项目简介

五子棋游戏程序，使用自由无禁手规则，无时间限制，可连续悔棋，提供双人对战和人机对战两种模式，其中人机对战提供先后手选择。

项目实现

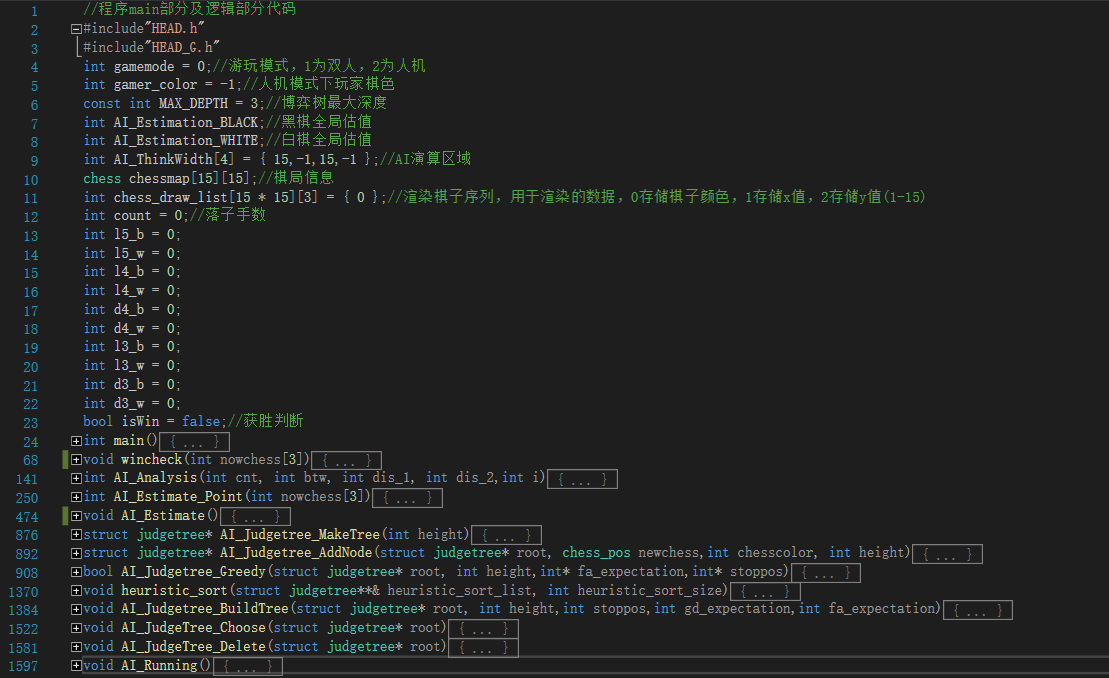
1. 开发环境

项目运行环境为windows（调用了<windows.h>库），项目编译环境为Visual Studio 2019，程序使用c++进行编写（但未涉及c++特性，程序整体保持c风格），基于GLFW库和GLEW库实现程序的图形部分。

1. 项目模块

程序本体由两个源代码文件：main.cpp、graphic.cpp，以及两个对应头文件：HEAD.h、HEAD\_G.h组成，其中main.cpp为项目入口文件，包括项目逻辑部分及项目通用部分，graphic.cpp包括项目的图形部分。

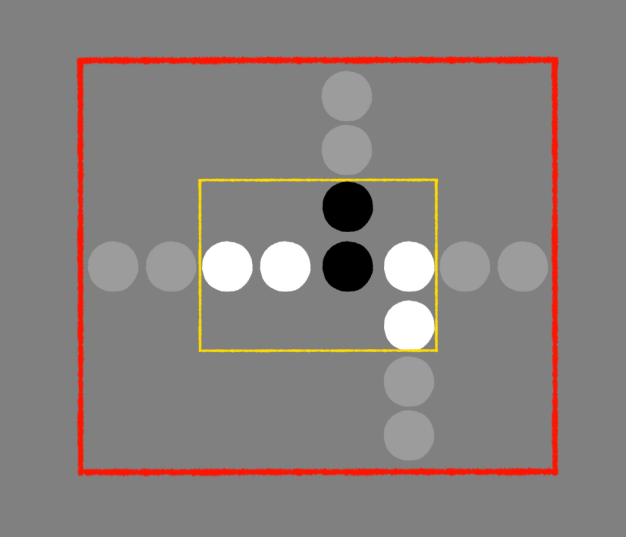
（1）.main.cpp

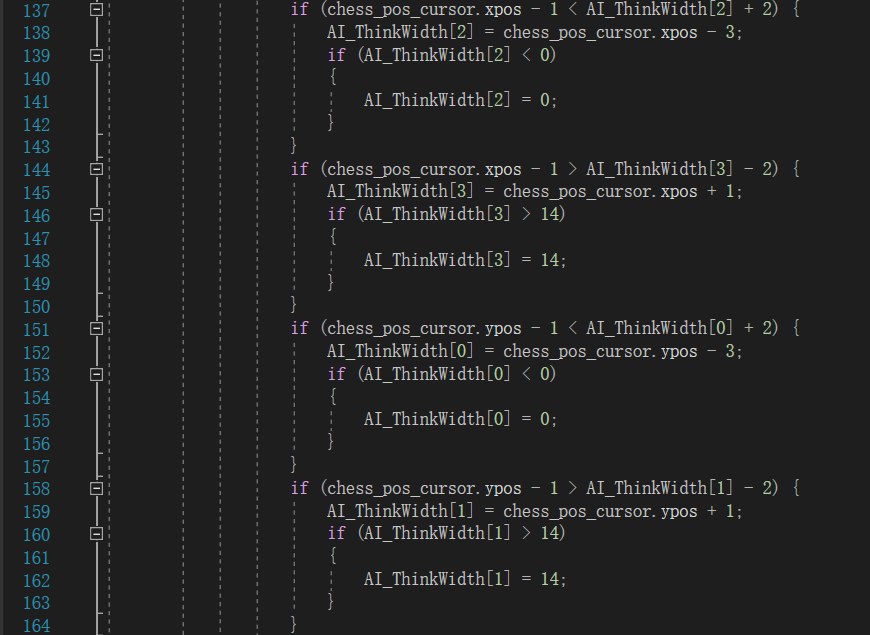


gamemode通过main函数设置。

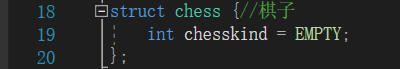
常量MAX\_DEPTH为min-max树默认最大深度，在仅使用CPU的情况下，遍历全部落子可能，想要AI正常运行就只能到达4~6层（深度从0计）。

黑、白全局估值用于在评估函数中独立记录双方评分。

AI\_ThinkWidth为AI演算区域，min-max树仅在区域范围内递归。区域在每次实际落子后进行更新，演算区域为以棋局最外围棋子外推两线（基于现有棋局能达到活五的最小范围）为边界的矩形区域。 



（更新区域代码）

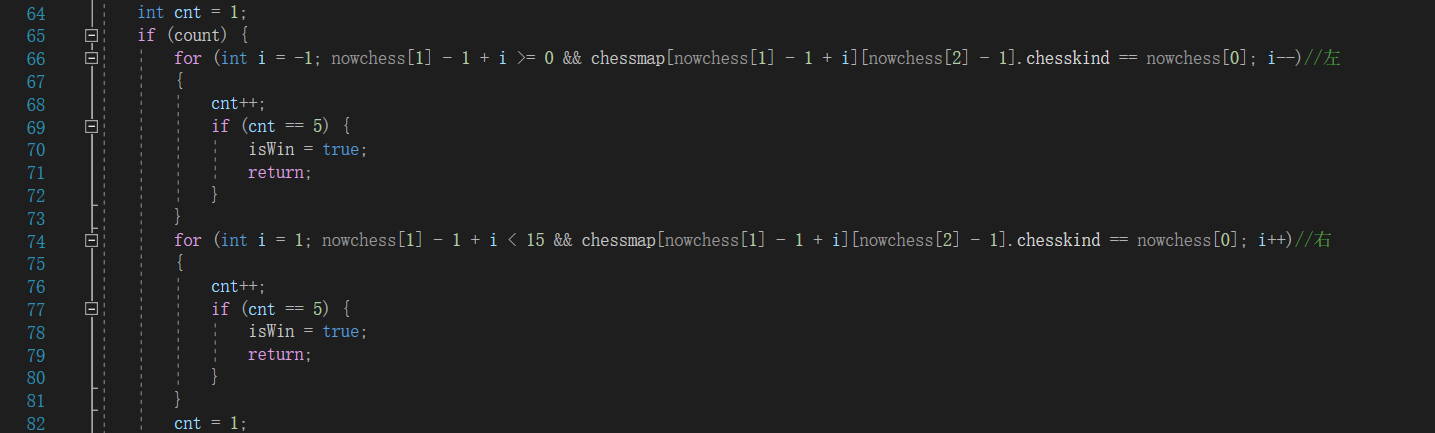
chess类型二维数组chessmap统计棋局所有落子点颜色信息

棋子渲染序列chess\_draw\_list负责按照落子顺序依次记录各棋子颜色、坐标数据。

项目中使用枚举常量记录颜色：

变量l5\_b至d3\_w用于在棋型分析函数中统计活五至眠三各种棋型的数量。

函数wincheck()在渲染循环中调用，在每次实际落子后判断游戏是否结束。函数以最后落子为中心向各方向统计连子数。

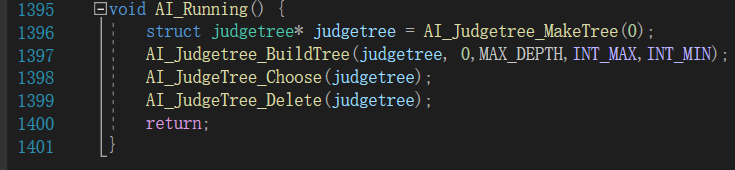


(以左右方向为例)

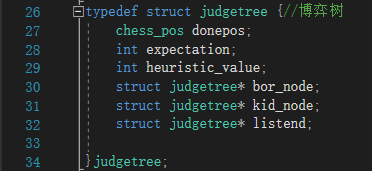
文件中函数wincheck()以下部分，为五子棋AI部分。

项目中的五子棋AI基于Min-Max树与Alpha-Beta剪枝实现，并结合了贪心和启发式的方法优化了搜索。

AI运行函数AI\_Running()为AI的接口函数，完成根据现有棋局选取最优落子点的任务。



Min-Max树使用子节点链表示法：



函数AI\_Judgetree\_BuildTree()

为AI核心函数。

参数中，stoppos为函数结束递归的层数，gd\_expectation为当前层结点的祖父结点（位于height-2层）的期望（简称祖期望），（如果按照剪枝算法将结点分为AI层、玩家层两类）即与height层结点最近的上级同类结点的期望值alpha或beta，fa\_expectation为当前层结点的父结点的期望（简称父期望）。

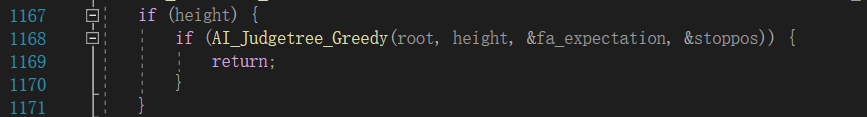
gd\_expectation取自height-1层的当前期望极值（即祖期望的备选值），fa\_expectation通过进入height层前，对其父节点调用评估函数（作为启发函数使用）得到。

进入函数的初始状态为：

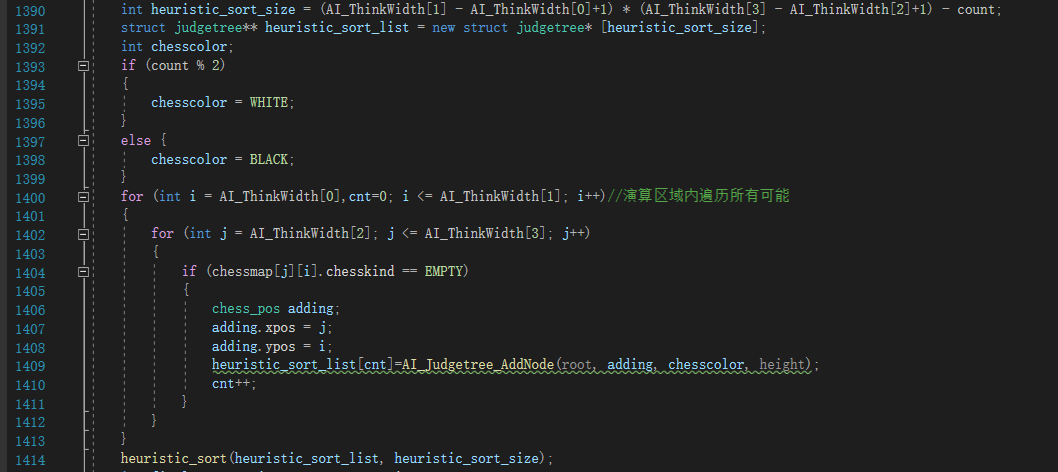


0层的祖期望设为最大值，使得0层作为AI层不可能被剪枝。

进入buildtree函数后，先调用AI\_Judgetree\_Greedy()函数（0层不调用），判断是否应该停止展开（已有某方必赢？）或展开至更深处，增大stoppos（当前所得评分值得怀疑？）。



如果通过了greedy函数，接着就将结点展开，准备考察所有可能性。



在不适用启发函数的条件下，之后的递归进入是按照深度优先的顺序进行，不利于剪枝算法发挥效能。于是在博弈树结点中添加有启发变量heuristic\_value，在部署展开结点时使用启发函数AI\_Estimate\_Point()计算出每个结点的启发值（这一函数实质为评估函数的单点版本，统计当前结点落下后能形成的棋型数量），启发函数内置于AI\_Judgetree\_AddNode()函数中，见后续例图。

在遍历完全后，根据启发值对诸结点进行排序（通过把结点指针存在数组中进行排序），高评分者之后会优先进入。

（就是简单冒个泡，本来想优先队列，搞不懂怎么自定义结构指针的比较，重载老报错。。。）

展开后将判断height是否达到stoppos。

当height!=stoppos，依次对展开得到的结点进行拟落子，递归进入深层。

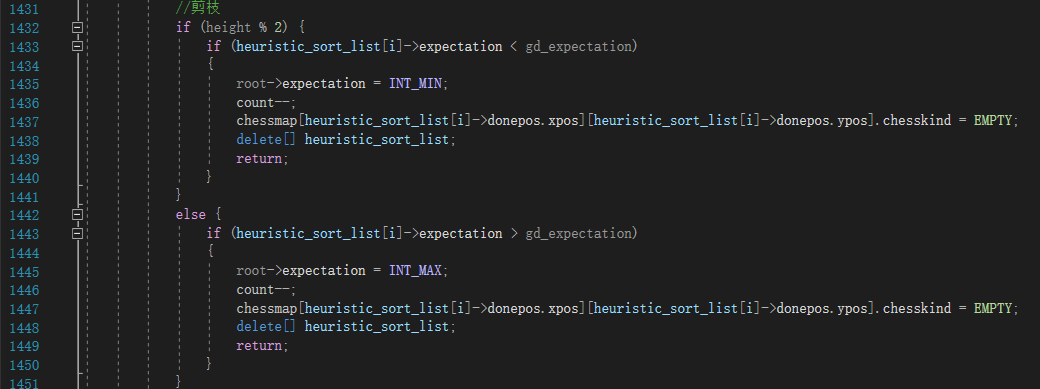
递归回退后，将其子结点期望极值（存储于kid\_node中）赋为自身期望。接着对已经完成搜索的树枝进行删除。



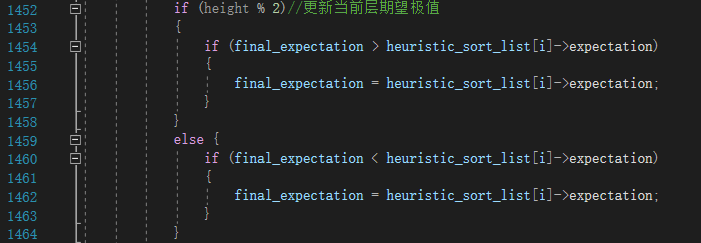
（由于不是按照顺序DFS搜索，不能直接将期望候选存于root结点中，此处另设变量final\_expectation暂存，在回退时赋给root）

之后根据所得期望，完成对剪枝的判断。玩家层（奇数层）的剪枝临界为下界，AI层（偶数层）的剪枝临界为上界。

当剪枝判定成立，则将期望极值设为（极大或极小）无穷，传出赋给其父结点，使其父结点在height-1层不可能被选为期望极值，即被淘汰。

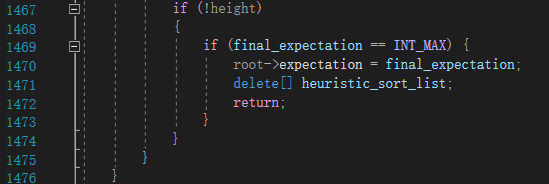


如果没有被剪枝，就正常完成对期望极值的筛选，期望极值存储于所在层的表头结点，即当前层的root，父结点的kid\_node。

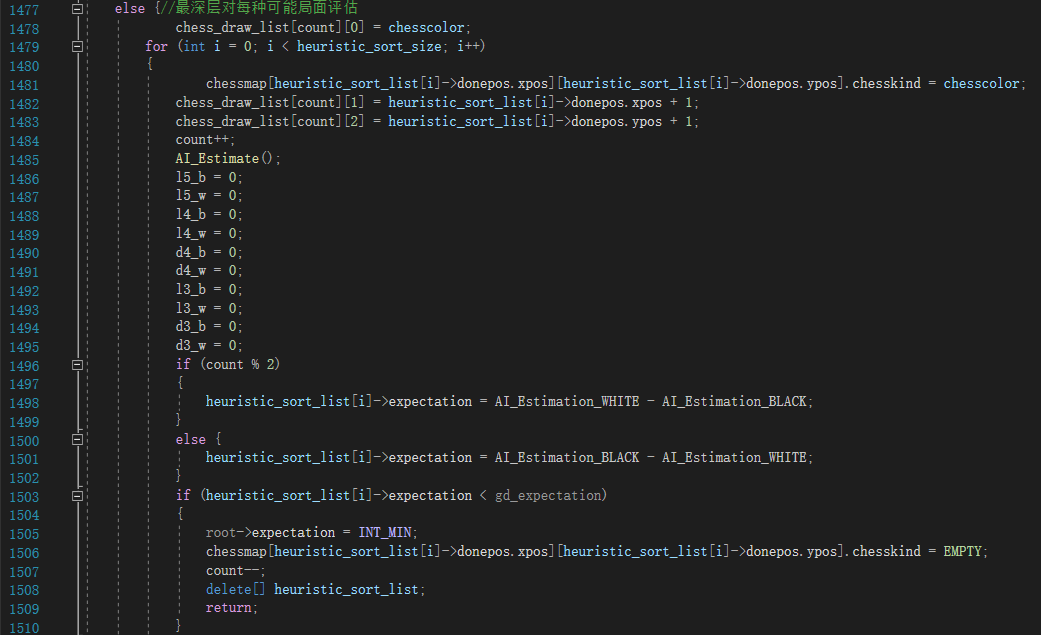


之后撤销拟落子，回退至上一层函数。

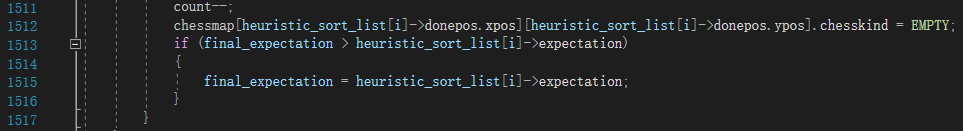
特别地，当height==0，在回退前会检查当前期望极值是否为极大上限，即是否找到了一步取胜落子点。



当height==stoppos,则当前层作为最终层，对展开得到的叶结点进行直接计算，得到相应期望。

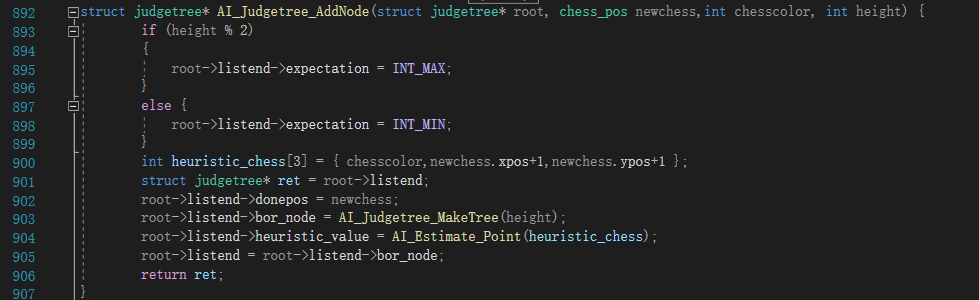


之后同样考察是否需要剪枝，选出其中极值，最终返回作为其父结点期望。



（图示为叶层，默认为玩家层，取极小值）

关于结点的初始化

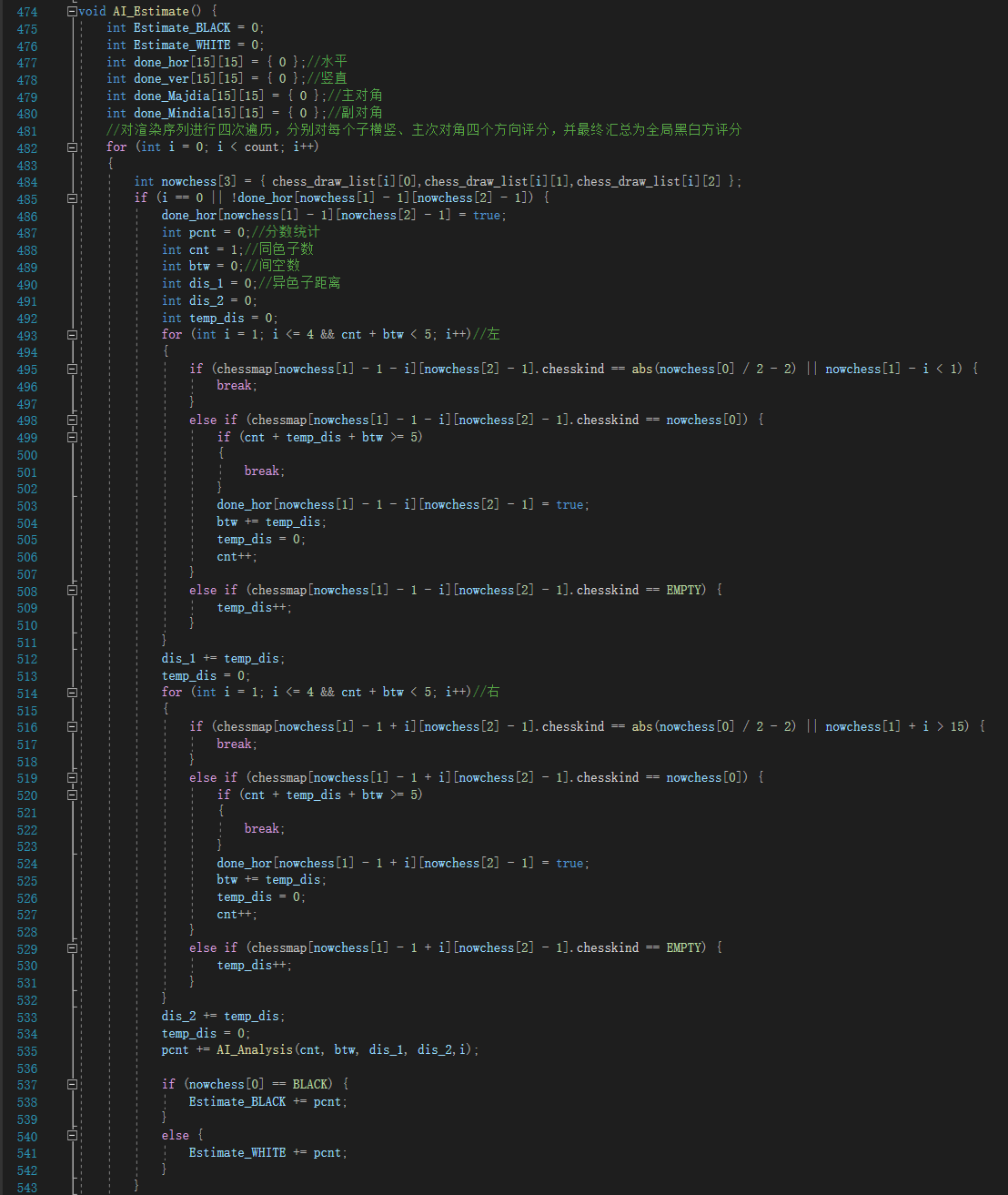


作为kid\_node，存储期望极值的结点root，其期望之后会作为下一层的gd\_expectation，故将结点root的期望初始化为（极大或极小）无穷，从而在这个期望被替换为正常实值前，不会发生剪枝，即在展开结点中至少会完成一个分支。其余结点的期望不会在被赋为正常实值前使用，但在此也以同样方式赋值。

指针ret存储每次新增的结点，传出赋予排序数组。

评估函数AI\_Estimate()，通过对棋子渲染序列遍历，得到每个棋型（棋局在检索中被分为多个可以形成活五的部分，每个部分中同色子数不超过五）的四个参数：同色子数cnt，间空数btw，两个方向的异色子距离dis\_1,dis\_2。将参数提供给分析棋型函数AI\_Analysis()从而得出当前棋子在各个方向所属的棋型。

（评估函数例图抛去外层循环，进行四个方向的评估，即为启发函数，在启发函数中，pcnt被连续使用统计总分，并最终作为返回值传出启发函数）



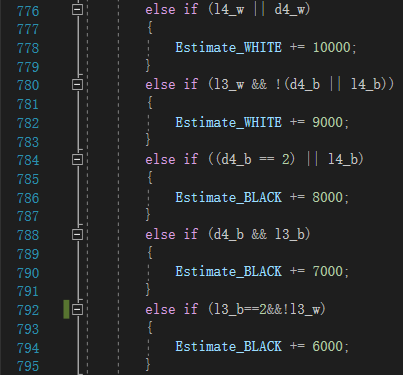
（以左右方向为例）

评估函数使用四个二维数组done分别从四个方向记录已经在遍历中涉及过的棋子，从而避免重复统计棋型，并减少循环次数。

分析棋型函数AI\_Analysis()在得到参数后，分析得出参数对应的棋型是什么（活五、活四、冲四、活三、眠三、活二、眠二），累加到对应棋型变量中，对可能无法实现必胜的棋型（冲四，活三，眠三，活二，眠二），分析函数会给评分加上适当的分数。

函数中有变量able=btw+cnt+dis\_1+dis\_2，表示当前棋型是否有成为活五的可能（当前方向是否已经被异色子堵死？）当able<5，则不予加分。

当评估函数完成遍历，得到全局双方的棋型统计与评分后，函数会评估当前局面是否为必胜局面，按照必胜局面的优先级，给评分加上必胜局面分数。必胜局面分数会远大于在棋型分析函数中所加的评分。

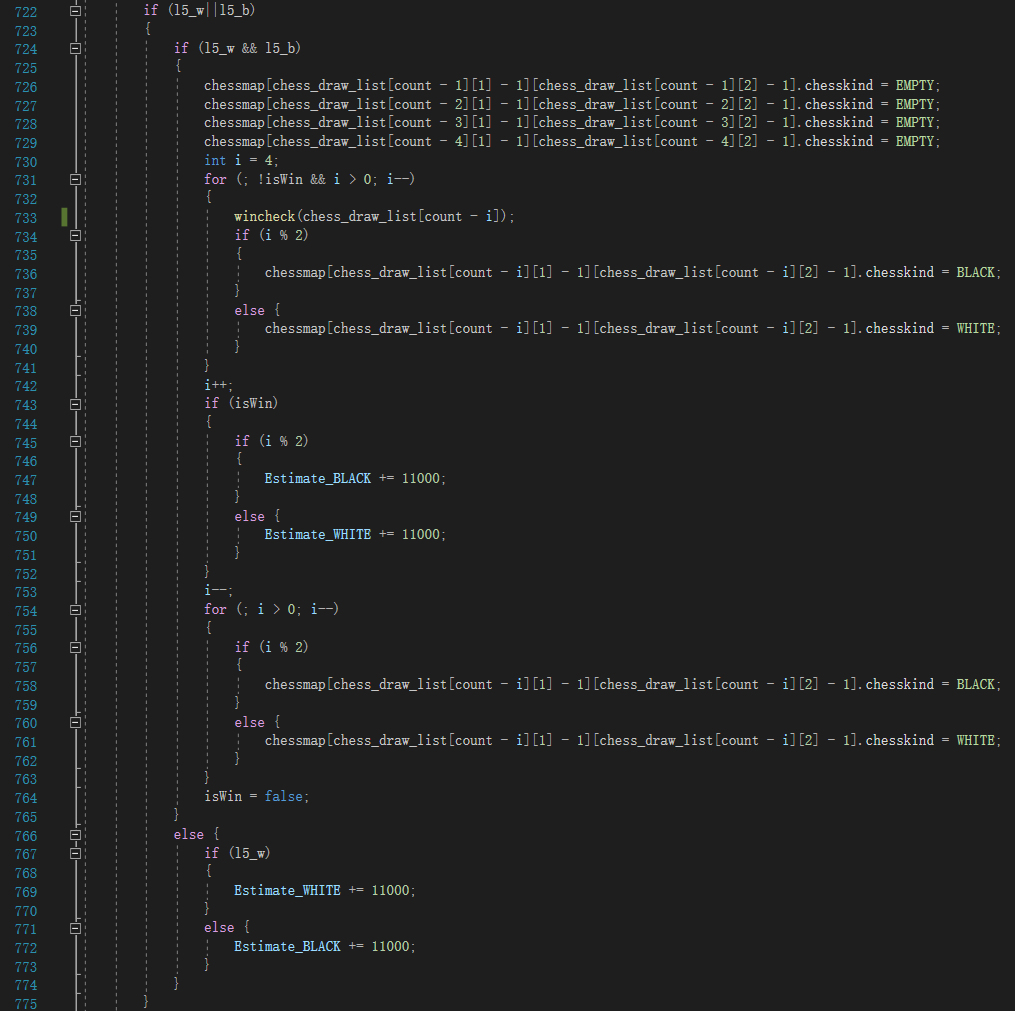


(例图是最后落子为黑子的情况)

注：就图中所示的情况，有白方有双三而黑方无活三的这种情况，存在黑方下一落子由眠三形成冲四的同时堵住了白方活三一端的情况。故准确的说仅仅黑方无活四并非必胜，但这一情况在实践中比较少见，且难于精确描述，故仅保留“无活四“这一部分。

”

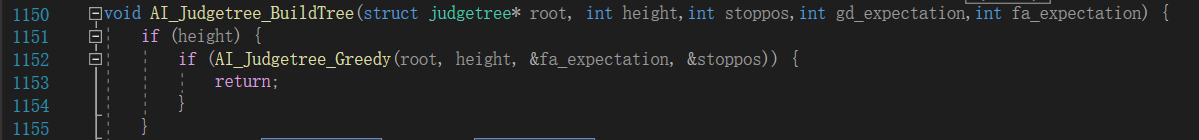
特别的，当局面中同时出现活五时（在加入AI\_Judgetree\_Greedy()函数后，这种情况基本不可能出现），会对落子进行回退，逐步重加，借助函数wincheck()分析出优先实现活五的一方。



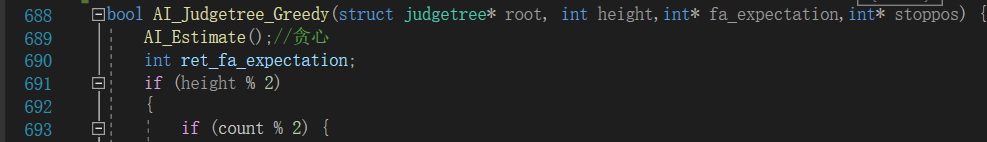
(例图是最后落子为黑子的情况)

在buildtree()函数中，当评估函数结束后，会对从函数得到的黑白双方分别得分进行运算，将局面的综合评分赋予结点，作为期望。

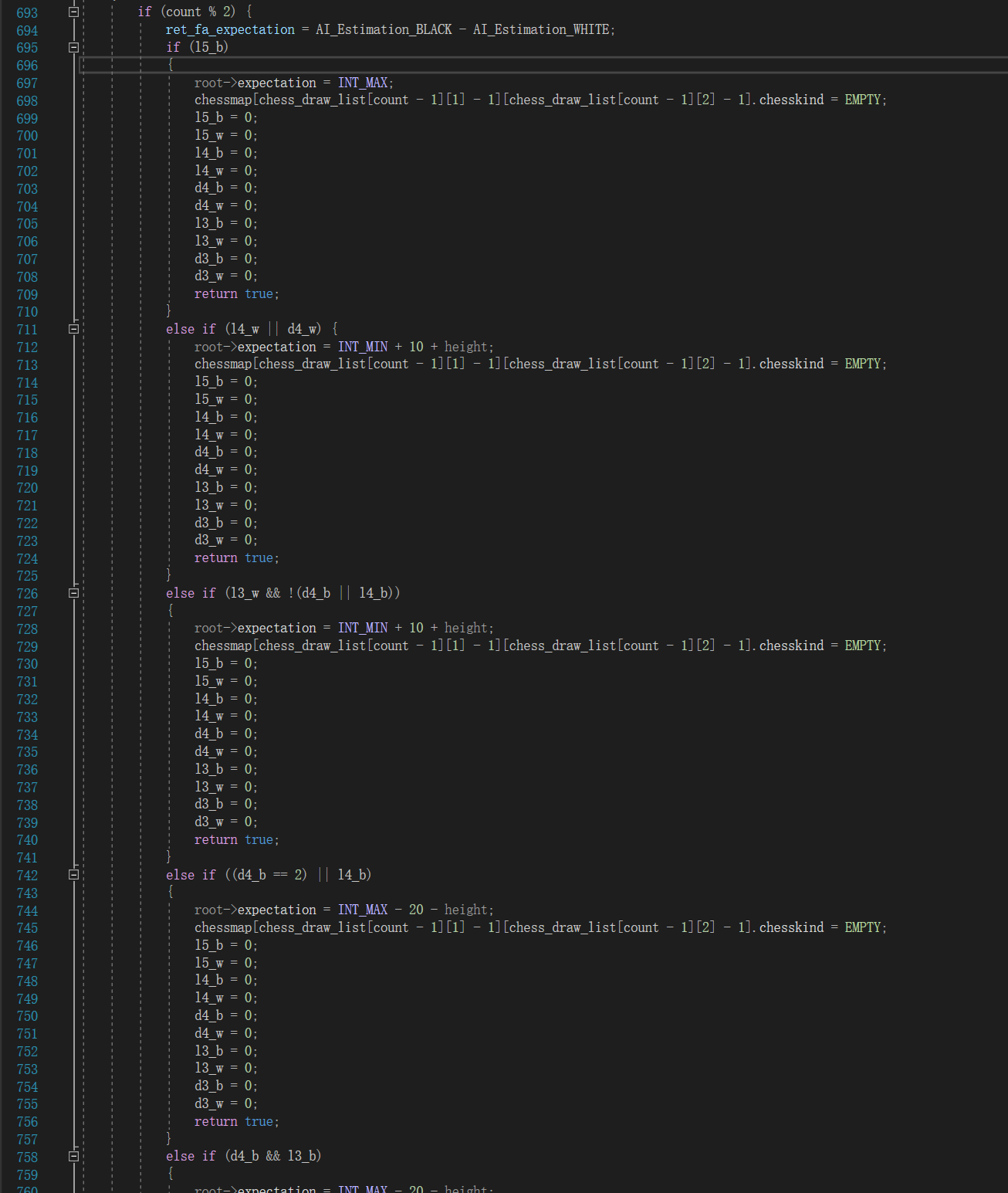
贪心函数AI\_Judgetree\_Greedy()通过在每一层开始时调用评估函数，判断是否需要中止深入，提前结束递归，或更加深入，增加递归次数，实际实现了优化，减少了不必要的搜索。



使用变量ret\_fa\_expectation记录下此次评估所得分数，为当前局面的期望，传入的参数fa\_expectation为当前局面的父期望。



通过两次判断得到当前的双方执子情况以及最新落子的黑白，判断当前局面是否为必胜局面。并根据必胜局面的优先级将（极大或极小）无穷作为期望极值返回，结束递归。

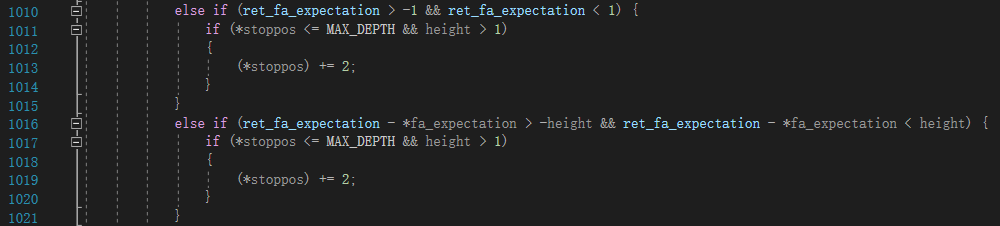


（例图为AI执黑，最新落子为黑子的情况）

期望计算中的0、10、20，分别表示已胜、一手必胜和两手必胜，必胜所需手数越多优先级越低，同时计算中还要加入height，当前height越深，优先级越低，以避免AI绕远路（第一手就能完成必胜，不应该浪费一手等到下一轮）或得出错误判断（浪费一手后反而使对手实现了必胜）。

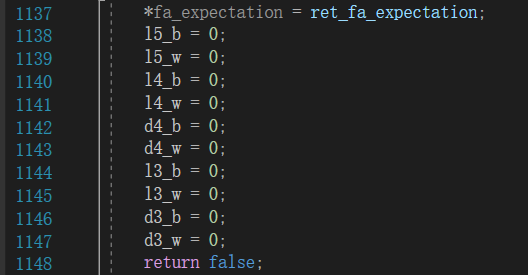
（为避免两个因素互相干扰，故将第一个因素提升一个量级）

当没有必胜局面时，还要分析最新落子对局面的改变如何，从而得到是否应该增加递归次数。



（例图仍为AI执黑，最新落子为黑子的情况）

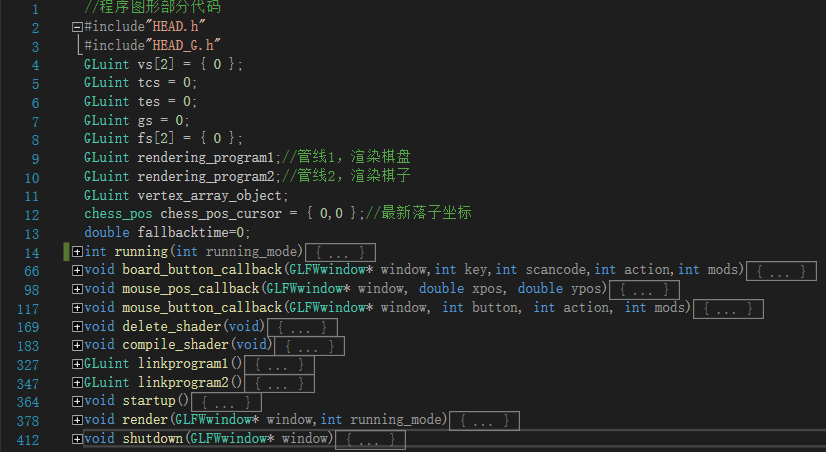
有两种被认为应该深入的情况，一种是黑白双方评分差距过小，另一种是新落子带来的变化过小，两种情况下局面都存在模糊不清，有待深入考察。同时递归次数最多只允许增加至MAX\_DEPTH+2。



之后会将ret\_fa\_expectation赋予fa\_expectation，作为当前结点展开得到的子节点的父期望，进入下一层递归。

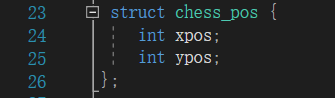
在buildtree函数结束后，Min-Max树会保留住0层，进入AI\_Judgetree\_choose()函数，根据0层的期望极大值，选出最佳落子点，并完成实际落子，同步更新AI演算区域。

在choose函数结束后，最终调用AI\_Judgetree\_Delete()函数，删除Min-Max树。至此AI\_Running函数结束。

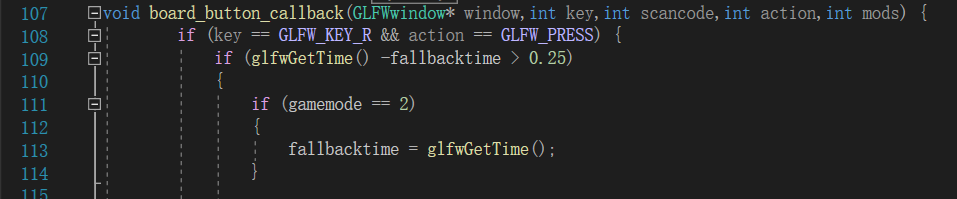
(2).graphic.cpp 

GLuint变量为着色器及渲染管线。项目在绘制画面时共使用了两条渲染管线，管线1负责绘制棋盘，管线2负责绘制棋子。

chess\_pos类型变量chess\_po\_cursor用于鼠标光标回调函数，实时记录距离光标最近的棋盘落子点坐标。

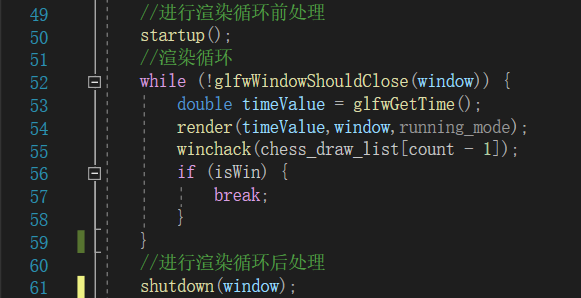


变量fallbacktime用于下面的键盘输入回调函数，为连续悔棋设置时间间隔，从而防止由于不正确的操作导致多次悔棋事件积攒在队列中一次性执行，使局面出现用户不希望的过度回退。



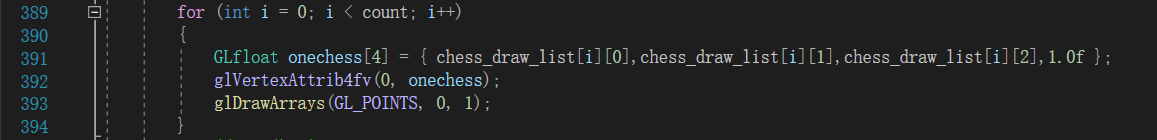
函数running()为游戏主体的接口函数，从函数main()进入，通过参数running\_mode决定游戏进行的模式。

函数running()包括两部分，首先对图形库进行调用并完成初始化，同时完成窗口的初始化与设置，接着是进入渲染循环开始游戏。渲染循环包括渲染前处理函数startup()、以函数rander()为核心的渲染循环、渲染后处理函数shutdown()三个部分。

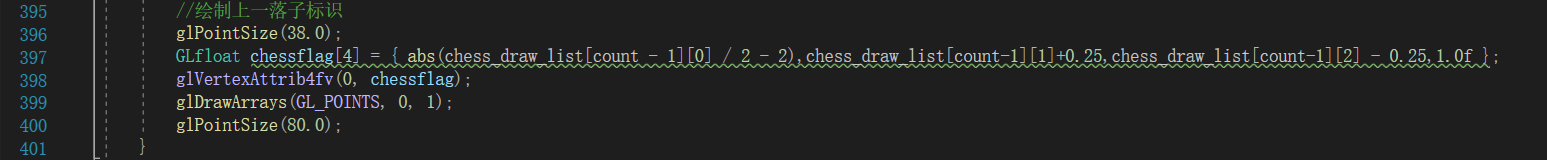


函数startup()完成着色器、管线的准备，渲染循环在每个循环负责在rander()中实现落子，接着判断游戏是否结束，函数shutdown()负责完成对结束状态棋局的绘制，输出胜负提示（使用windows句柄），并删除着色器、管线。

在rander()中，通过遍历棋子渲染序列，将坐标、颜色参数传入管线，以绘制单像素点作为棋子，依次绘制棋子。



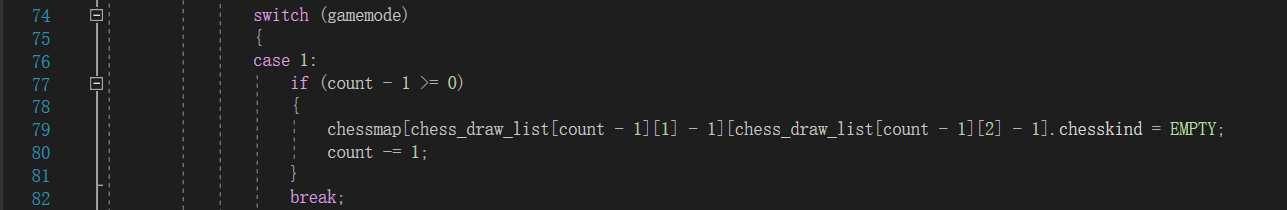
在绘制完全部棋子后，借用棋子渲染管线，在最后一个落子的右上角，取反色绘制上一落子标识。



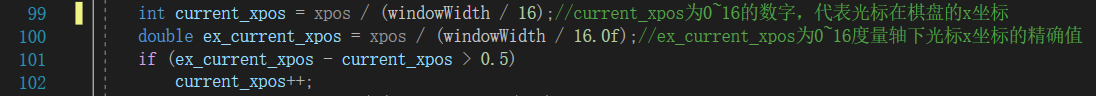
绘制结束后，函数接受输入的落子信息（处理输入事件），或进行AI评估（调用AI运行函数AI\_Running()），完成来自玩家或AI的实际落子。

三个callback函数，自上而下分别为键盘回调函数、鼠标光标回调函数、鼠标按键回调函数。

键盘回调函数设置为在按下键盘R键后对落子回数count做减值，并将对应回数在chessmap中的值设为EMPTY，实现在渲染与逻辑判断中屏蔽最新一手（或一轮）棋子，实现悔棋。（图为双人模式下的操作）



鼠标光标回调函数mouse\_pos\_callback()中，通过对光标精确x坐标向下取整得到current\_xpos,与光标精确x坐标ex\_current\_xpos对比，得到距离鼠标光标最近的棋盘落子点x坐标，y坐标同理。

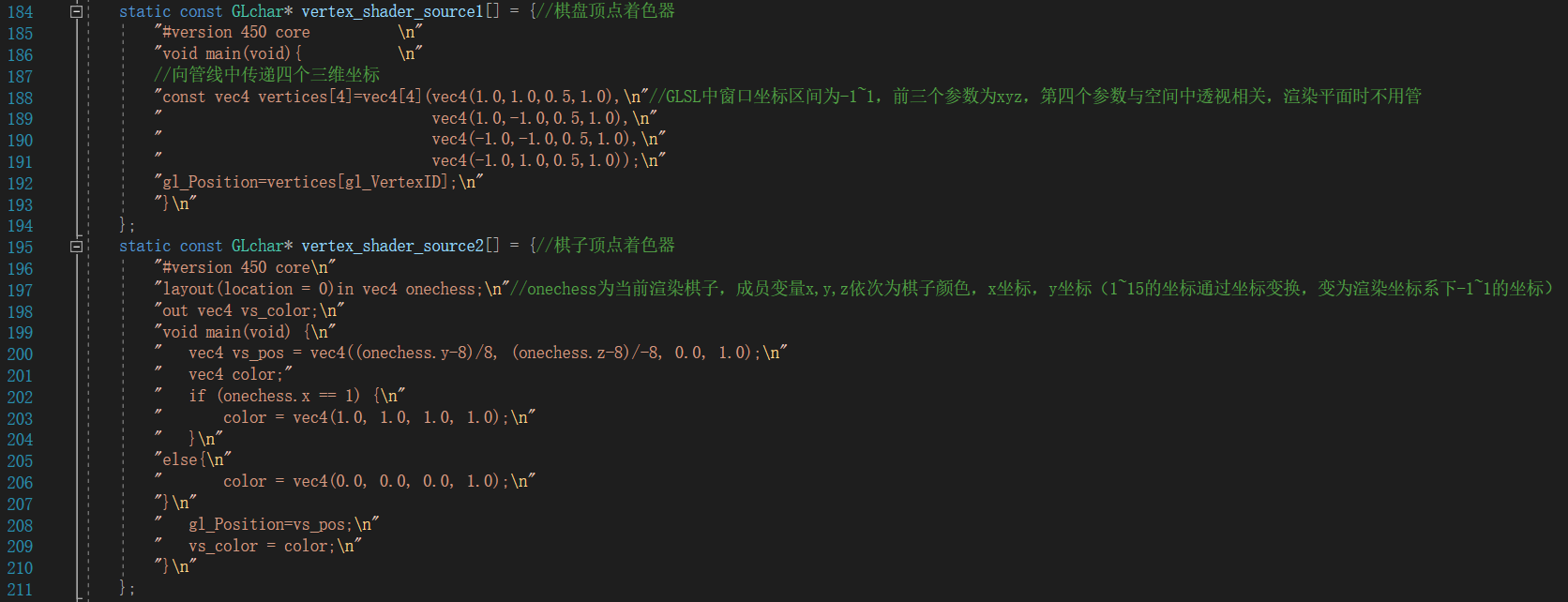


鼠标按键回调函数mouse\_button\_callback()在按键后，通过光标回调函数获取落子点坐标后，完成落子。同时，落子后会对AI演算区域进行更新。

函数delete\_shader()在shutdown()中调用，执行具体删除着色器的任务。

函数compile\_shader()中包括了绘制棋盘棋子的GLSL代码：

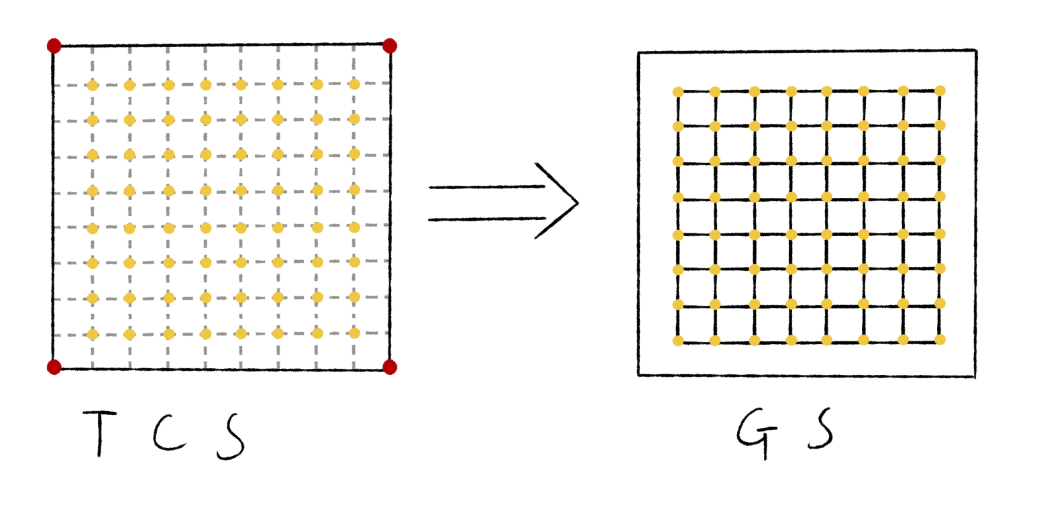
顶点着色器部分：



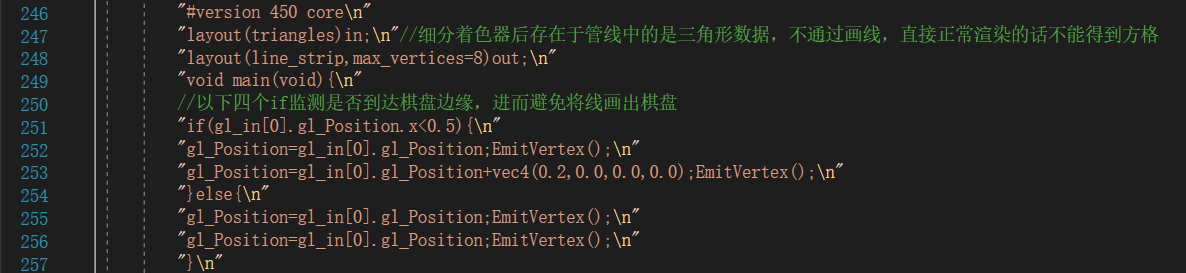
棋盘顶点着色器取窗口四角顶点，棋子顶点着色器将传入的棋子坐标转化为GLSL坐标系下对应坐标



棋盘曲面细分着色器（TCS、TES），基于四个顶点将窗口划分为16\*16的方格，并取得内部交点传入几何着色器。

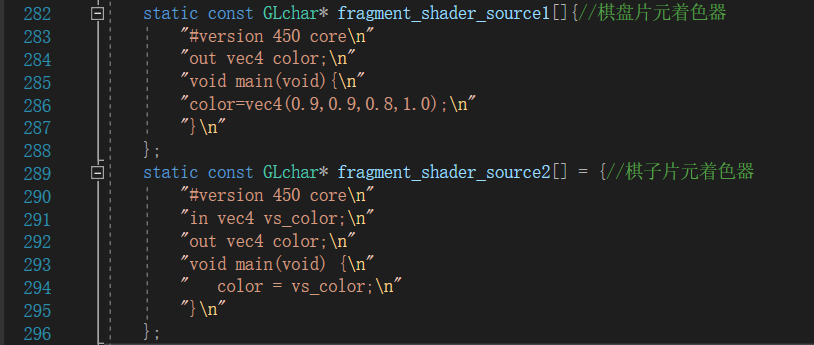


棋盘几何着色器（GS），根据细分着色器得到的交点数据，以每一个交点为起点向四个方向画线，从而得到棋盘，在绘制时识别边缘点，防止线段画出棋盘。



（以右侧为例）

片元着色器设置固定棋盘线框颜色，根据棋子颜色数据绘制棋子颜色。

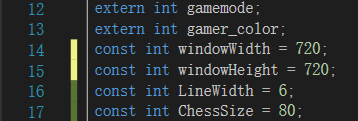


函数 linkprogram1()、linkprogram2()分别负责链接棋盘管线与棋子管线。

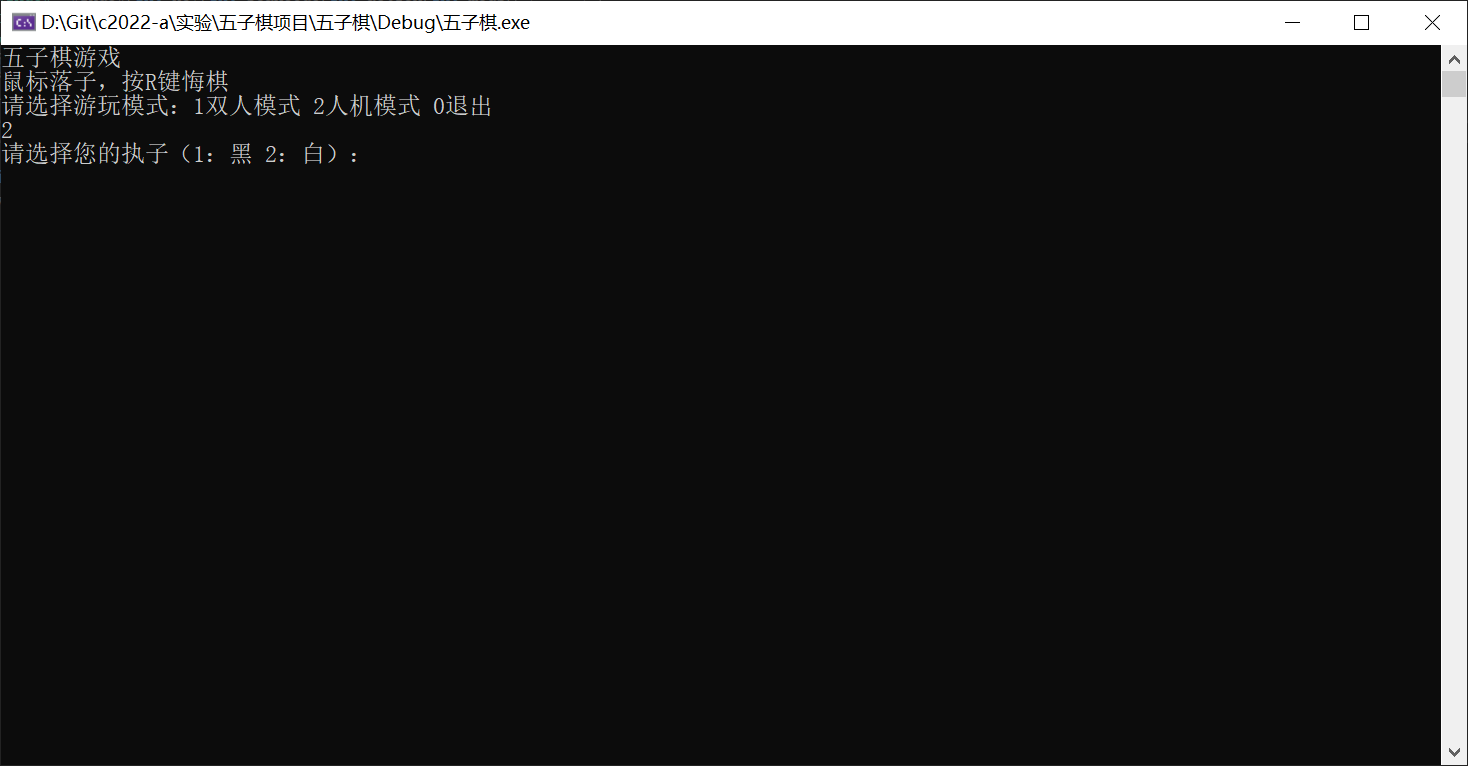
1. 实际运行

运行须知：由于程序的点、线图形的粗细大小均用常数定义，当显示器的大小过大或过小时会导致窗口出现错误。需根据实际情况进行调节。

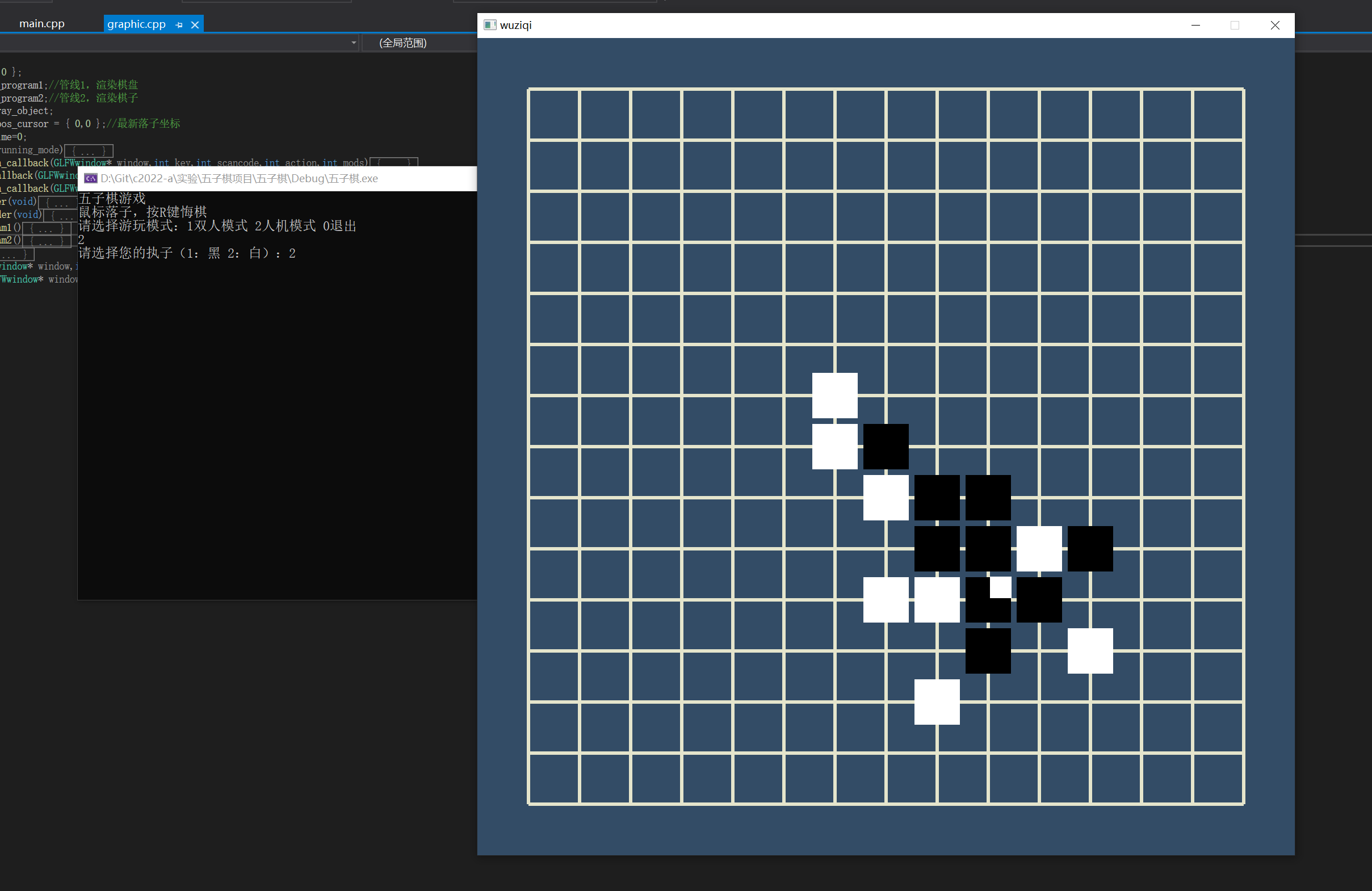
HEAD.h中的常数，调节窗口长宽、棋盘线宽、棋子大小：



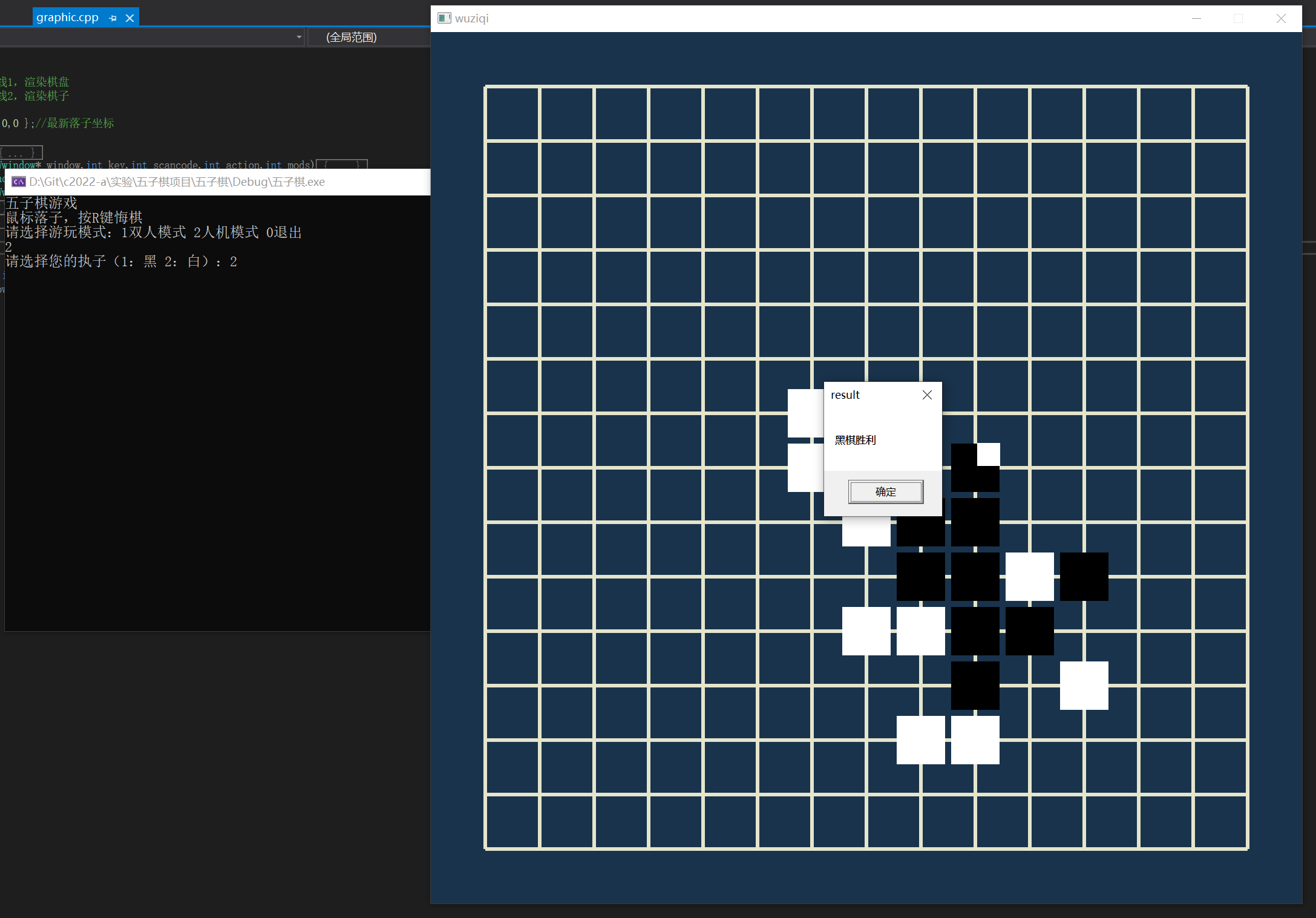
上述常数需保持一定比例，更改时需要同步变化。



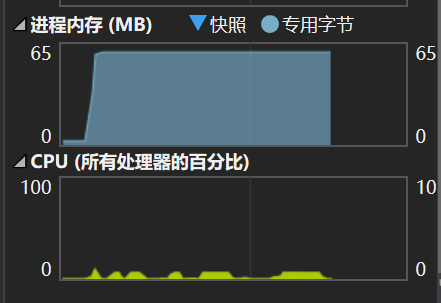
(程序进入界面)



（程序实机运行界面）



（获胜提示）

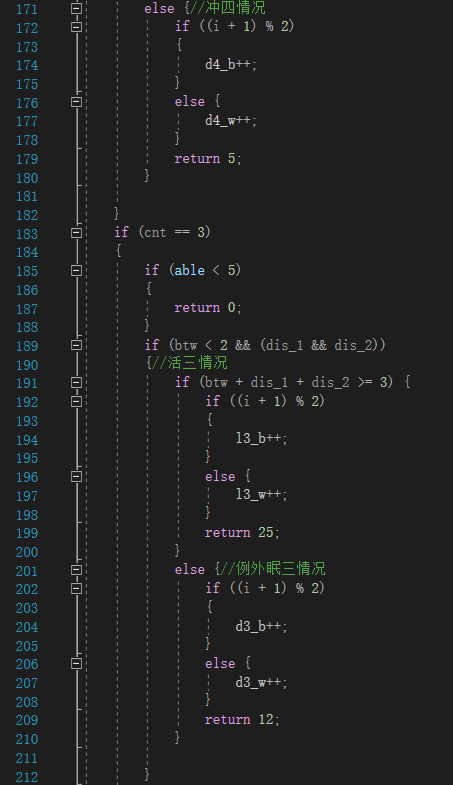
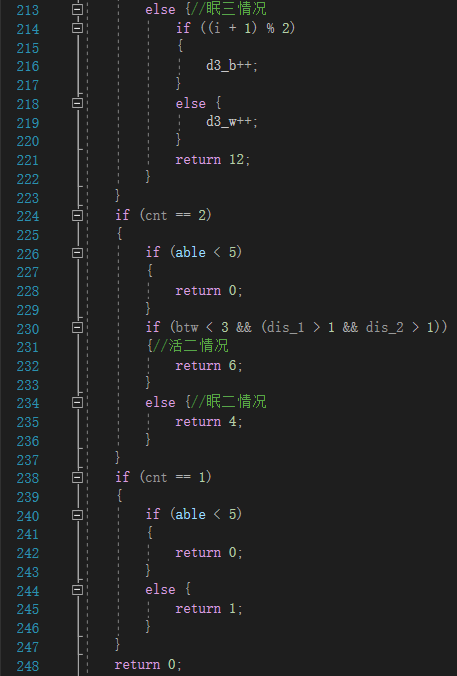


（运行状态）

抛去VS组件，程序本体约55mb，没有掉帧卡顿问题（如果AI的长考不算卡顿的话）。在动态管理内存的条件下，如图所示，进程内存可维持在60mb左右。

AI的智能水平

本项目中的AI博弈树基本只到达4层，能够预想两个回合后的局面，而五子棋中基本的必胜局面同样最多为二手必胜（四三，三三），在将必胜局面作为公式记入AI中后，可以认为其能够预想到三个回合后的必胜，大致属于仅仅明白如何获胜的水平。在添加贪心、启发要素后，AI能做到越直观的局面越快速做出反应。AI评估函数在非必胜情况下进行的评分是基于各种棋型数量给出的，太过粗放，可能比较难以预测出棋局的走势。

在某专业五子棋平台软件里，AI在执黑的情况下可以胜过初级AI（初级往下还有新手和入门级），而使用新号在平台中进行人机对战时，除去长考超时的情况，经常能够胜利。这表明项目中的AI在面对没有研究过五子棋的新人的时候是很有一战之力的，但面对五子棋爱好者就显然难以取胜。

当然，现阶段项目中的AI在关闭Greedy函数中增加递归次数的代码后，是可以使用6层博弈树运行的，这样AI的棋力又能有所提升，但在游戏进行到后期时仍会因运算时间过长影响游戏进程。

1. 参考文献

openGL学习资料：OpenGL超级宝典（第7版）笔记13 前三章实例 下个五子棋 （上）

https://blog.csdn.net/qq\_44016525/article/details/114532039

（图形界面的写法都是从这个系列博文里学的。五子棋图形界面中棋盘的细分shader代码是原样搬过来的，因为没弄懂，棋子的顶点、片元shader倒是学完自己写的）

棋型理论来源：Python五子棋AI实现(2)：棋型评估函数实现

https://blog.csdn.net/marble\_xu/article/details/90450436

贪心、启发思路来源：五子棋AI算法第四篇-启发式搜索函数

https://blog.csdn.net/lihongxun945/article/details/50668622