****

**C语言**

**实验报告**

|  |  |
| --- | --- |
| **主 题** | **五子棋AI** |
| **学 院** | **计算机科学与工程学院（网络空间安全学院）** |
| **专 业** | **计算机大类** |
| **姓 名** | **汪世鉴** |
| **班 级** | **2022080905** |
| **学 号** | **2022080905012** |

实验报告

题目：

本次实验的题目是在五子棋游戏中加入人工智能，实现计算机持棋落子。

实验目的：

实验的目的是掌握在五子棋游戏中加入人工智能，实现计算机白棋的落子，通过算法的优化和分析来使得游戏更加具有挑战性和难度。随后加入计算机黑棋的落子，并实现了计算机黑白子对弈的功能。

实现的功能：

1：UI界面。

2：落子功能。

3：估值功能。

4：棋型判断功能。

5：悔棋功能。

6：计数功能。

7：禁手功能。

8：位置减枝功能。

9：算杀功能。

10：防御功能。

设计思路：

此项目主要分成UI与AI函数。

每个人的UI各有特色，但对棋力无影响，所以按自己喜好设计即可。

决定棋类游戏AI棋力高低的主要因素是对场上棋型的识别以及对各类棋型的估值，以及对接下来局势的估计。

从这个思路展开，我首先设计了判断棋型的函数与对相应棋型进行估值的函数。而后发现只对当前局势进行估值的话棋力有限。所以我一极大极小值搜索为基础设计了深度为四的搜索函数，用来对局势进行更好的判断。而为了解决深搜带来的时间增长，又加入了位置减枝的功能，至此基本框架建立完成。

随后为了实现的更好的游玩体验。之后又陆陆续续加入了悔棋，禁手，算杀，进攻，防御的功能。

实验过程：

备战：使用VS2022作为编写工具，使用必应作为搜索工具，使用easyX作为图形库，开发环境选择了VS2022默认的环境。

代码实现：

头文件（wuziqi.h）：

用一个头文件包含了需要用到的头文件库，宏定义与全局变量声明，函数声明。

初始化函数文件（main.cpp , InitBoard.cpp，Game.cpp）：

实现棋盘跟部分变量的初始化。其中Game(int t)实现了不同游戏模式选择的功能。

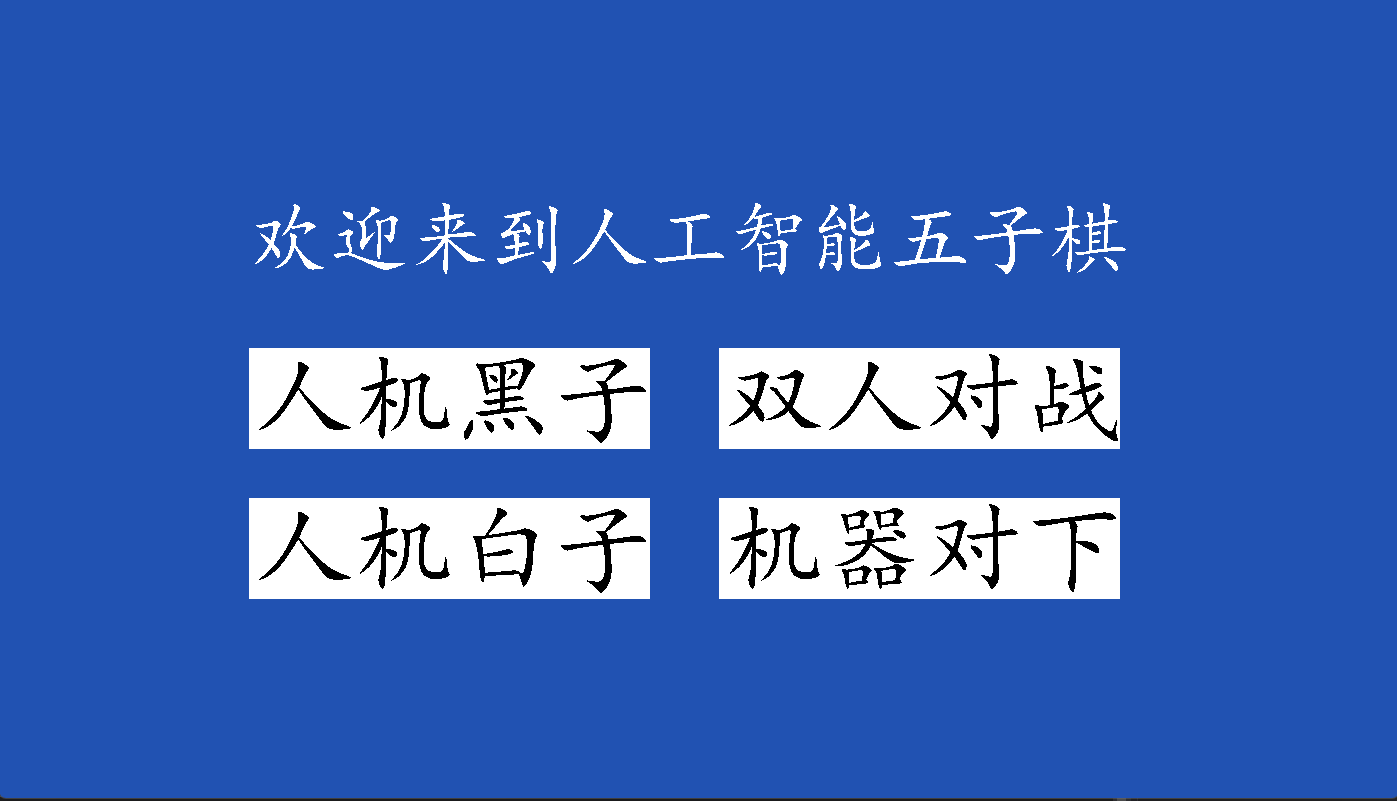
UI文件（UI.cpp，ButtonClick.cpp）：

本次UI实现以esayX作为图形库，配合在必应上搜索到easyX的对应的绘图函数进行本产品中的UI函数的实现。

具体的分为启动时的游戏界面跟对战时的游戏界面，我没有调用网图作为资源，纯粹使用easyX库中的函数实现了UI，所以比较简陋。

InitUI(void)：该函数绘制了启动界面的UI

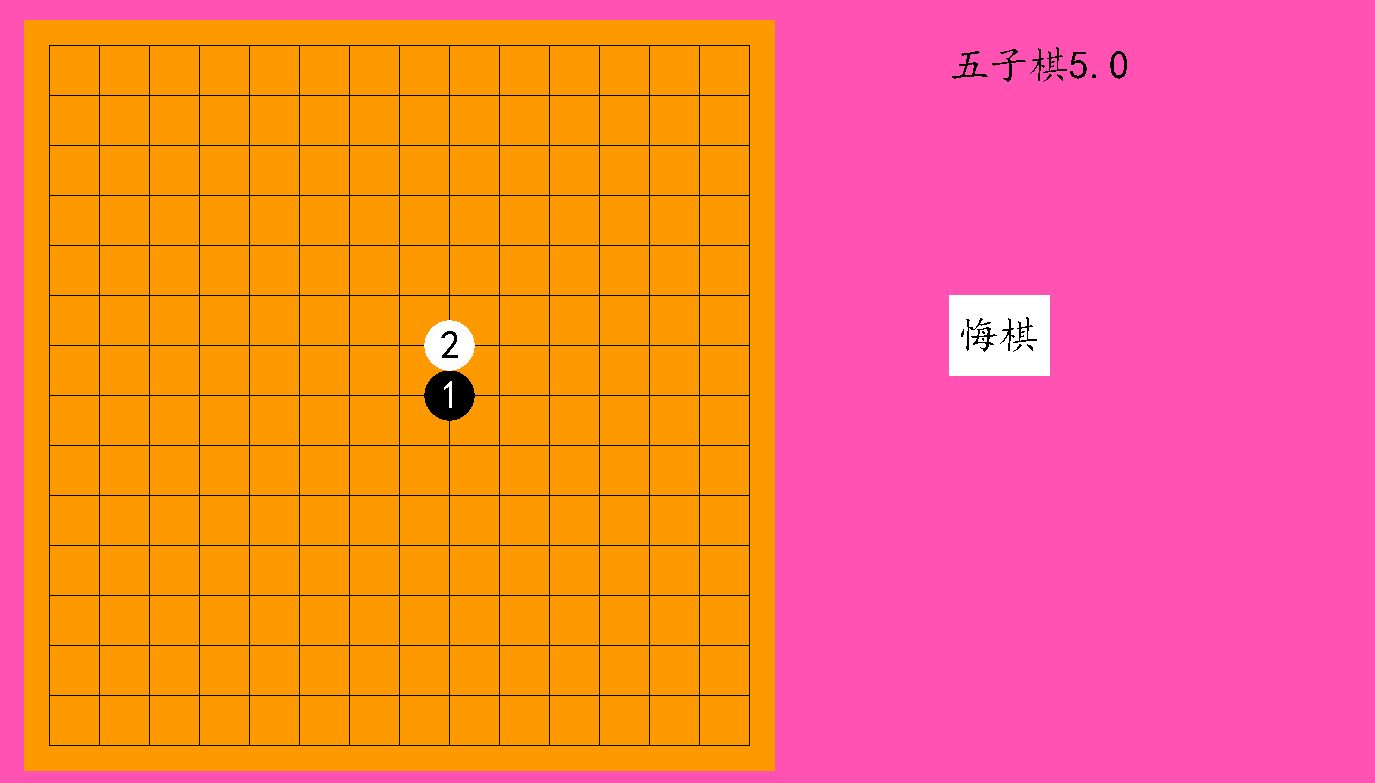
启动时的游戏界面如下图



在这个界面只要点击对应功能所在的白框就能跳转到下棋的UI界面

GameUI(void)：该函数绘制了对战界面的UI

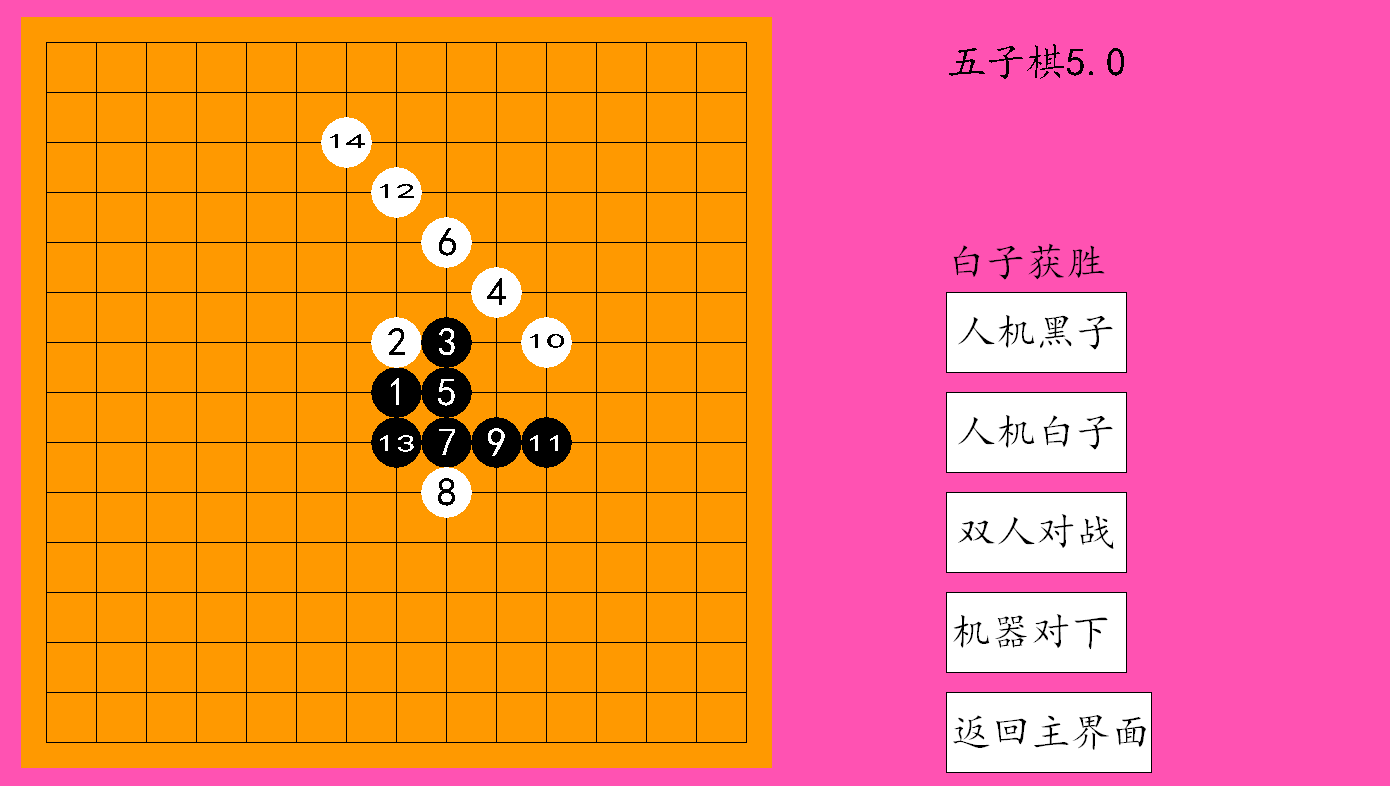
运行时的UI界面如下图



左侧是对战的棋盘，可以看到棋子对应的步数。右侧的悔棋按钮可以实现全局悔棋的功能。

ButtonClick2( )：该函数兼具了按钮跳转的功能与绘制UI界面的功能。

对战结束后的UI界面如下图



右侧的悔棋按钮消失了，出现了获胜方的信息。同时弹出了选项，可供玩家选择接下来想玩的模式。

以上就是该产品中的三个UI，虽然简陋，但基本功能都已经具备，可以说是麻雀虽小五脏俱全。

UI辅助函数文件（ButtonClick.cpp ， countstep.cpp）：

ButtonClick.cpp中的函数实现了UI界面的跳转。具体方法是调用easyX中对应的获取鼠标左键输入位置的函数，实现了当鼠标点击对应图标时可以跳转到对应的UI界面。跳转时会重新绘图覆盖掉原有的UI界面。

count\_step(int x, int y,int color)：  
 实现了计数功能与步数的绘制，以及把落子的坐标push入栈中

悔棋功能文件（Chess\_Repentance.cpp中的函数）：

为了下棋体验，也为了方便测试代码。我用俸老师上课讲的stack函数实现了悔棋功能，也算是勉强使用了一些高级的数据结构。

我一开是是想用笨方法存储位置实现一步悔棋的功能，但实际测试后发现运行起来很慢且容易出Bug，随后我想到了俸老师上课讲的stack。使用后发现响应速度极快，并借此实现了整局悔棋的功能。

这个功能实现之后也极大的提升了我调试程序的效率与游玩乐趣。算是我的好帮手。

Compete\_Chess\_Repentance()：

是利用stack特性将count\_step函数中最后存入的位置Pop出，以此实现了悔棋功能。同时配合绘图函数将对应位置的棋盘通过重新绘画覆盖的方式实现还原棋盘的效果。  
悔棋辅助函数（stack.cpp中的函数）:

实现了创建stack，Pop与Push等操作。

落子函数文件（Move.cpp ，PlayerMove.cpp）：

使用easyX中对应的获取鼠标左键输入位置的函数实现了落子的功能，同时调用Compete\_Chess\_Repentance()实现悔棋功能。

Move(void)：实现了双人玩家对战的落子。

PlayerMove(int color)：实现了人机对战中人类玩家落子的功能。

set(int row ,int col)：实现了落子时绘制图形，计数，与胜负判断。

电脑落子功能文件（AI.cpp）:

AI(int color)：

能对局势做出一定判断然后落子，基本思路是对可能落子的点全部搜索过来，然后根据我方成五，敌方成五，我方VCT冲四算杀与敌方成活四，我方活三算杀的情况分成不同优先级去判断是否在对应位置落子。若以上情况都不存在，则视为普通情况，则通过极大极小值搜索找出四步之后的最优位置落子。

棋型判断功能文件（Chess\_type.cpp ,Opposite\_shore.cpp ）:

为了更好的判断场面局势，我的棋型判断不局限于一个方向上的15类基本棋型，而是囊括了一个方向上的基本棋型及其各类变种棋型，总计40多种。

为了实现这一功能，我的思路是先找到一个已经落子的点，然后根据它的颜色棋盘上的落子情况分为三类：同色，异色，空格。然后以Chess\_type(int row,int col,int direction)为基础，同时调用Opposite\_shore(int row,int col,int direction,int color)辅助做出棋型的判断。

判断功能文件（check.cpp，defend.cpp，forbidden.cpp，Judge.cpp ，kill.cpp）：

fobidden(int row, int col)：

在棋牌类游戏中先手往往意味着巨大优势，因此各类游戏中往往会给后手一些补偿来减少这个差距。五子棋中的先手优势更是巨大，事实上，在没有禁手的情况下五子棋存在先手必胜的下法。因此出现了先手的禁手。

该函数调用Chess\_type函数，实现了常见的三三，四四。长连禁手。当触发禁手时虽然没有文字提示，但会让在这个点的落子无效化，以此来实现禁手功能。

check(int row, int col)：

通过调用Chess\_type函数，对当前落子点进行一个简单的判断，包括成五，活三，冲四。服务于输赢判断与算杀判断。

Judge(int row, int col)：

调用check函数进行输赢的判断。

thedefend(int row, int col, int color)：

调用check函数实现对对手成五与活四的判断，实现对这两种情况的防御。

Kill(int depth ,int player,int color)：

利用极大极小值搜索的思路，寻找能实现VCT算杀的点，因为计算数据少所以能实现10层算杀。具体思路是在我方落子的时候寻找能冲四或者活三的点并落子进入下一层，直至算杀失败或者算杀成功（同时有两个以上的活三或者冲四，或者直接成五）。在敌方落子的时候调用defend来防止我方的成五或者活四。

估值功能文件（min\_max\_dfs.cpp，Settled\_Situation.cpp，SituationValuation.cpp，Location\_pruning.cpp）：

min\_max\_dfs函数利用极大极小值搜索并在子节点处调用Settled\_Situation函数进行估值，来找到4步之后的最优落子点。这里直接上代码：

int min\_max\_dfs(int depth, int player,int color, int a, int b){

int key;

int v;

if (depth == DEPTH + 1){

return Settled\_Situation(color);

}

if (player==1) key = b;

else key = a;

for (int i = 0; i < ROW; i++){

for (int j = 0; j < COL; j++){

if (board[i][j] == 0 && vision[i][j] != 0){

if (player == -1){

board[i][j] = color;

Insert\_Neighbor\_Location(i, j);

v = min\_max\_dfs(depth + 1, 1,color, a, b);

if (v > a){

a = v;

if (a >= b){

board[i][j] = 0;

delete\_Neighbor\_Location(i, j);

return 1e9;

}

key = v;

}

board[i][j] = 0;

delete\_Neighbor\_Location(i, j);

}

else

{

board[i][j] = -color;

Insert\_Neighbor\_Location(i, j);

v = min\_max\_dfs(depth + 1, -1,color, a, b);

if (v < b){

b = v;

if (a >= b){

board[i][j] = 0;

delete\_Neighbor\_Location(i, j);

return -1e9;

}

key = v;

}

board[i][j] = 0;

delete\_Neighbor\_Location(i, j);

}

}

}

}

return key;

}

Settled\_Situation(int color)：

在min\_max\_dfs的子节点处调用该函数，通过对每个点调用SituationValuation函数实现对该点的估值，最终求和计算出棋盘总体的估值。

SituationValuation(int row, int col, int color)：

对单点进行估值，分为两部分一部分是调用棋型判断函数进行该点棋型的判断；另一部分是按棋型跟颜色给出相应的估值。此处估值我只按基础棋型的死，半死，活再加上对应的棋子数分为12类，从死一到活四，再加上成五，一共13类估值，变种棋型的等级与对应的基础棋型等级一致。并且根据棋子颜色选择不同的估值数组（虽然因为我是臭棋篓子，不太会估值，导致不同颜色的估值数组的元素基本一致）。

Neighbor\_Location(int row, int col，int t)：

为了提高min\_max\_dfs函数的速度，我依照一般人落子时的规律对可能落子的点位进行了一个位置减枝，把落子点为中心5×5的范围划为可能落子的区域。利用一个初值为0发二维数组vision[][]去存储这些区域，当数组值不为0时代表着该点为可能落子的区域。t的值代表着是插入还是消除这些区域。

代码优化：

本项目经历多次重构与去重，从原先完成时的2500多行优化至1100行左右。

SituationValuation函数原先只有15类基本棋型的判断且棋型判断与估值两部分杂糅在一起，在优化过程中先把原函数中棋型判断的部分取出形成独立的棋型判断函数并在判断中加入了基本棋型的变种。同时还将原函数改成按颜色分开赋值来优化棋力。

Compete\_Chess\_Repentance函数原先利用两个全局变量实现一步悔棋的功能，之后在课上学了栈的操作，将这个函数改造成利用栈来实现整局悔棋的功能。不仅功能得到强化而且在代码也变得优美了。

AI函数原先是简单的调用了局部估值的函数进行落子判断，之后逐渐地扩充了dfs，局部减枝，进攻，防守，算杀等模块，逐步增加棋力。

Set函数是原先不存在的函数，它是我从落子函数中（AI，Move）中逐步提取出来的加在一起，形成的新的函数，集成了原先落子函数中共性的部分。

Neighbor\_Location函数原先是将一个一个点位单独枚举出来的函数，将近160行代码，之后突然发现利用一个双层循环加条件判断就能实现原有功能。最终在效果不变的情况下讲代码压缩成了简洁的16行。所用行数直接去掉了一个零！

Fobidden，thedefend，check三个函数都是调用棋型判断函数来实现特定棋型判断的功能。理论上能将三者和而唯一，只需调用一次棋型判断函数。但是我为了显著区分出他们各自不同的功能，没有选择将他们合在一起，而是各自独立组成函数。

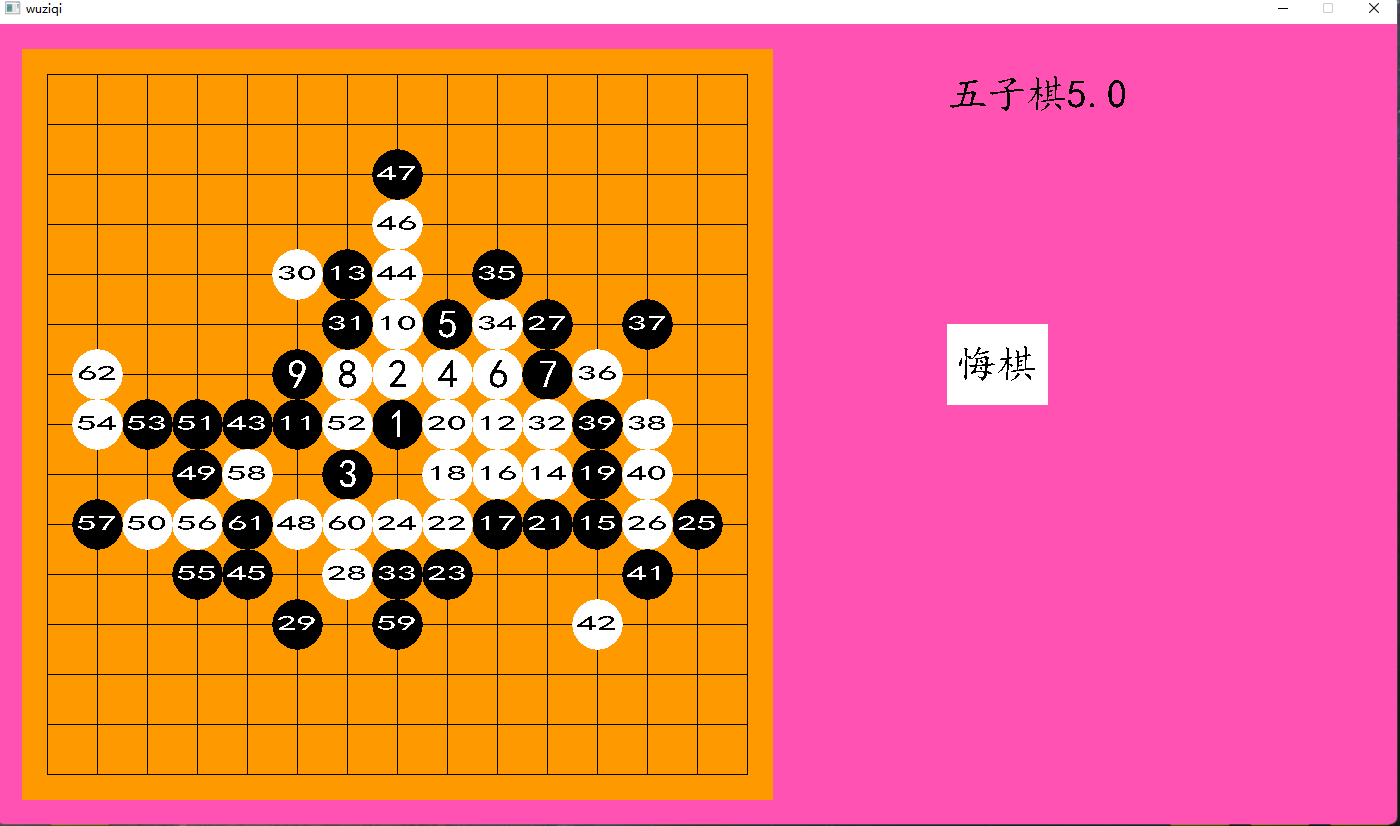
实验结果：

通过实际的目前实现了人人对战，人机对战（黑子），人机对战（白子），电脑对下。四个大功能。

在落子情况不极端，运行前10步时，电脑AI时运行时间的反应时间基本在一秒以内。

五子棋的棋力方面，我叫了几位同学对战过，基本没下过（都是臭棋篓子）。然后我在他taptap上下载了一款五子棋游戏，并轻松战胜了其中的普通难度（我自己没下过），在困难难度中下到了6,70步不分胜负。

下图为其中一次在困难难度下本程序执白子有禁手的情况下，把手机五子棋app下到突然无法响应的情况。



参考文献：

CSDN上的实现五子棋AI的帖子，实现的语言包括但不局限于C语言，看的数量太多，无法全部举出，仅举出印象特别深的一个帖子：[五子棋AI算法\_lihongxun945的博客-CSDN博客](https://blog.csdn.net/lihongxun945/category_6089493.html)。

还有这位博主的博客[五子棋AI强化：VCT、VCF算杀 | 小毅博客-分享的不只是代码更是生活！ (xeblog.cn)](https://www.xeblog.cn/articles/89)。

还有零零散散的知乎上的对于估值算法实现讲解的帖子。

感想：

本次实验用C语言实现了初具智能的五子棋AI，棋力强于一般人，逊色与五子棋高手。从实验一开始时只知道一个目标，不知道如何实现，然后经历不断去网上看博客，询问老师，师兄，自己独立思考的过程，逐步将需要实现的功能拆解，并将功能具化成一个个模块，以此来实现实验的目标。

在这个实验过程中我深刻体会到了调试（测试驱动）的重要性，举个例子：在一开始我的落子程序出bug时我怀疑是某个循环出错，但不知道具体是哪个节点，如果用最直接的断点去触发循环语句，VS会在几次循环后停止运行，无法知道bug的位置，只能慢慢地在纸上整理逻辑去查找bug。之后我学会了条件断点的用法，让我在后期的调试过程中节省了许多不必要的时间。

代码不是写出来就完事了，相反是是越重构越优美的，在一次次的重构代码的过程中我体会到了重构的好处。我原本功能未齐全的程序有近3000行代码，在我一遍遍重构之后，现在这个完成品只有1100行代码，这并不是他的极限，还有精简的余地。同时我也逐渐喜欢上了重构，以位置减枝为例，经过多次重构优化它从原本的160行变成了现在的16行，变得简短而优美。

这是我第一次从0开始逐步搭建出一个项目，让我深刻体会到了“项目是程序员的孩子”这句话的含义。在它犯错时，虽然骂它怎么这么不争气，但静下心以后还是耐心地去慢慢查找，纠正它的bug：在它成长时（添加了新功能并运行成功时），你会感到无比的欣慰与高兴，随后玩的不亦乐乎。虽然我是一个臭棋篓子，虽然我跟它下屡战屡败，但我仍玩的十分开心。

当然这个项目还有些缺点，比如棋力始终受限于我那并不精确，手搓的估值。原本有一个计划是通过随机函数赋值，让两个程序对下，逐步淘汰不精确的估值，留下更优秀的估值。但最后在各种因素下让这个计划不了了之了，算是我这次实验的遗憾。

同时我要感谢那些耐心指点我的学长，尤其是我的导生:周仔谏。他给了我很多指导与帮助。

最后，感谢俸老师能耐心地看完我的实验报告看完，祝愿 俸老师身体健康，工作顺利！也希望之后俸老师的班级办得更好。