|  |
| --- |
| 电子科技大学  **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：陈南熹**  **学　　号：2023080910003**  **指导教师：俸志刚** |
| **日　　期：2023年12月17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**

**一、实验项目名称：**

五子棋游戏

**二、实验原理：**

博弈树，极大化极小值算法，Alpha-Beta剪枝算法，启发式搜索

**三、实验目的：**

通过C语言知识，实现五子棋AI，即让电脑能够与人类进行五子棋对战。

并以此掌握：

1.如何使用极大化极小值搜索算法处理博弈树

2.如何使用Alpha-Beta剪枝算法和启发式搜索优化上述算法

3.如何使用图形库（如ralib或者easyX）为游戏提供GUI（图形用户界面）

**四、实验内容：**

五子棋AI分成五个主要的功能板块，分别为对应五个文件:

main.cpp,Gobang.h,Gobang.cpp,EasyX.cpp,Stack.cpp。

main.cpp源文件包含main函数，是完成游戏的逻辑实现。

Gobang.h是游戏的头文件，包含宏定义，对标准库中头文件的引用，结构体类型的声明和函数的声明

Gobang.cpp源文件包含游戏中功能函数的定义

EasyX.cpp源文件中运用easyX图形库定义函数实现GUI

Stack.cpp源文件实现了栈的功能（用以储存每一步棋实现悔棋功能）

**五、实验器材（设备、元器件）：**

硬件平台：PC端

开发环境：操作系统Windows11

测试环境： Clion 2023.2.2和Visual Studio 2020

**六、实验步骤：**

1. 要求描述

实现一个五子棋AI，让电脑拥有一定的棋力。

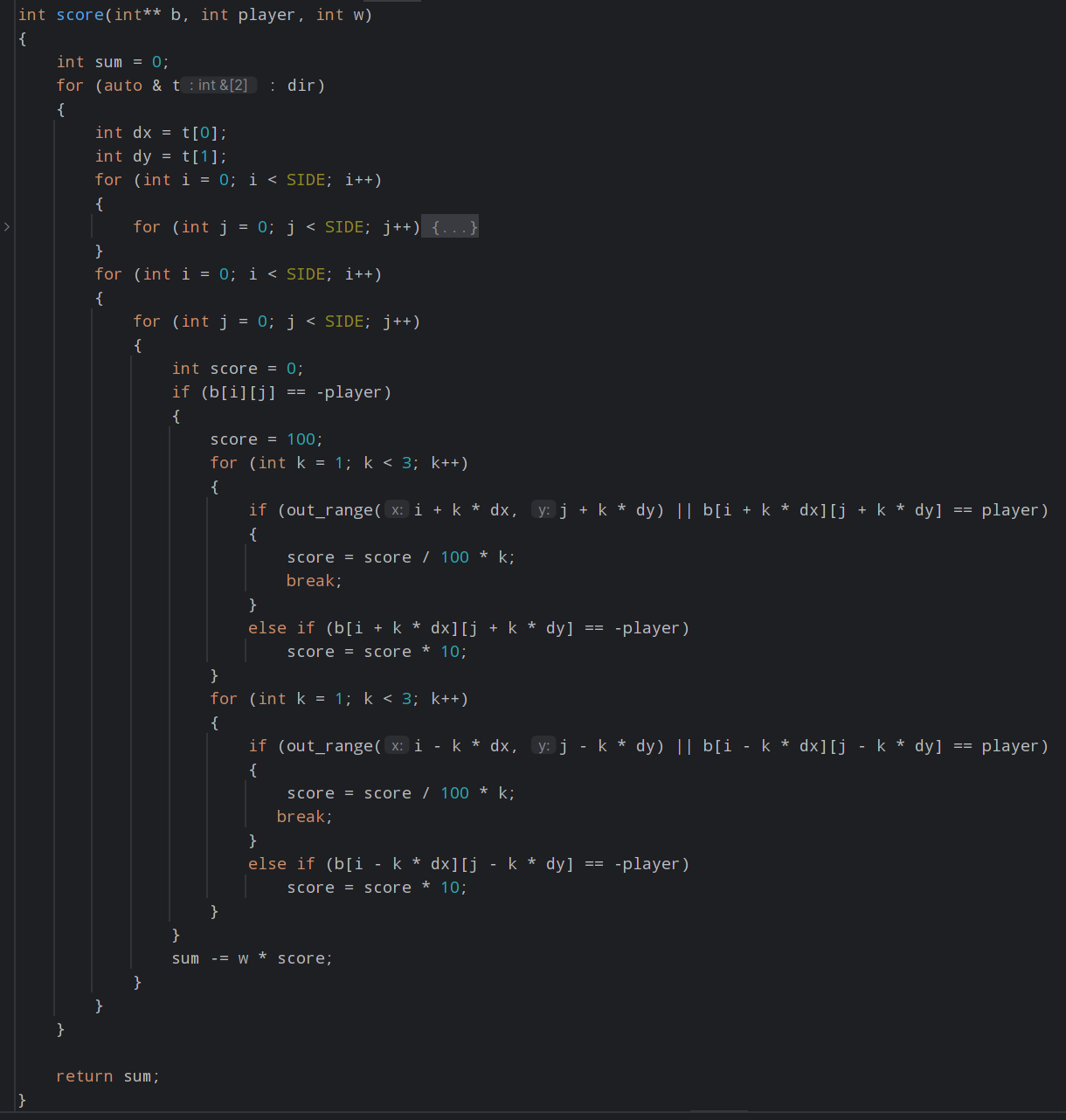
算法的概要设计与分析

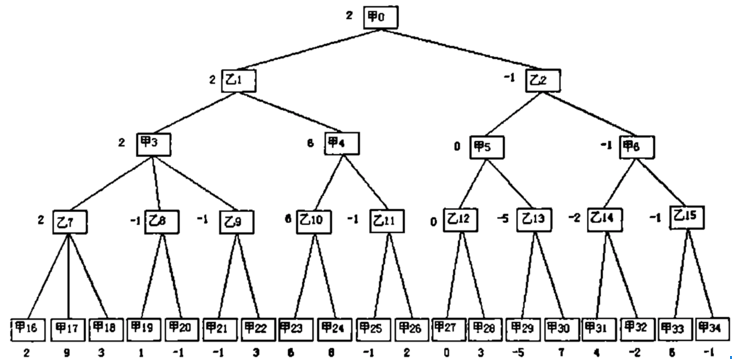
下五子棋是一个博弈的过程，将五子棋的每一步走法展开，就是一颗巨大的博弈树。假设电脑先手，在这个树中，从根节点为0开始，奇数层表示电脑可能的走法，偶数层表示玩家可能的走法。

有了对博弈树的基本认识，我们就可以用递归来遍历这一棵树。但是要想知道哪一个分支的走法是最优的，我们还需要一个评估函数能对当前整个局势作出评估，在最后一层的每一个节点（代表一种局面）返回一个分数。我们规定对电脑越有利，分数越大，对玩家越有利，分数越小，分数的起点是0。

我设计的局势评估函数是这样的：以黑棋为例，对于棋面上每一个黑棋所在的点，对以该点为中心距离为二以内的每一条直线（包括横，竖，斜），规定初始分数值为100，遍历这条直线，若点为空，分数不变，继续遍历；若为黑棋，分数值乘以10，继续遍历；若为白棋，分数值除以100，并退出循环。这样把每一条直线的分数相加，得到这个点的分数，最后将所有点的分数相加，得到局面的黑棋得分a。与此对应，也可以得到白棋的得分b，并将a-b\*w作为黑棋最终得分，其中w是调节电脑侧重进攻还是防守的参数。代码如下：

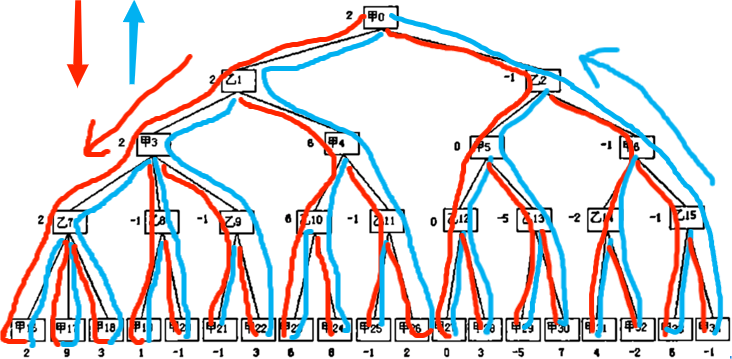
有了局势评估函数，就可以遍历博弈树了。

这是博弈树的一张示意图：



这张图中甲代表电脑，乙代表玩家。如果采用极大化极小值搜索算法处理博弈树，那么我们把电脑走棋的层称为 MAX层，将玩家走棋的层被称为MIN层。

在MAX层中，电脑要保证自己利益最大化，该层的每一个节点选择下一层中分数最高的节点，并将这个最高分作为自己节点的返回值。在MIN层中，作为假想敌的玩家应该保证自己利益最大化，该层的每一个节点选择下一层中分数最低的节点，并将这个最高低作为自己节点的返回值。这样电脑就能够搜索出在以玩家每一步都采取最优走法的前提下第n层后得分最高的落子点。



如上图所示为计算机遍历博弈树的过程，红色路线表示进入下一层，蓝色路线表示返回上一层。

这样就可以了吗？理论上没问题，但是初略估计一下就会发现遍历整棵博弈树并不现实。五子棋棋盘为15\*15，假设n=4且平均每一步有50种走法，那么博弈树最后一层就有50^4=6250000个节点，对每一个节点进行局势评估，这是相当消耗时间的。

因此，我们需要对上述算法进行优化。首先，最容易想到的是舍弃一些位置明显不合适的落子点。比如一般来说，我们落子地方两格范围内有子，也就是说，两格以内全都是空的点可以舍弃，因此我写了一个judge函数用于判断，代码如下：



其中的out\_range是判断是否越界的函数。

只是这样简单处理还不够，因此就用到了Alpha-Beta剪枝算法。

Alpha Beta 剪枝算法是一种安全的剪枝策略，也就是不会对棋力产生任何负面影响。它的基本依据是：棋手不会做出对自己不利的选择。依据这个前提，如果一个节点明显是不利于自己的节点，那么就可以直接剪掉这个节点。

前面讲到过，电脑会在MAX层选择最大节点，而玩家会在MIN层选择最小节点。那么如下两种情况就是分别对双方不利的选择：

在MAX层，假设当前层已经搜索到一个最大值 X， 如果发现下一个节点的下一层（也就是MIN层）会产生一个比X还小的值，那么就直接剪掉此节点。

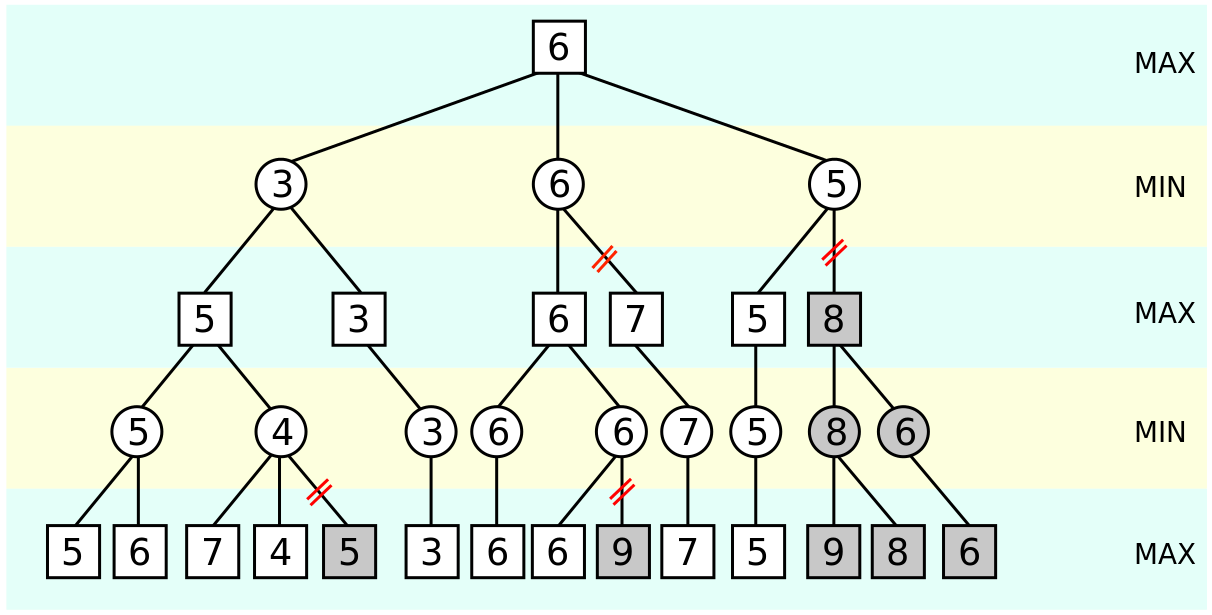
解释一下，也就是在MAX层的时候会把当前层已经搜索到的最大值X存起来，如果下一个节点的下一层会产生一个比X还小的值Y，那么之前说过玩家总是会选择最小值的。也就是说这个节点玩家的分数不会超过Y，那么这个节点显然没有必要进行计算了。

通俗点来讲就是，电脑发现这一步是对玩家更有利的，那么当然不会走这一步。

在MIN层，假设当前层已经搜索到一个最小值 Y， 如果发现下一个节点的下一层（也就是MAX层）会产生一个比Y还大的值，那么就直接剪掉此节点。

这个是一样的道理，如果玩家走了一步棋发现其实对电脑更有利，玩家必定不会走这一步。

用示意图来表示更为直观：



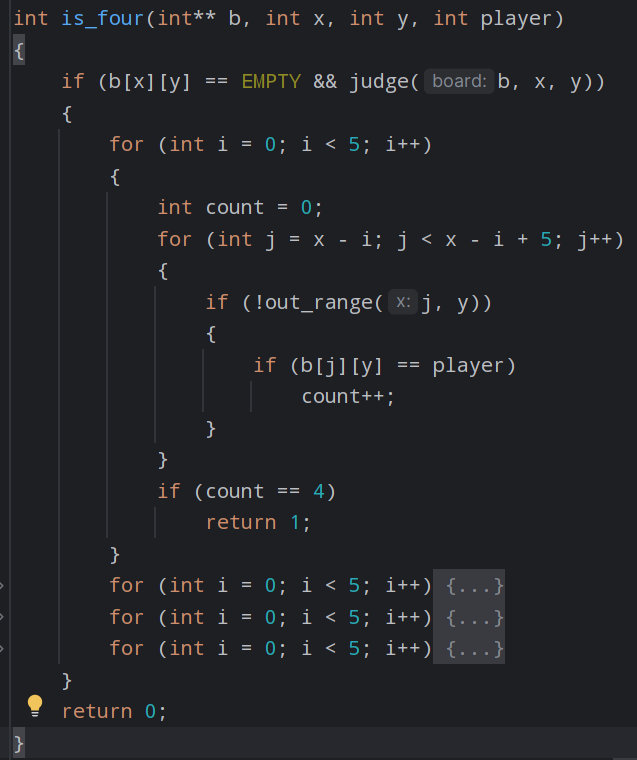
如上图所示，在第二层，也就是MIN层，当计算到第二层第三个节点的时候，已知前面有一个3和一个6，最大值至少是6。 在计算第三个节点的时候，发现它的第一个子节点的结果是5，因为当前是MIN节点，会选择子节点中的最小值，所以此节点值不会大于5。而第二层已经有一个6了，第二层第三个节点肯定不会被选择。因此此节点的后续节点就没有必要计算了。同理，该图中第三层值为7的后续节点也不用计算了。

这时，你会发现，根据上面的那张图，在第二层中，第一个节点的值是3，第二个是6。因为3比较小，而这一层的最大值会被选中，所以第二个节点也需要完整计算所有子节点。如果3和6调换一下顺序，6在前，3在后。那么当第二个节点计算出第一个子节点5的时候就没有必要计算之后的孩子了。也就是，Alpha-Beta 剪枝的效率和节点排序有很大关系，如果最优的节点能排在前面，则能大幅提升剪枝效率。

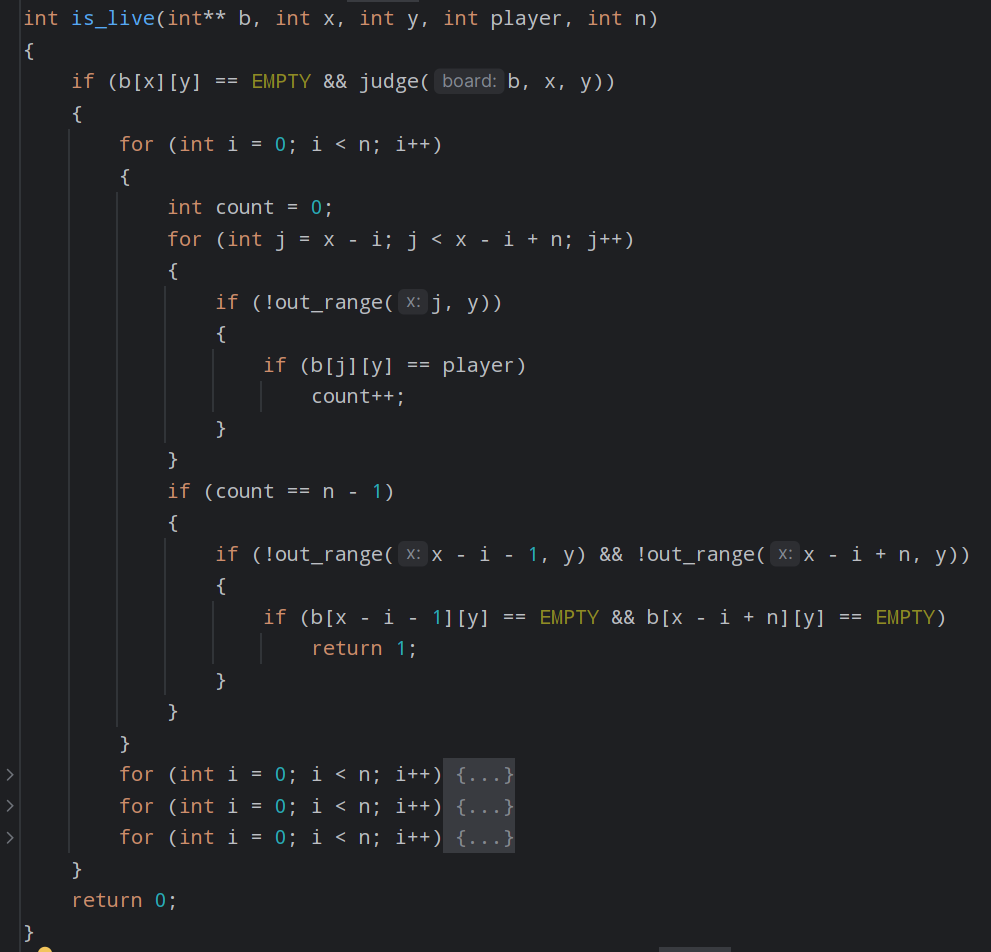
对于Beta 剪枝也是同样的道理。所以说Alpha-Beta剪枝的效率是取决于每一层节点的顺序的。 我们肯定是无法精确排序的，因为每一个节点的值并不能直接计算出来，需要递归计算子节点。 但是我们依然能对节点进行大致的一个排序。并且只要有一个大致的排序，就能很好的提升剪枝效率。这也就是启发式搜索算法的核心思想：利用启发函数来引导搜索过程，以便更有效地朝着解决问题的方向前进。

那么具体如何进行大致排序呢？我是这样做的，即只对一个点进行评估，判断该点能否成五，活四，冲四，活三或者连三，将所有的点分类，放入不同的栈中，最后按照成五，活四，冲四，活三，连三和其他的顺序依次排列，即将能成五的点放在最前面，然后是活四，以此类推。如此的话，先要写函数判断能否成五，活四，冲四，活三，连三。代码如下：

成五：



活四和活三：



冲四和连三：



那么接下来，我们尽可以用极小化极大值算法，结合Alpha-Beta剪枝算法和启发式搜索，用递归写出一个函数，得到下一步的最优落点。

代码如下：

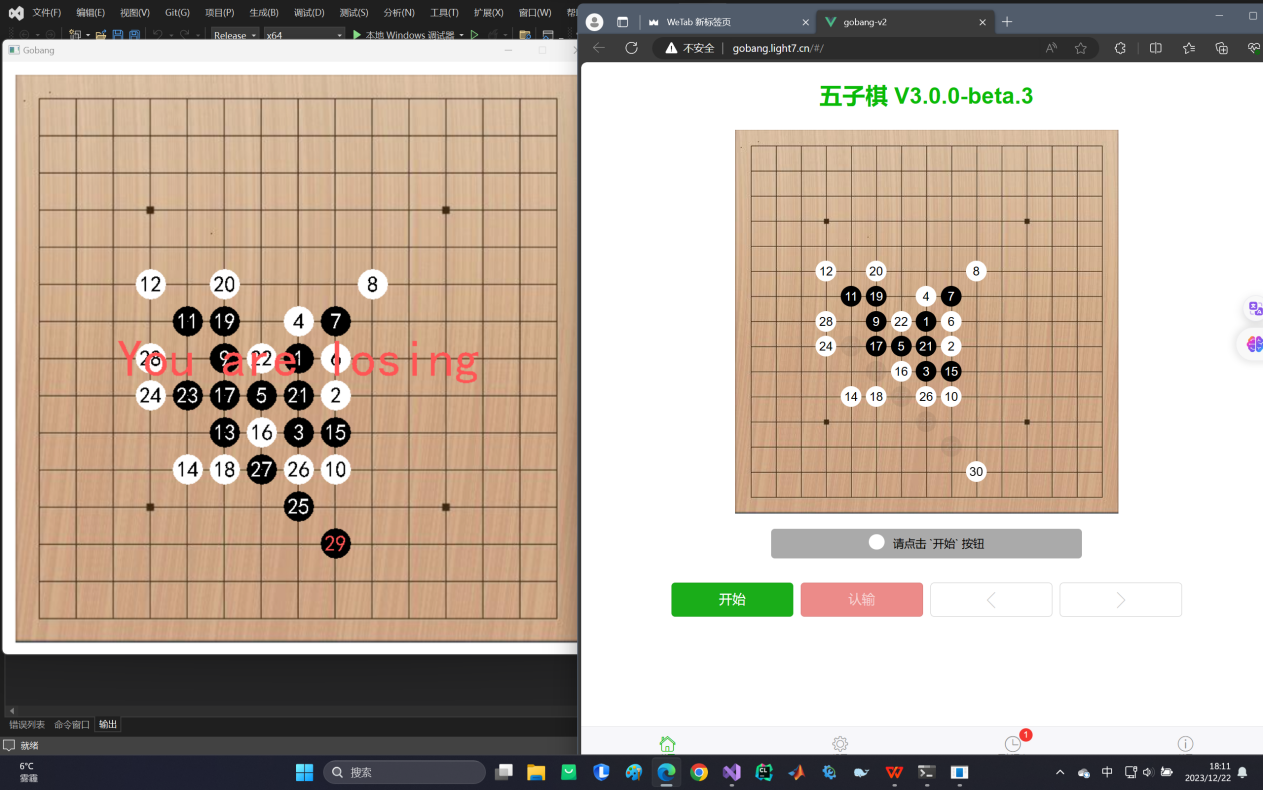


其中的Row和Col是int类型的指针，用于储存所得到的最佳落点的横纵坐标。至此，五子棋AI 的核心算法大体上完成了。

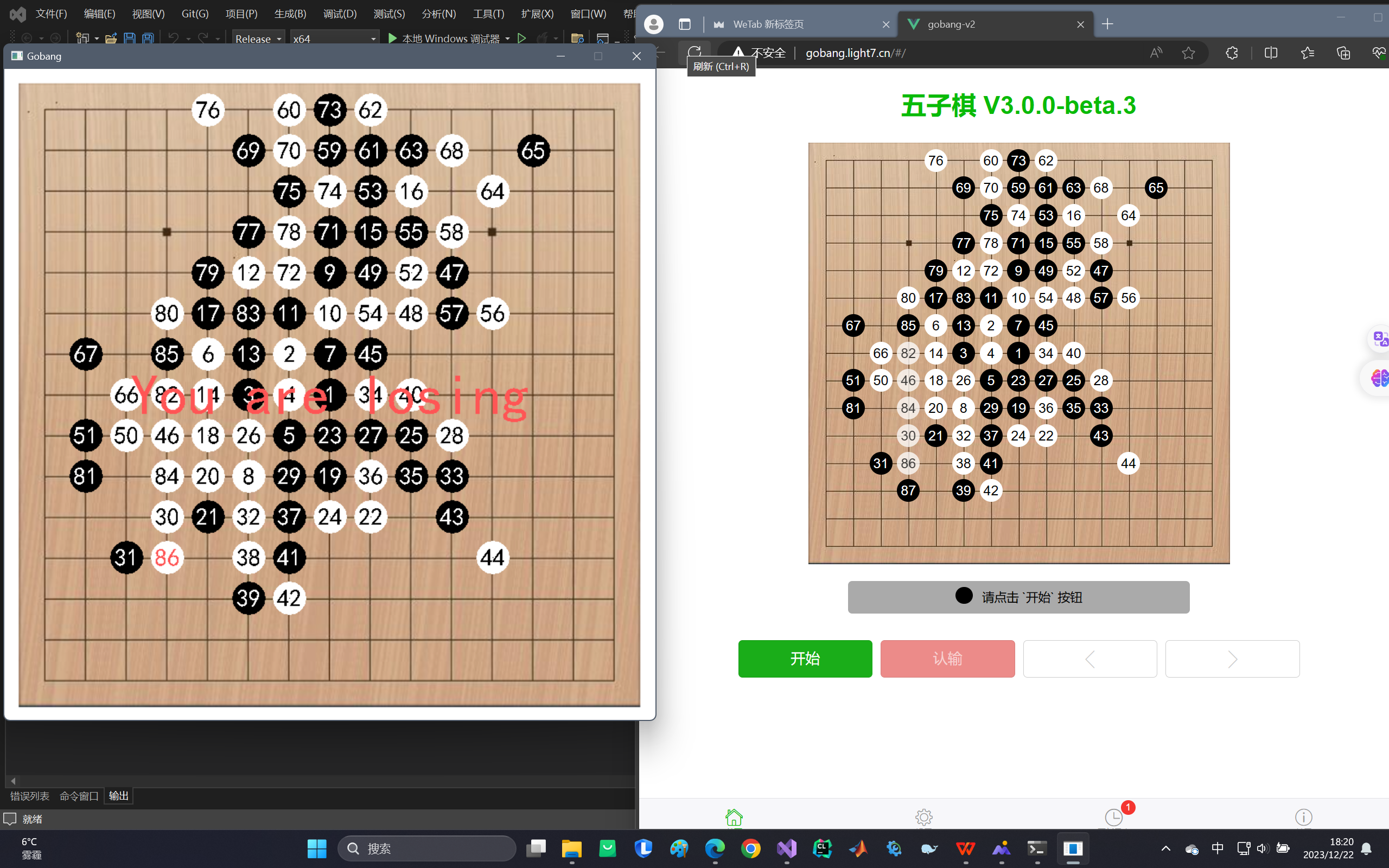
1. **实验结果分析：**

最终我实现了四层的五子棋AI，并且通过对w的调整，最终确定w=6时有较好的棋力效果。

下图是在我的AI先手的情况下与另外一个五子棋AI对弈的结果，最终获得了胜利。

****

下图是我的AI后手的情况，仍然获得了胜利。

****

通过大量的测试，如果对手是普通玩家，没有研究过五子棋，在先手的情况下，

我的AI胜率大约为90%，在后手的情况下，我的AI胜率大约为70%。