**电子科技大学**

**计算机科学与工程学院**

**标 准 实 验 报 告**

**实验名称 五子棋**

**姓名 周正奇**

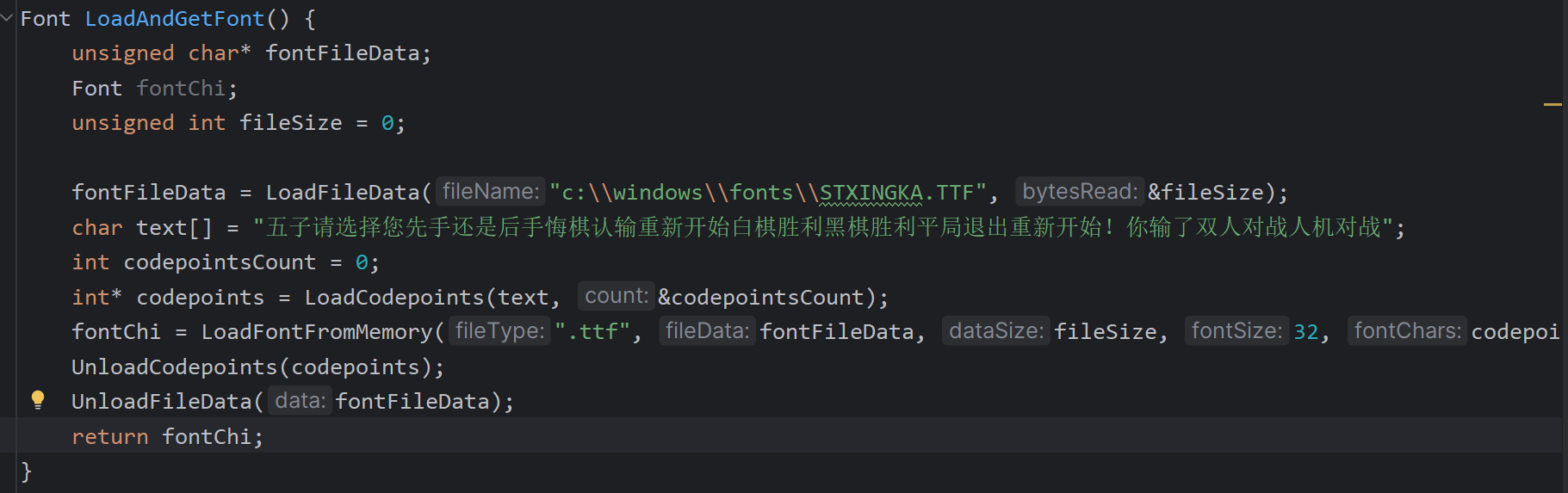
**学号 2023080904022**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

# 第一版：

1. 在帮助下搞定了raylib的配置，但是具体里面是什么，怎么用，不知道
2. 尝试在B站大学找相关教程失败，自己根据给的那个例子加自己的理解开始画第一个棋盘
   1. 下载第一个完整图片想用，但总觉得不舒服，感觉什么都没学到，决定自己找背景图，自己画格子，画按钮，贴棋子图片，修改格式，大小
   2. 摸索下画出了棋盘，开始调试寻找坐标，找棋子的中心，对上该有的坐标
   3. 用1、0表示棋盘是否有棋子，额外增加棋子颜色黑色白色
   4. 为了使棋盘划线更清晰立体，选择画两个像素
   5. 再根据玩家先手情况不同绘制不同颜色棋盒。
3. 第一次尝试，选择只使用了一个结构体，记录棋子的坐标，并没有预先思考太多逻辑，选择写一步是一步，用数字设置一切状态，搞得自己头昏脑涨，把基本下棋，判胜负搞定，开始加入菜单，开始研究按钮的实现，这时候发现画按钮没法画出来中文字体，画出来的是杠或者空白，换了字体文件还是不行，最后只能自己研究进行字体转换，把指定字转换为码点，转换后可以正常输出中文了



1. 接下来就是胜负的判断，这个还是挺简单的，配合上交换棋手

这个交换棋手最开始是判断当前步数是否是奇数来判断是第一位下棋者还是第二位，后面发现了枚举的好处就改了。

1. 功能键实现：返回主菜单，直接goto回到最初菜单函数。
2. 悔棋：（涉及到栈，第一时间待实现）

认输：弹出结束菜单-操作只有退出

重新开始：棋盘置空，棋手为1

6. 当实现了这些去尝试双人对战时发现了问题：越下越卡，把16G的电脑内存跑没了。当时很奇怪，尝试自己解决，问百度，B站，谷歌，人工智能都失败了，我去问了咱们班某位同学，他告诉我先试着把代码改改，数字用太多了会晕，建议尝试枚举，从此开始了第一轮改写

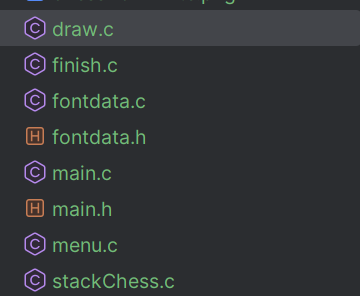
# 第二版

在第一版的基础上改进了代码，开始使用枚举表示状态。游戏模式，棋子颜色为黑白和空，省去了一次判断，在改进过程中发现之前代码很多错误之所以能跑是因为负负为正，出来正好可以用，但在需要在两个进程之间加东西就出问题了，所以又重新修改。

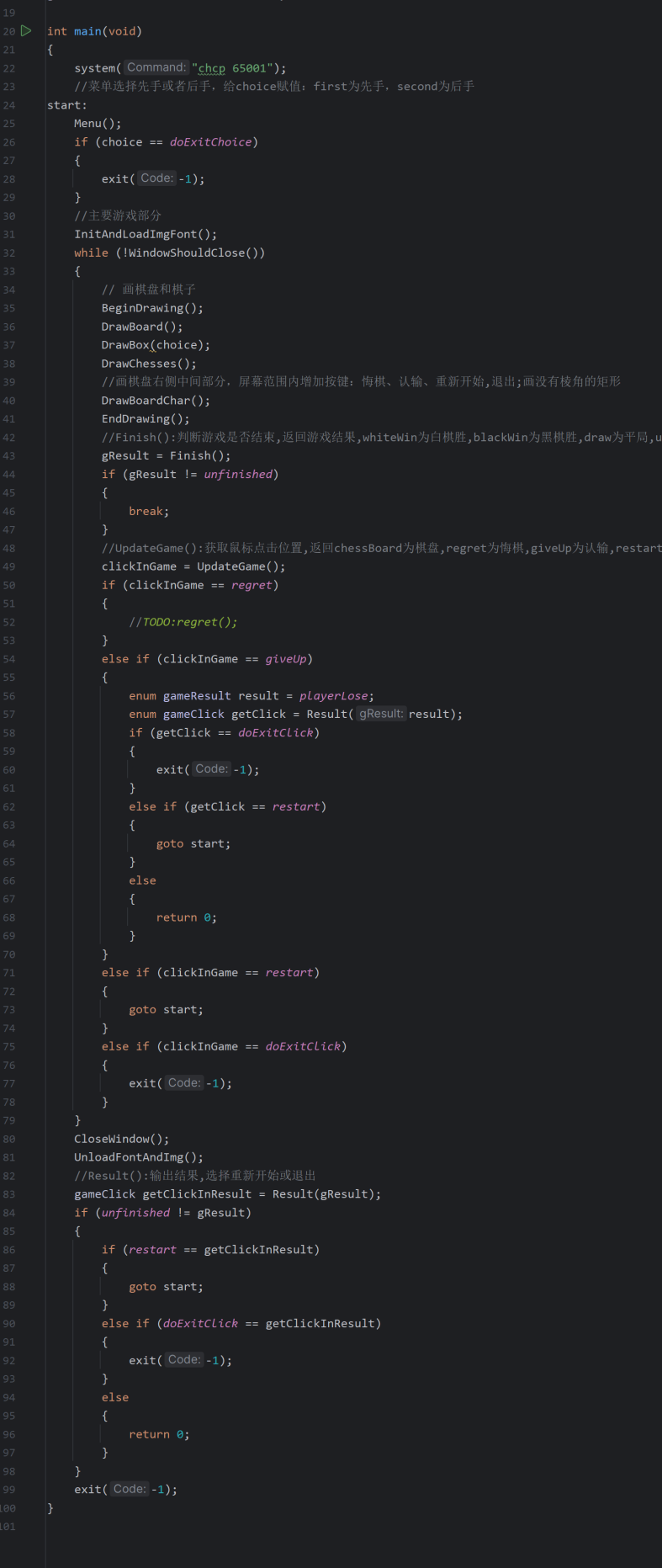
将代码整体翻修之后发现那个内存跑满的问题还在，还在调试过程中发现了新的问题

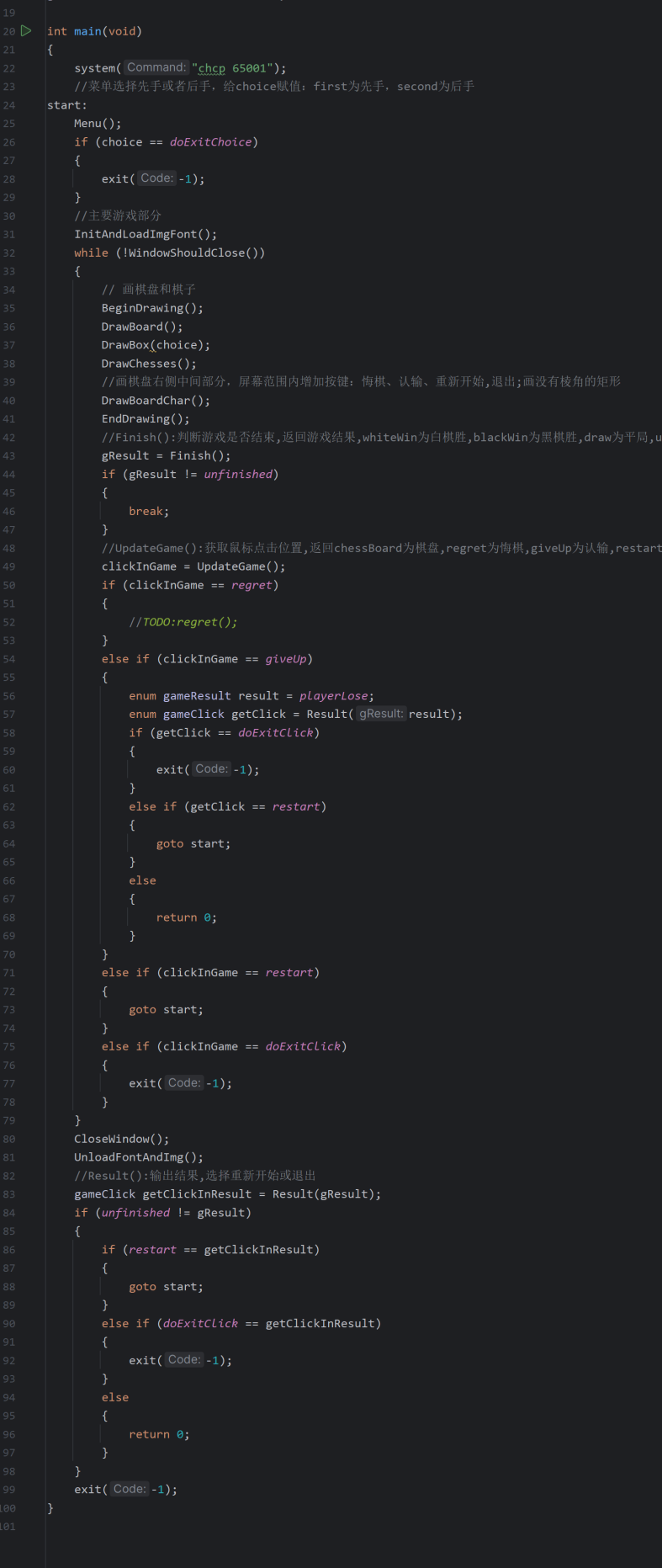
1. 在用右上角关闭结束窗口时会报错，并且程序沿着goto语句又跑一遍
2. 控制台一直有输出，但是自己用单一图片测试却没有这个情况
3. 询问滕鎧泽，让他帮忙看，他就直接告诉我我的程序逻辑太差了，没有帮我看的想法

于是开始了解决，从网络上学习和询问AI找到了一个还不错的结构，开始了从头写，因为保留了原版程序，在发现控制台输出问题时便注意了一下输出内容，最开始是以为是修改字体时造成了内存泄漏但是修复后发现是不断在重载图片，每一次画图都会重载一次所有图片，但是这个图片不像变量会被销毁于是内存爆了，于是便在新的版本里修改了这个bug



为了写起来效率高一些，只分了这几个文件，但是其实分的很差，屎山堆在draw函数里





当时的主函数，用了大量的if，也导致看起来很混乱，以为需要使用goto就把主要逻辑都放在了main函数里，

待实现悔棋：利用栈存入之前下棋的位置，

# 第三版

## 代码实现：

为了调整方便，直接把图片文件放在了代码的文件夹中

### 第三版的主函数：



开始前先InitGame();初始化相关设置

将游戏更新的相关内容全部放在了UpDateGameState();函数中，在退出游戏前DoFinish();释放相关空间。



这个函数再根据不同的游戏状态进入不同的窗口，就避免了goto语句造成的逻辑紊乱，

MenuWindow()：

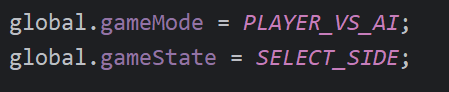


利用DrawMenu()画出菜单，以主页面做背景用用HandleMenuClick();来处理点击。根据点击不同，再进入不同的进程。



### 人机对战：

当选择人机对战，更新游戏模式和状态，再进入选择先手界面



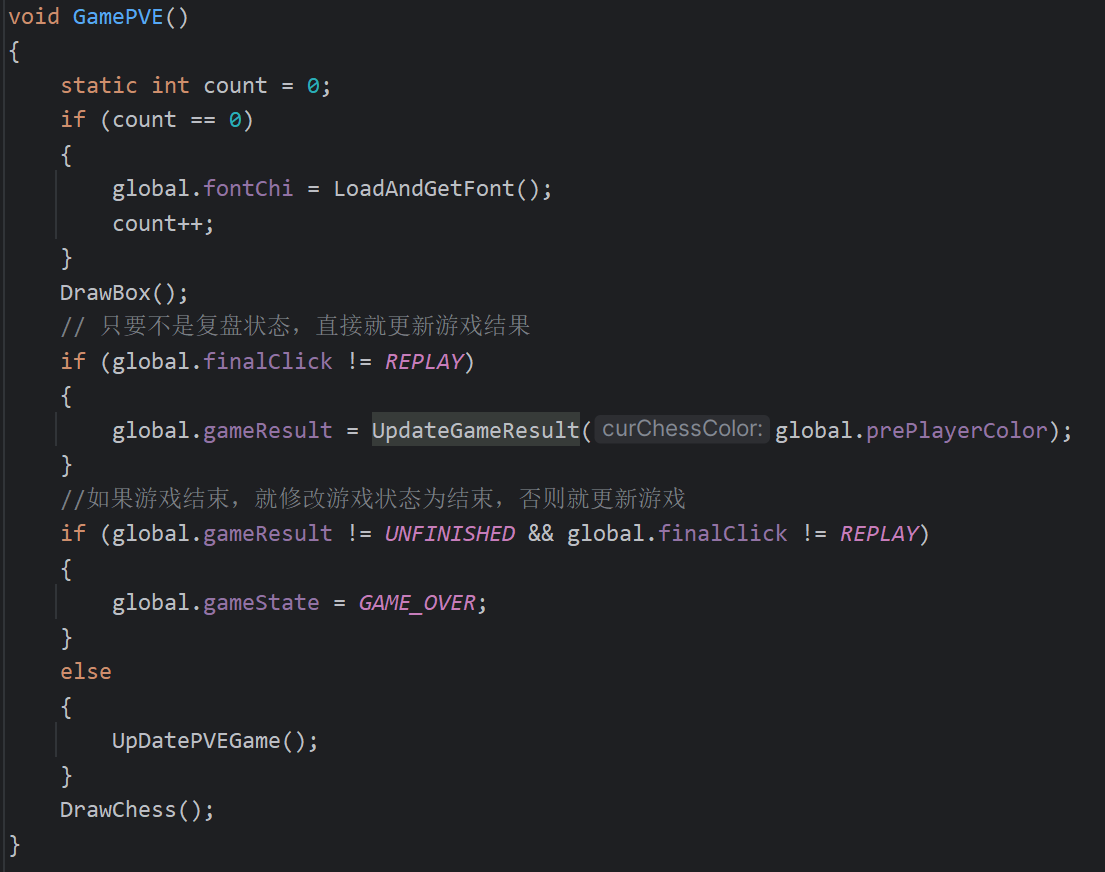


选择之后，再根据选择更新当前玩家上一步第一个玩家，以及玩家和AI的棋子及游戏状态



只要不是复盘状态就更新游戏结果，如果是复盘状态可以进行悔棋等操作。

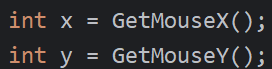
复盘状态：结束窗口没有操作直接右上角关闭

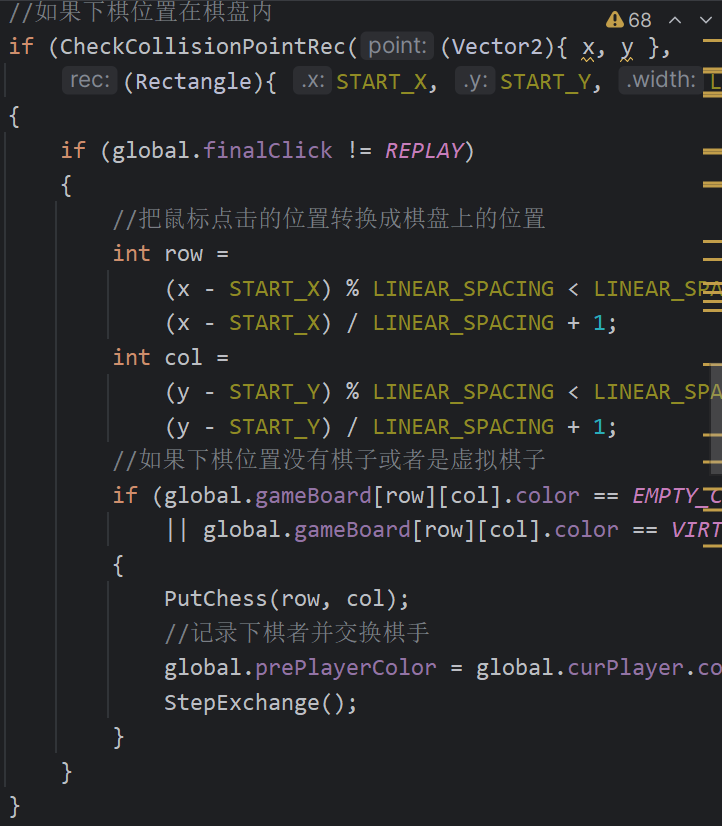


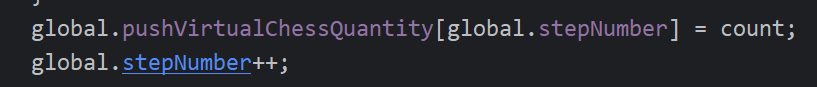
##### 判断游戏是否结束：

传入的是上一位下棋者的颜色，判断上一步有没有使游戏结束，



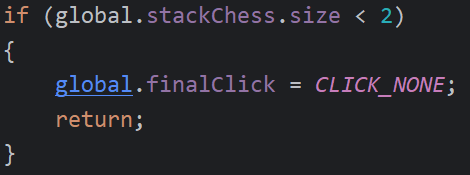
在绘制完棋盘后录到玩家下棋，获取点击位置如果点在棋盘内，只要棋盘位置没棋子则下棋，记录棋手并交换棋手。

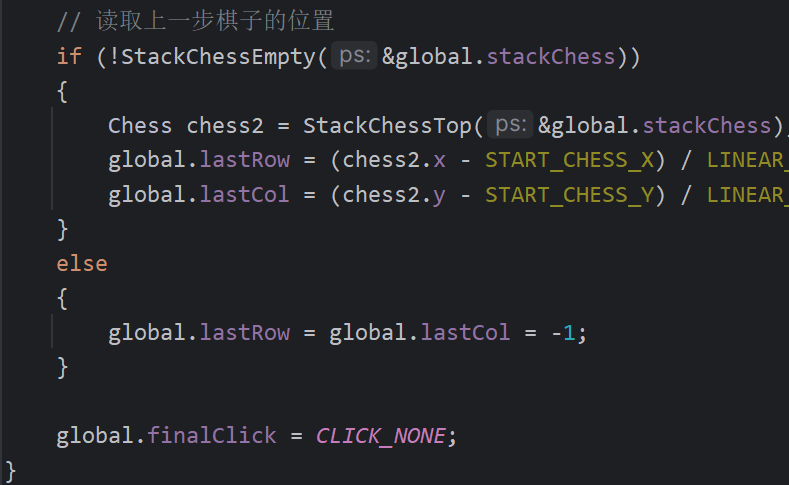
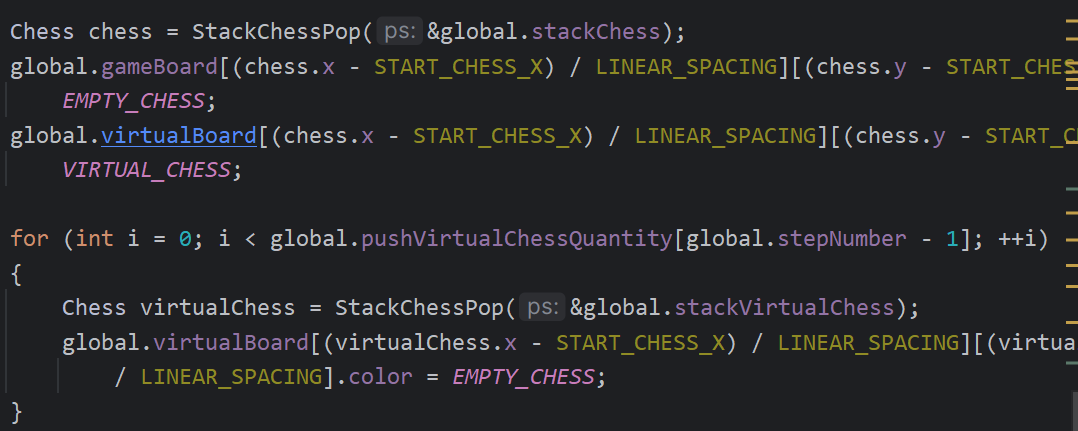


下棋：，把棋子放入棋盘，把该棋子信息放入栈中，并在新的棋盘数组中，把棋盘上该棋子周围两圈棋子标记为虚拟棋子（可能下棋的位置），减少递归点位，并将这些位置入栈，并用数组记录该步棋子延伸出的启发式点位个数，方便悔棋操作一处这些点位。并将当前下棋位置画出棋子并且高亮显示。

### 按钮实现：

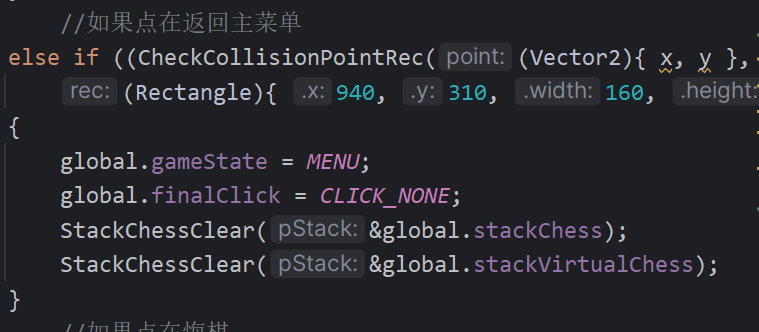
#### 悔棋：

先检查是否可以悔棋如果不能悔棋就直接返回，如果可以，就悔两步棋，人一步，AI一步，玩家棋盘上一步位置清空，另个棋盘清除之前的虚拟点位操作，因为是撤回两步就不需要交换棋手，并读取栈顶棋子作为最后一步高亮显示。



#### 返回主菜单：

如果点在返回主菜单更新游戏状态为menu，并清空栈，在MENU函数中会清空棋盘，



#### 重新开始：

直接初始化两个棋盘数组，当前玩家为先手玩家，清空栈，将最后一步棋初始化，



#### 认输（游戏结束）：

游戏状态为游戏结束，认输则为玩家输了，游戏结束则根据判断更新游戏结果

##### 

在主函数中接收游戏结束窗口的选择，主要目的：处理复盘

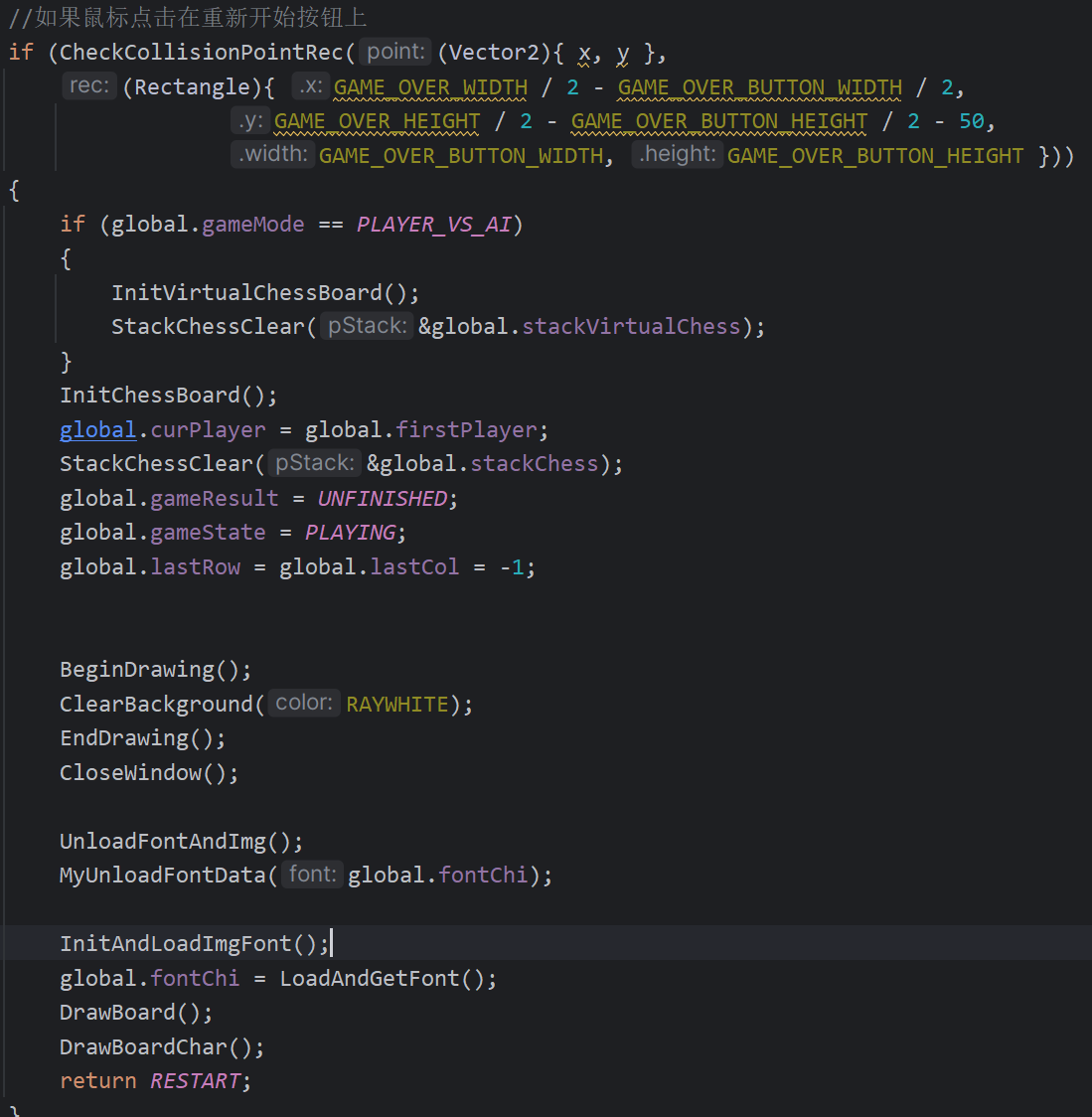


显示游戏结束窗口：



#### 游戏结束窗口操作：

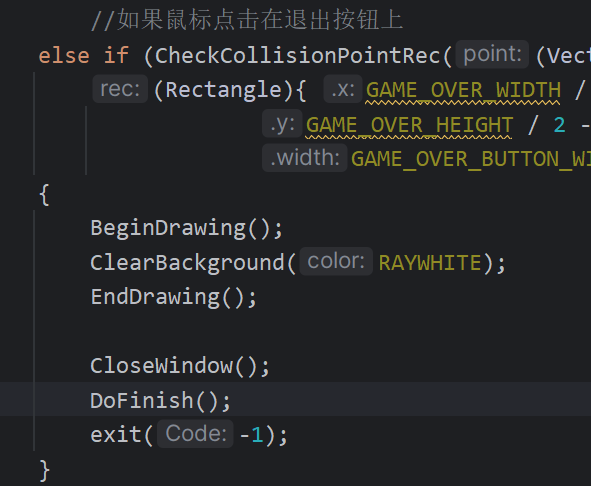
重新开始：（一系列初始化操作），更新游戏结果，关闭当前窗口并重建窗口。



#### 返回主菜单：更新游戏结果，关闭当前窗口并初始化为开始状态



#### 退出：（简单粗暴）



#### 复盘状态：

这个状态下可以悔棋，返回主菜单，重新开始，退出，悔棋之后可以重新下棋

### 双人对战模式：

简单粗暴，下棋交换棋手，按键功能同理



## AI算法：

赋值算法加剪枝算法加启发式搜索，接口：GetBestPoint()

如果是AI先手且第一次下棋，第一步坐标为天元。如果重新开始，就再次算第一步。如果不是第一步则进入极大值极小值搜索加剪枝



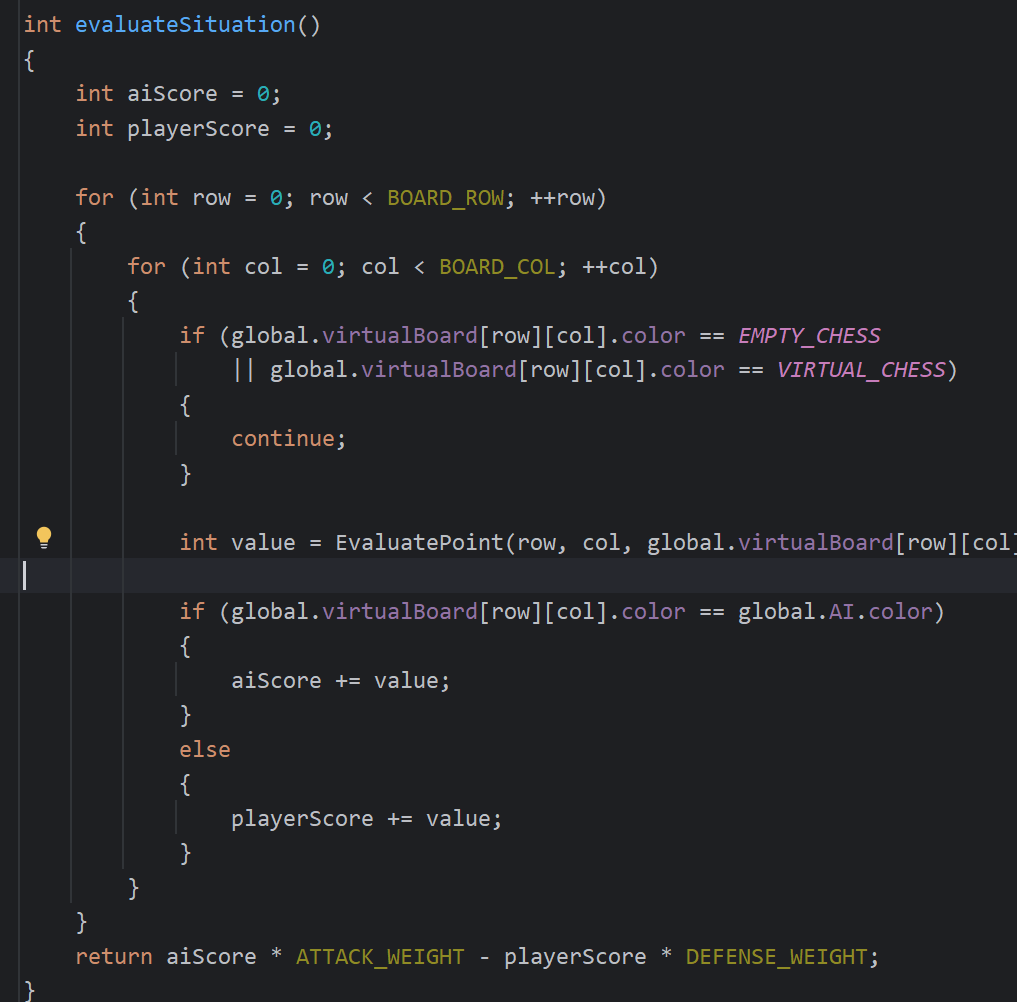
极大值极小值搜索：

如果在叶子节点则进行局势评估：

否则生成博弈树继续向下挖，每次depth--，如果判断这个位置下下去有连五直接返回最大值减一或者最小值加一，评估的位置是之前虚拟棋盘的虚拟点位



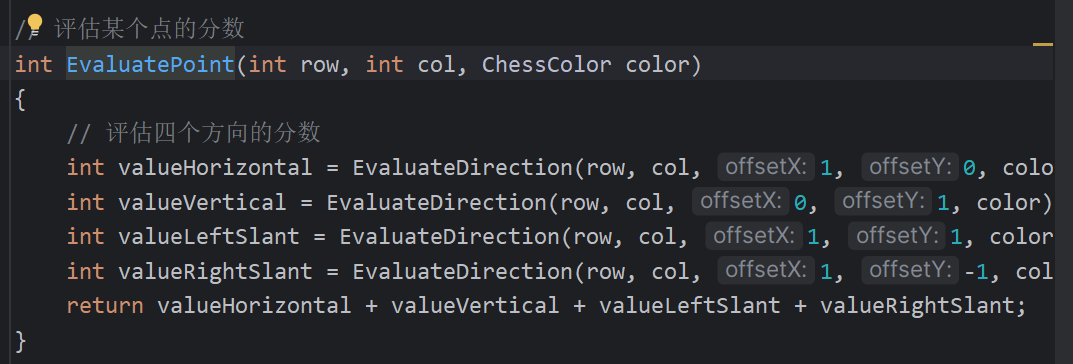
### 局势评估函数：



每个有棋子点位都进行评估，如果是AI的棋子加到AI的得分，如果是玩家的棋子加到玩家的得分，在乘以他们各自的系数，玩家得分取负数。

点位得分评估：

四个方向得分总和



单一方向评估：

先储存棋型，

读取棋型的序列

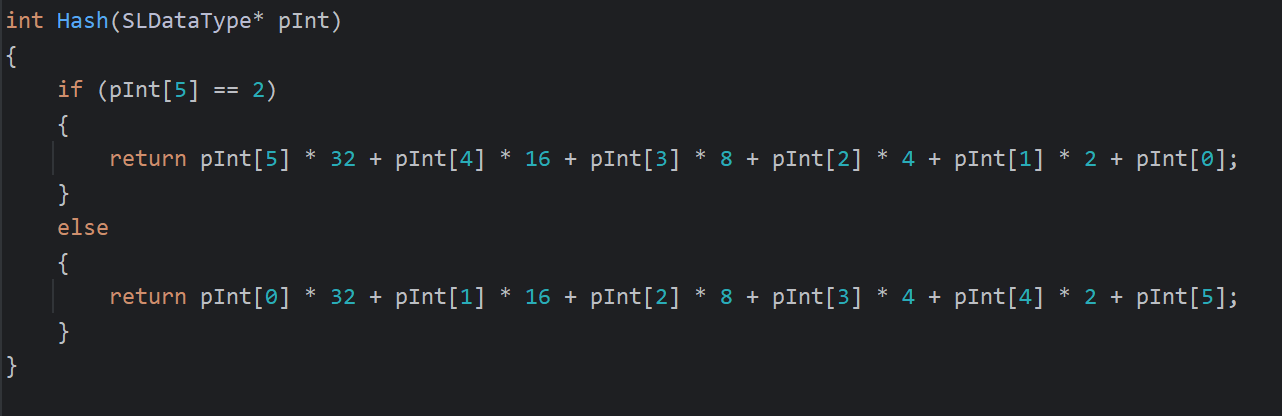
向某个方向遍历，如果当前棋子颜色与所下颜色相同则头插1存入顺序表，如果为空则头插0存入顺序表，

如果为敌方棋子或者超出边界则头插2或者遍历到第六个位置后从原本的位置向反方向遍历，

或者遍历到第六个位置后从原本的位置向反方向遍历，直到遇到敌方棋子或者边界



随后将棋型进行类二进制转换后获得key值



再把KEY值拿到hash表中比对，利用数组存入计算过的棋型，每一个顺序表都只有一种棋型，如果遇到了一样的大概率不是完全有效的棋型，此时计算当前棋型的1/3分，每次得分累加，最后返回该方向得分



检查是否有连五：



# 第三版遇到的问题：

1. 如何判断当前的棋子是谁的棋子：尝试直接判断两个结构体是否相等，发现无法实现，就干脆直接判断curPLayer.color和 AI.color是否一样。
2. 在重新开始和返回菜单都必须关闭再重建窗口，没办法直接只退出当前窗口，调用CloseWindow();时会导致两个窗口都直接关闭，除非像之前一样不使用弹出式窗口，直接多画点东西，最后还是决定使用弹出式窗口，多尝试点东西，避免划水
3. 按钮功能实现：
   1. 悔棋：在PVE悔棋时只悔棋一步会使AI再次进行计算并下棋，修改后直接悔棋AI和玩家一人一步，再由玩家下棋。
   2. 悔棋：在悔棋到最后一颗棋子在场时再次悔棋会直接崩溃，因为悔棋第二步棋子时会导致断言触发，修改后判断能否悔棋的条件从栈是否为空改为栈内元素是否大于1（ < 2直接返回形成空操作）。
   3. 为了减少游戏结束的判断时间，从检查双方棋子是否满足有连五改为检查刚才下棋的一方是否有连五，减少运算量
   4. 返回主菜单后或者重新开始后AI不认为自己是第一步，会把上一次下棋的位置读出来并下在那个位置，修改代码，当返回主菜单或者重新开始就修改global.finalClick 的状态，利用这个状态更新static 的 count的值为0即判断该次仍为第一次，并重置finalClick的状态
   5. 游戏结束：点右上角会默认两个窗口都关闭，并执行后续语句导致程序卡死，不断弹出结束窗口。修改：该操作默认成为“复盘状态”，在这个状态下不执行判断结束，直到棋盘状态被更新
4. AI算法实现：
   1. 估值算法：剪枝算法理解和写出来并不难，困难点在于估值算法，如何读取和评估棋型很关键，因为个人喜欢下五子棋，发现网上给的估值算法要不就太过于简单，让棋力过低，要不就是其他语言导致读不懂，便自己研究棋型，取一定量各自研究棋型的可能性，全部列出来，最多用到2^9个数据，但是后面发现数据量太大，而且重复项过多，最后借用一些想法让读取棋型以“边界”“敌方”“读了五个”为截止条件，再反向探索并按顺序插入顺序表，因为要单个点位，包括中间点位的读取，以及反向读取，单链表不够用，所以使用了顺序表。（双向链表或许可行 ，但是属于是遗漏的知识点），之后进行类二进制转化，获得key并与hash表作比对。
   2. 启发式搜索：因为递归深度加深后很明显用时大幅度增长便不得不用启发式搜索，价值优先，以连五，活四为第一档，4-4 4-3 3-3 为第二档，活三，冲四为三挡，其余为最后点位，会使剪枝剪得更快。
5. 头文件互相包含：因为没有进行头文件管理，在写代码时直接多次包含，本以为使用#pragma once就可以无视互相包含问题，但是发现不现实，于是重新进行了头文件管理，把函数声明和结构体定义，宏定义分开。
6. 全局变量过多：使用了过多的全局变量，最终将他们放入Global结构体中，增加可读性，避免未来发生冲突。
7. AI下棋放着四个不堵下活三
   1. 活三评分太高，降低评分。
   2. 活三被重复读取，使用记录防止重复读取
   3. 进攻系数过高，调整系数
8. 重构游戏结束判断，不再遍历棋盘来判断是否结束，只判断当前棋子造成的影响，优化了判断，改为下棋点前后找自己同色棋子并计数。

再优化：

问题：层数只支持两层，到第四层越下时间越久

原因：探寻的节点过多，需要减少低价值点位的探寻挖掘

解决：启发式搜索

优先搜索连五和活四

其次四三和三三

再次活三冲四

最后活二眠三

新产生问题：第一二步下的不是很好，总是下到左上角第二个

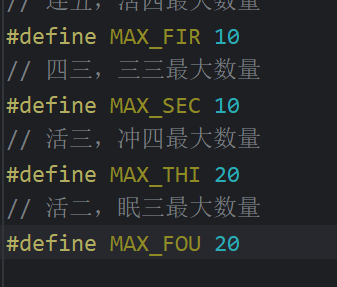
原因：前几步虚拟点位给太多了，但是节点少，会有漏查

解决方案：

1. 前几步走过去的判断逻辑，不用启发，（相当于增加节点数），最终放弃，因为6层时间太长
2. 减少前几步的虚拟点位数量

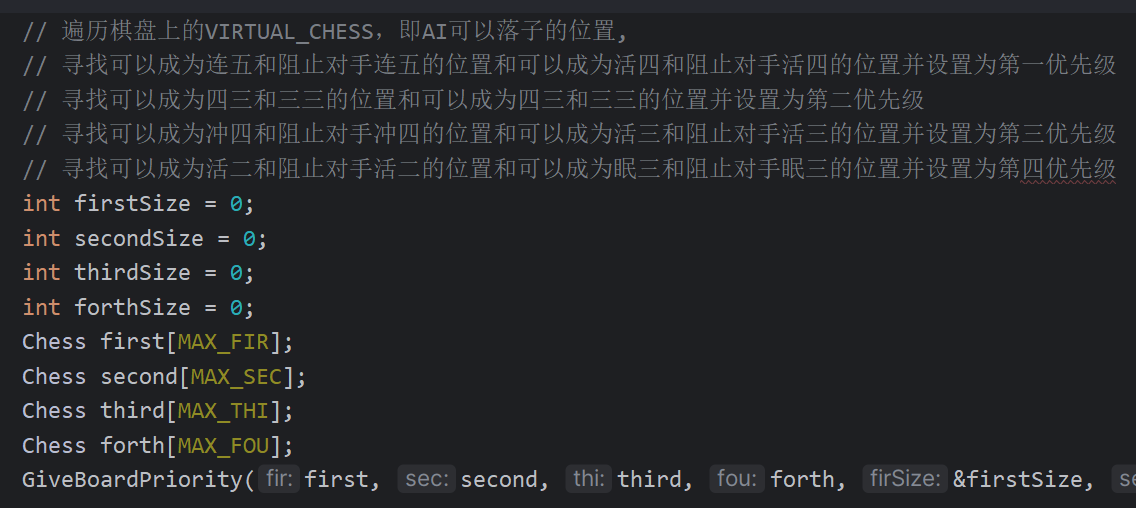
最终代码实现：

1.新增宏定义：节点数量



2.新增结构：判断在行列斜中的优先级

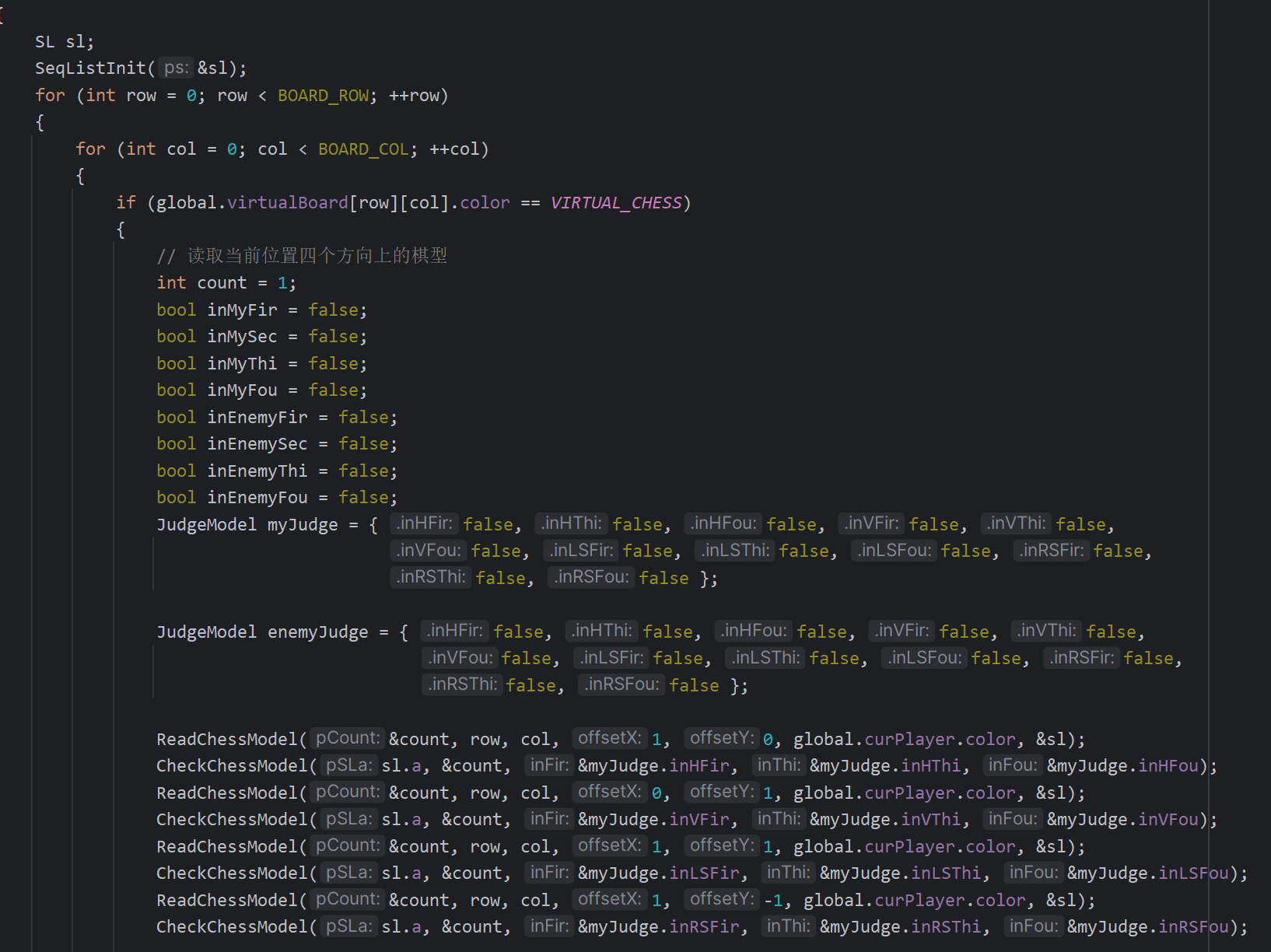


3.新增判断：整理价值优先

4.价值优先排序实现：

4.1读取该点位己方四个方向棋型数据

4.2检查该棋型四个方向棋型，并确定其可能所在的优先级，即是否有高优先级棋型，并更改其bool值





5.最终定级：

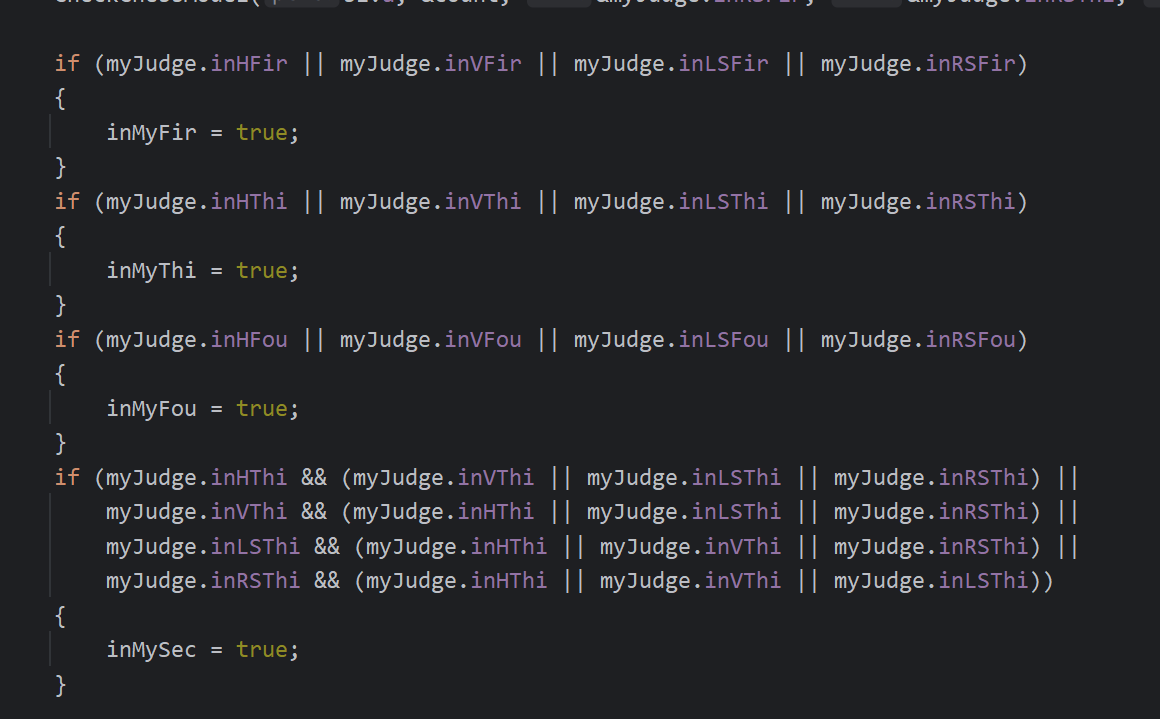
根据其可能的优先级确定真实优先级

第一优先级：任意方向连五，活四

第三优先级：任意方向冲四活三

第二优先级：有四四，四三，三三

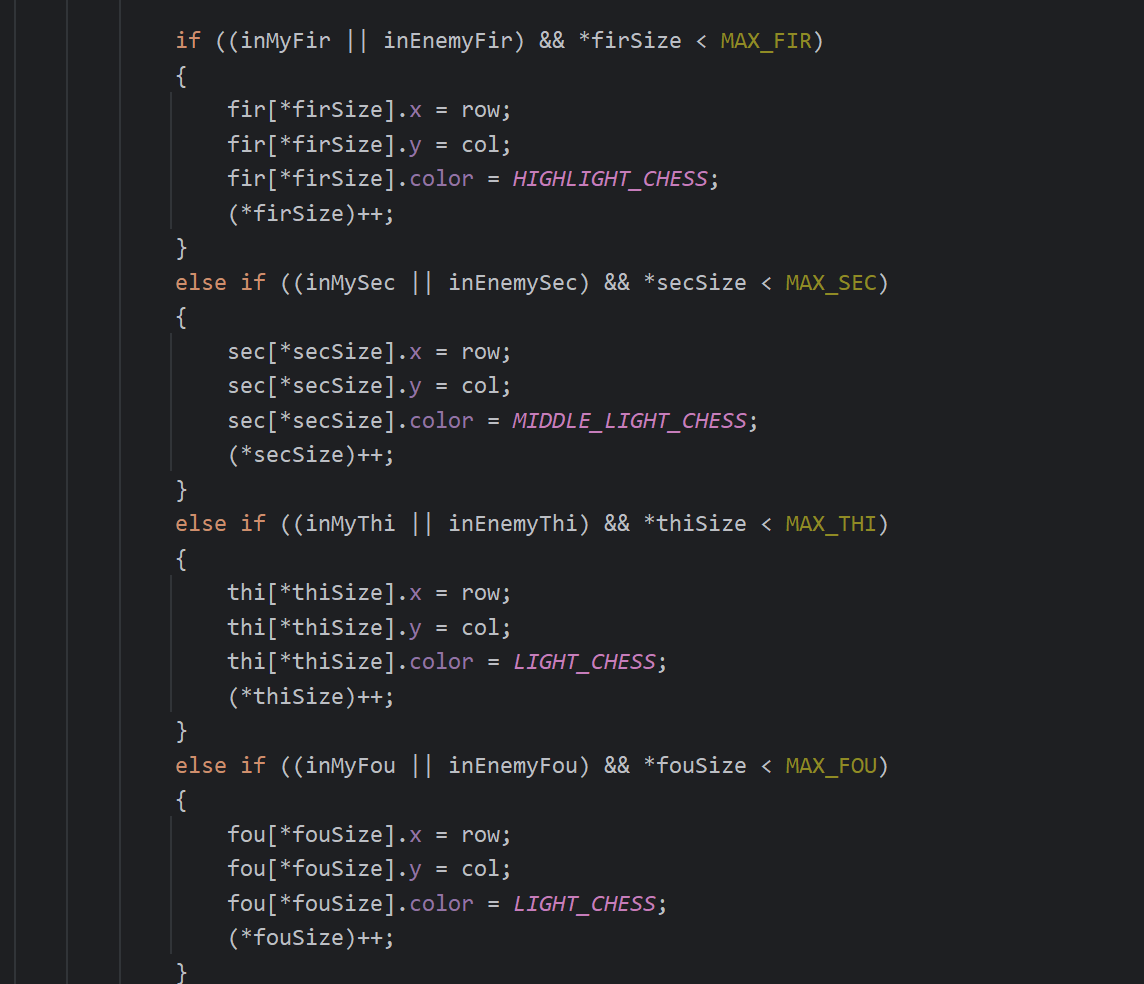
第四优先级：任意方向活二，眠三（可以优化多活二情况优先级，但不好实现）



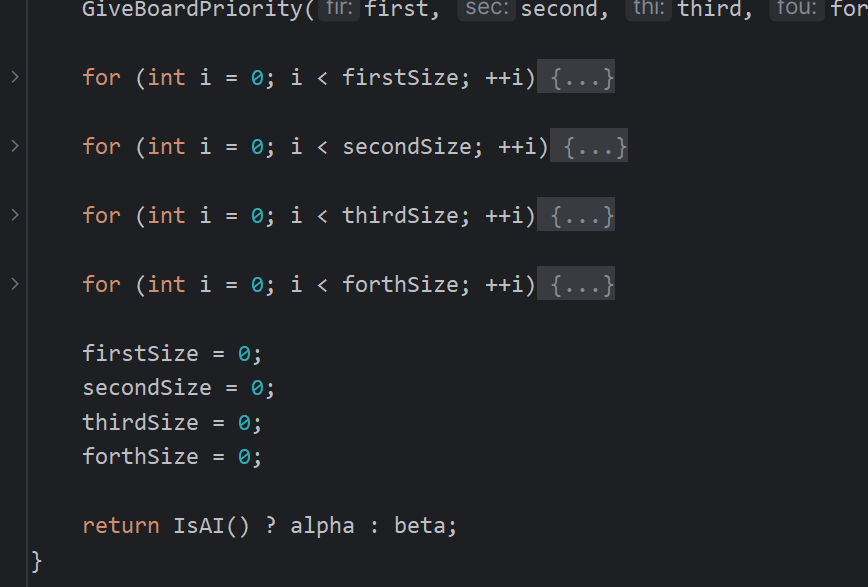
敌方位置判断同理

6.最终分类：

将对应优先级位置放入优先级数组中

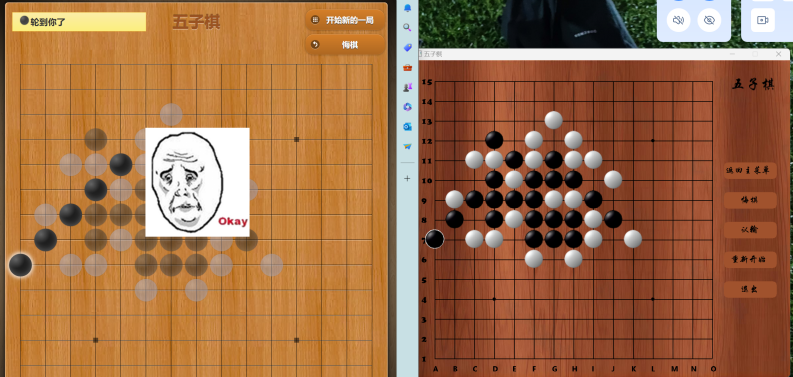


7.最后根据优先级进行评分和剪枝操作

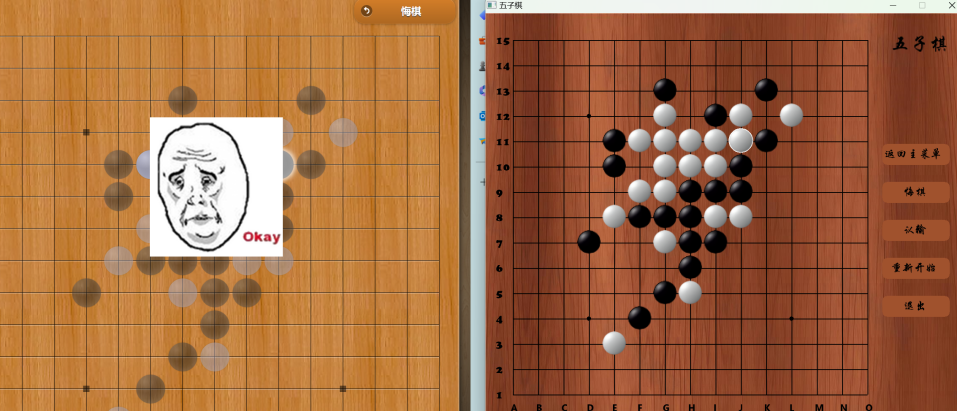


棋力测试：[AI在线五子棋——linfengfeiwu的小窝 (readgroup.cn)](https://blog.readgroup.cn/games/wuziqi2/)

（大师难度：先手胜）



（大师难度：后手胜）



[Gomoku Calculator (gomocalc.com)](https://gomocalc.com/" \l "/h8g9g6h6i7i9i11h9f9g10g7i8f11g11g8j9k9g12g13j7k6j10g4g5j8i10l10m11k8h10k10)

（同为6层）胜

