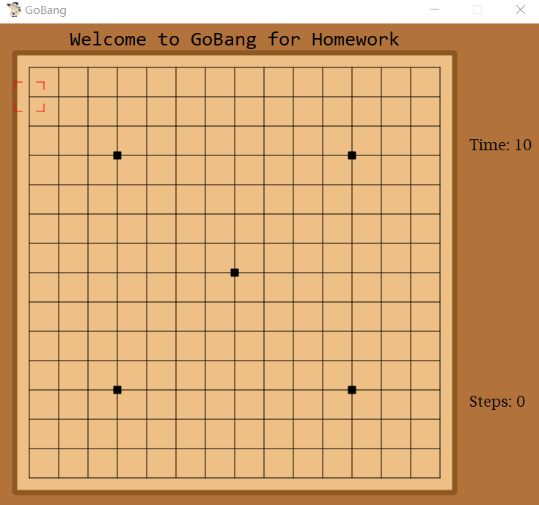
但事实上，每次我们下子时，都仅仅只改变了一个点，这样就只影响了4个字符串。但按照起初的暴力算法，每次重新计算，显得开销过大。所以可以用空间换时间，使用一个hash表在O(1)的时间内完成已计算分值的返回，这样可以分摊时间复杂度，而且在DFS博弈树频繁枚举和回溯的过程中效率提升更为显著。

伪代码如下：



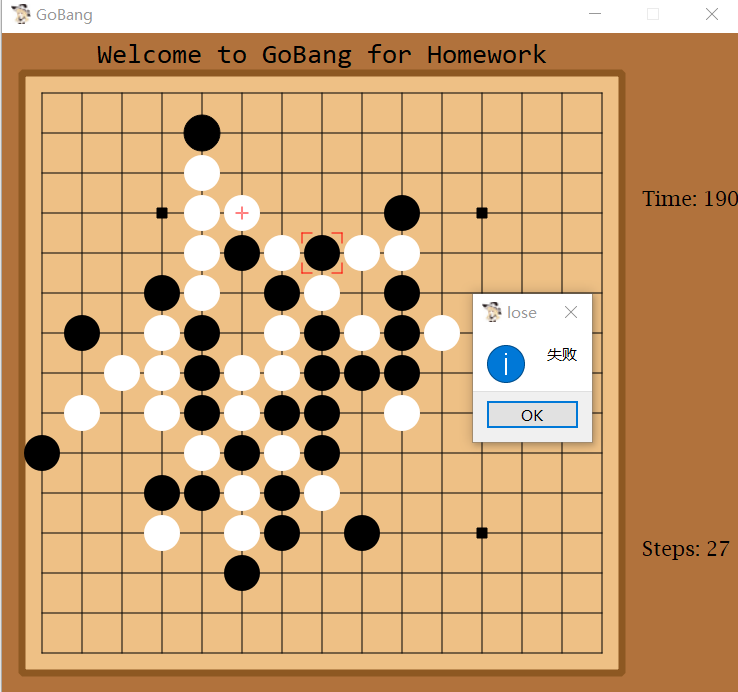
# 七、实验数据及结果分析：

**开始界面：**



可见，GUI美观清晰，对用户友好，还有光标位置，游戏时长，游戏步长等提示信息，使用简洁易懂。

**AI后手胜利，玩家先手失败：**



在五子棋的世界中，先手具有极大优势，所以有了禁手的存在。但是本程序并未实现禁手判断，AI在后手不利的情况依然可以战胜先手用户，可见已有一定棋力，应对一般玩家游刃有余。

**运行时间统计（部分）：**



可以看出,随着下子序数的增加,计算时长有增加的趋势，在搜索层数达到8层的情况下，时间却能够始终维持在1.5s内，说明算法运算效率得到了保证，减少了用户等待的焦虑，提高了用户使用的流畅性。

# 八、总结及心得体会：

优点：

1. GUI界面简洁清晰，用户友好
2. AI反应速度快，操作流畅
3. 在博弈树基础上综合运用了较多优化方法

缺点：

1. 棋形估分具有随意性。程序中所有模式串的打分都是实现者按照自己理解给出的分值，具有主观随意性。
2. 程序运行效率依赖于编译器优化。在源代码中可以看出，所有文件头均添加了O(3)优化，这是因为O(2)优化对map家族容器效率影响很大。然后开了O(2)优化之后就自然想开O(3)优化了。
3. 程序还存在小bug。虽然平均表现良好，但是偶然会出现犯傻的情况，这与程序逻辑和模式串打分有密切关系。考虑到再为此程序投入更多时间的学习收益不显著，且程序平均表现良好，故暂时搁置此问题。
4. 程序的功能较为单一。不像有些同学添加了悔棋，选择先后手等选项，该程序只实现了AI的核心功能，没有丰富这些额外的功能，显得较为单一。

心得：

对于棋类博弈问题，最基本和简单的算法就是极大极小搜索算法。但是，这远远不够，随着每层搜索范围的增加，搜索层数的层加，搜索的代价是指数增长的，因此，如何高效地剪枝就成了最为重要的问题。该程序使用了alpah-beata剪枝进行最优性剪枝；为了减少每层的不必要候选节点，使用了贪心算法筛选候选节点进行启发式搜索；同样为了减少不必要的节点，程序只在邻域内搜索；为了避免重复性的计算，使用了记忆化的策略；同时为了减少时间常数，使用了诸如预分配等多种常数优化手段。

同学们基本上一致使用了极大极小搜索和alpha-beta剪枝，但是达到的最大思考层数却不尽相同，原因其实在于每层的节点数目，和对单个节点评分的代价在每个程序中不尽相同。每层节点数目越多，评分代价越高，往往评分更准确，而思考的深度不够；而节点数目太少，评分过于随意，即使思考深度足够，也缺乏参考意义，这是一个需要小心选取平衡点的问题。

# 九、对本实验过程及方法、手段的改进建议及展望：

1. 程序中模式串的打分具有主观随意性，可以考虑使用各种最优化算法训练出最适合的参数。
2. 程序的功能较为单一，可以尝试添加悔棋等选项，丰富用户的使用体验。
3. 搜索中的记忆化还有提升空间，程序中只对单行，单列，单斜线，单反斜线的棋形进行了记忆化，可以考虑对整个棋盘进行状态压缩实现记忆化（此时需要考虑各种对称性）。
4. 程序的启发式搜索还有待提升，目前程序的启发式搜索策略是搜索贪心结果最优的前几个节点，显然还有很大的提升空间。

**报告评分：**

**指导教师签字：**