# 基于博弈树的五子棋设计

#### 一、游戏简介

五子棋是一种两人对弈的纯策略型棋类游戏，双方各执黑白两种棋子，下在棋盘直线与横线的交叉点上，先形成五子连珠者获胜。

#### 二、游戏设计

在基于raylib形成的可视化界面上，通过鼠标点击，以后手执白子与电脑进行对战。

#### 程序设计

程序采用博弈树算法和简单的alpha-beta剪枝。

极大极小树（Min-Max Tree）是一种将每一层划分为极大层和极小层的树。在极大层节点处（轮到我做决策）我们希望最大化收益，而在极小层节点处（轮到对方做决策）我们希望最小化收益（也就是对方最大化收益）。在五子棋中，最大层和最小层是交替出现的。而为了刻画收益，需要给每个节点赋值，这便需要估价函数。

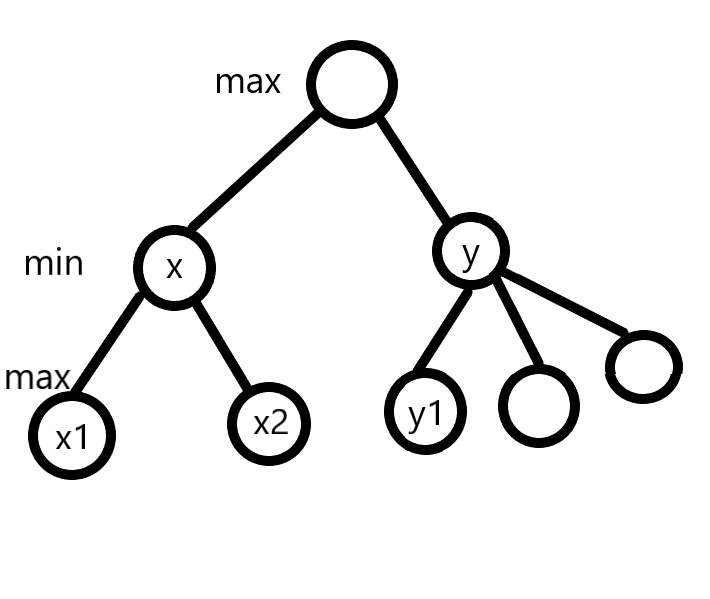
估价函数是通过对局面进行观察，评估并计算出权重的函数。在五子棋中，估价函数将局面分割成“五子”“活四”“活三”“冲四”“眠三”等一系列情况，分别给这些情况根据收益大小（即可能获胜的多少）设置不同的权重，计算白子与黑子的权重大小后作差即为当前节点的权重。

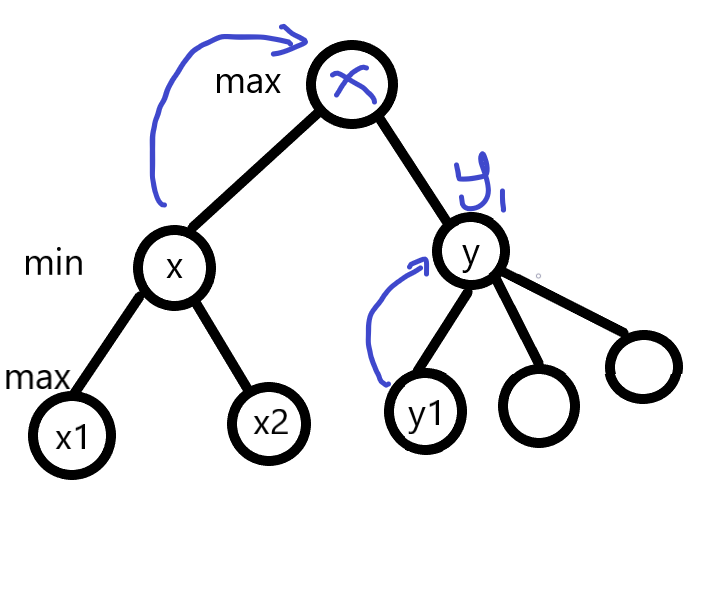
但这样的搜索是很复杂的。让我们以五子棋15\*15的棋盘为例，第一步有225种决策，第二步则有224种局面，如果想要穷举出所有决策则需要225！种不同的局面分别存储比较，这是十分不现实的。

让我们根据五子棋的特点进行一些简化：五子相连即可获胜，也就是说我们落子时仅需要考虑当前落子的周围5格范围，否则则会浪费自己的步数。但即便如此缩小成了局部的5\*5，每步也要计算出25！中不同的局面，依然是一个天文数字。

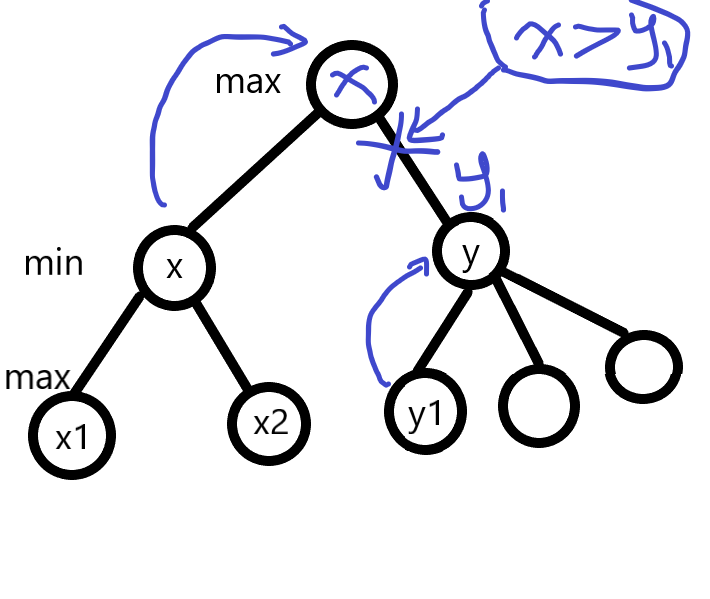
这时候我们发现，并不是所有情况都有必要计算。

看一个简单的情况：

在这个情况中x节点会计算x1和x2的大小，并选择那个小的值作为x，然后返回给父节点



接下来y节点也进行了运算，但是我们先进行一步判断：判断y1和y的父节点当前值的大小比较



当x>y1时，我们注意到由于y是min节点，它必然会选择一个不比y1大的子节点作为自己的值，而根节点作为max节点，它必然会选择一个不比x小的子节点作为自己的值，这就导致无论y的其他子节点是什么值，都不会选择它作为最佳的局面。因此我们可以将y节点剪枝。

而同理我们可以剪去不需要的max节点，唯一的区别只是判断时的符号方向。这种剪枝方法称为alpha-beta剪枝。

这样就可以在几秒内进行4-5轮搜索，可以初步的达到人机对弈的目的。

程序分为两个主要部分：电脑计算部分和绘图可视部分。

电脑计算部分：

核心在于所定义的GameTree类。GameTree类所包含的成员如下：

访问属性private：Node类：博弈树的节点

radius：在局面上的搜索范围

maxDepth：搜索时最大轮数

nodeRoot, nodeNext, openTable, closedTable: 用于建树和保存每一步的棋局

get\_empty\_pos：获取当前棋局可以作为落子点的格点

expandChildren：向下扩张树

isCut：alpha-beta剪枝

updateNode：更新节点（当前棋局）的权重

nextNode：选择下一步节点

访问属性public：构造函数GameTree()

game：电脑进行一轮的运算

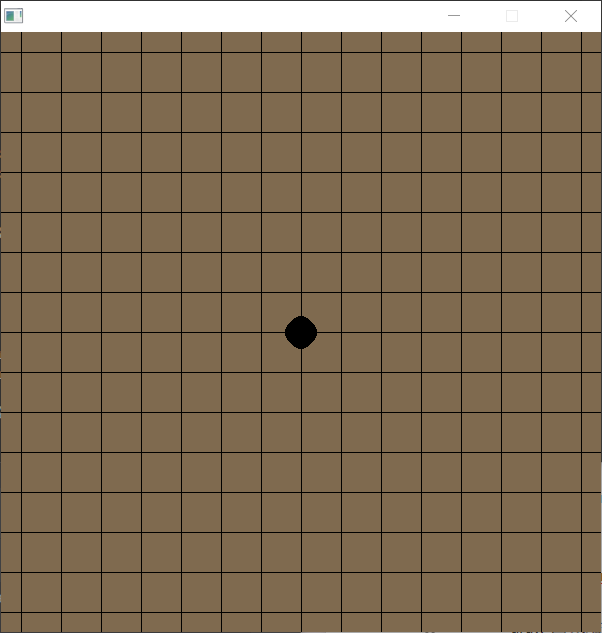
get\_next\_pos：确定电脑落子位置

除了GameTree类，还有用于启动运算的play函数

绘图可视部分：

draw函数绘制出600\*600，棕色底色背景的15\*15棋盘格子

效果如下：



1. Node类：

Node类是用于构建以及进行节点上的运算的类，其成员变量的访问属性均为public，包含下面的变量：

value(int)：若是叶节点则为该节点的权重，否则则是alpha或beta值

depth(int)：当前深度

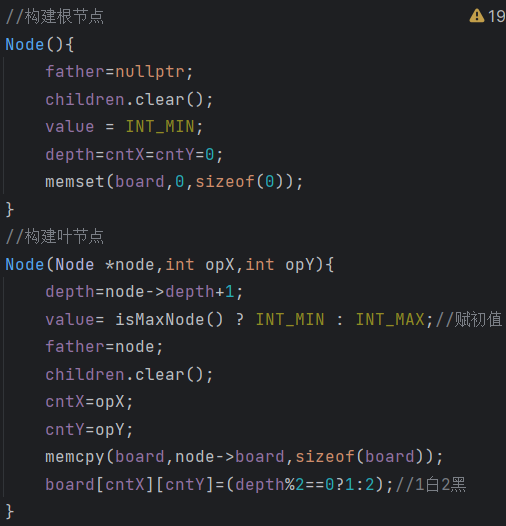
father(Node \*)：父节点

children(set<Node \*>)：所有子节点

cntX，cntY(int)：当前局面落子的x，y坐标

board[15][15](int)：记录当前局面，局面中1为白，2为黑，0为未落子

构造函数分为构建空节点与构建叶节点，代码如下：



同时包含以下函数：

isMaxNode：用于判断当前是否为max节点（此处的max节点是电脑的）

bEvaluate：对黑棋估价

wEvaluate：对白棋估价

check\_win：判断是否已经结束

evaluate：对局面评估并更新权重

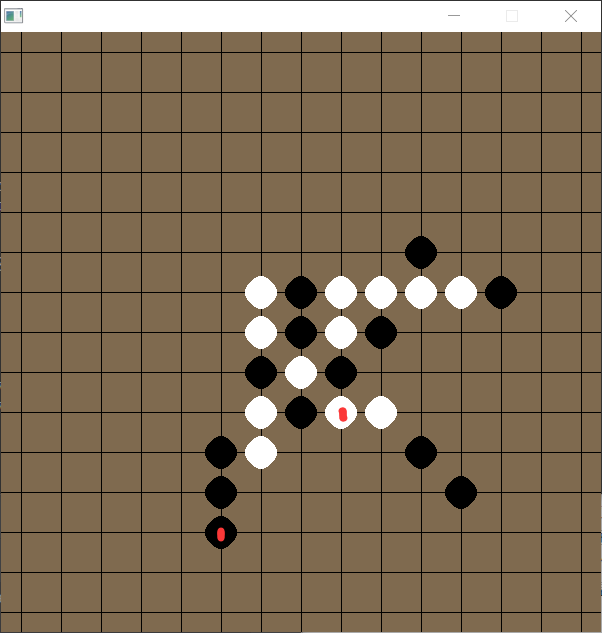
下面介绍估价函数：

这里制作的估价函数是比较简单的，因此会出现电脑走的子并不是最佳选择的情况。

由于电脑是先手，而同时更要兼顾阻拦。

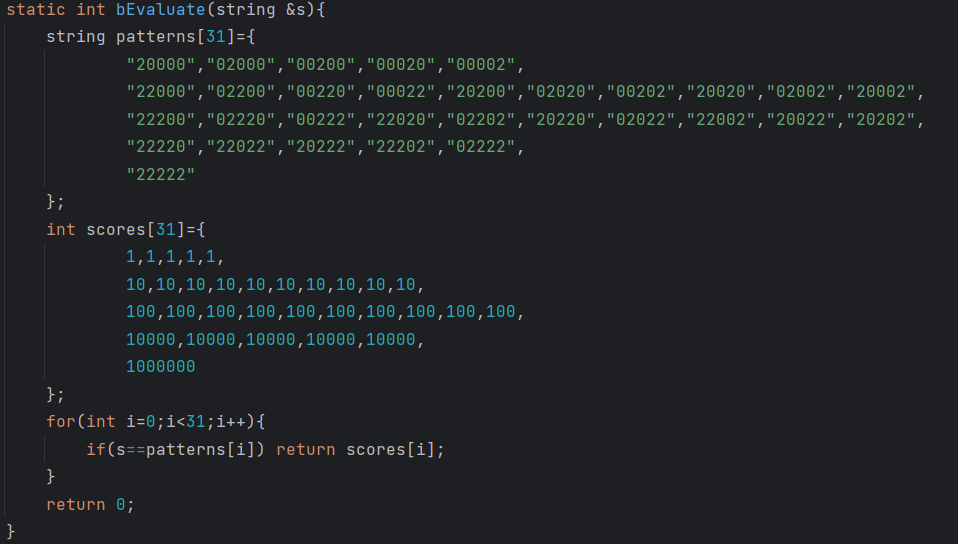
如果对黑白估价情况相同，电脑会过于注重进攻从而忽略对方的局面。

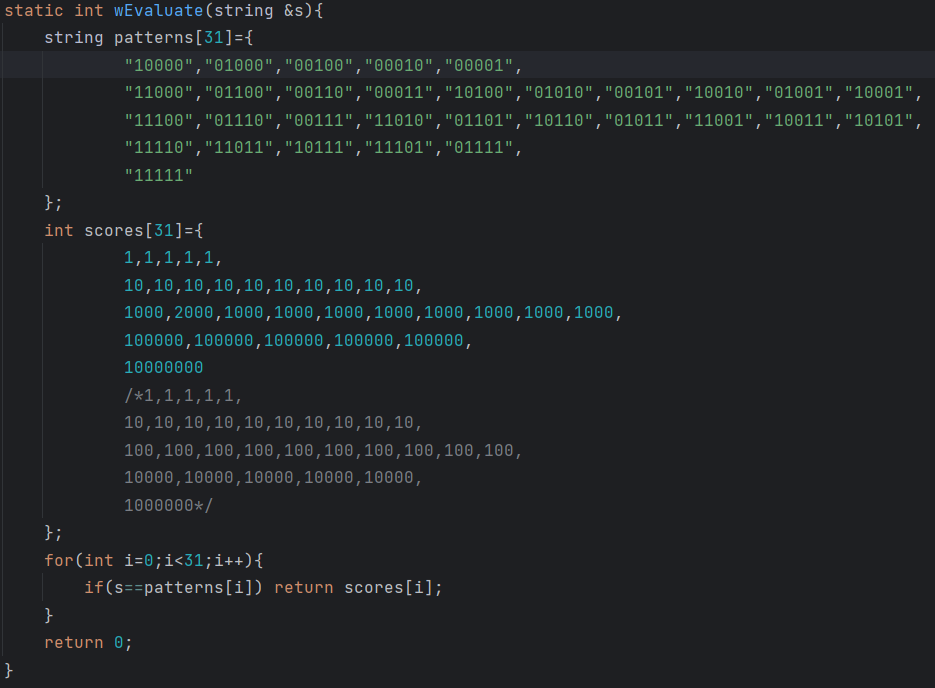
如图局面为对黑白估价函数相同时，在我下出标红的白子后电脑下出了标红的黑子，而我此时只需形成活四就将获胜。



因此为了提高防守的重要性，我将白子的“三”“四”权重进行了10倍处理，并将“活三”的权重翻倍，因为“活三”是威胁很大并且很容易做出的棋型。

估价函数如下：





注意到进行估价时均只考虑了五元组，这也提示我们在进行搜索时可以围绕一个棋子周围的所有五元组进行搜索。

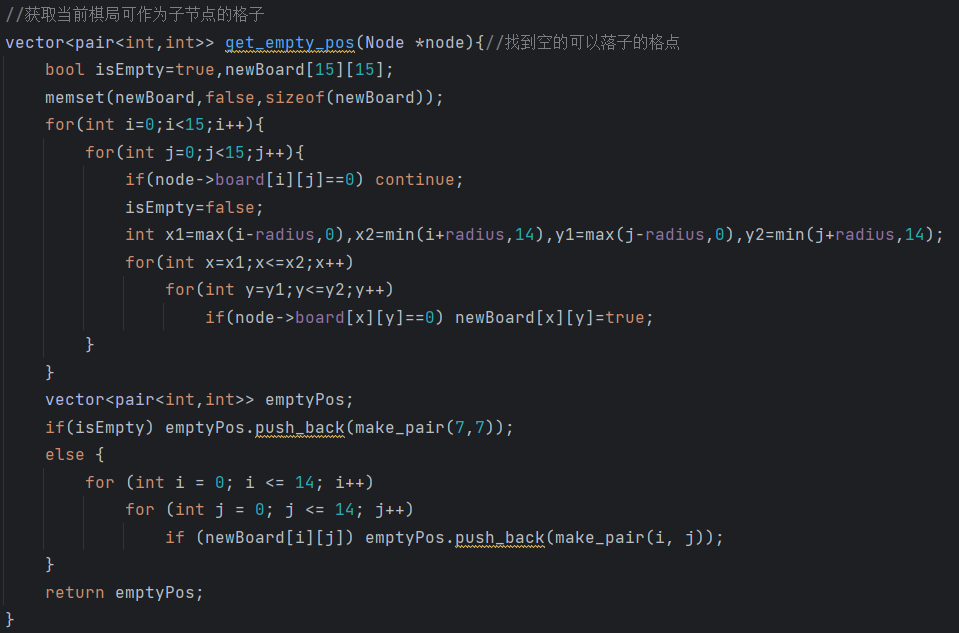
1. 局面相关函数

Node类主要负责了对单个节点信息的处理，估价。而GameTree中的其他成员

函数则负责了对局面进行信息获取和处理。

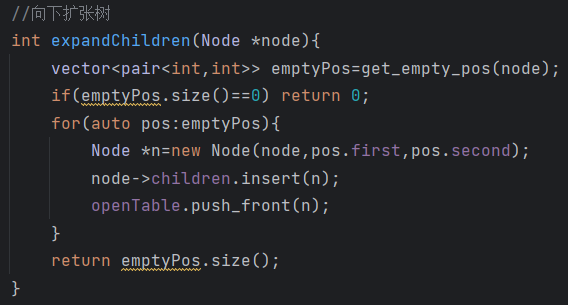
get\_empty\_pos函数用于找到空的可以落子的格点，根据所设定的查找范围可进行范围内的所有可落子格点的查找。这里设定了范围2，是由于从该格点向左向右2格加上本身一共是5格，恰好是五子棋的合理落子范围。当收集到了所有可能落子的格点将会返回所有可能格点的坐标。

代码如下：



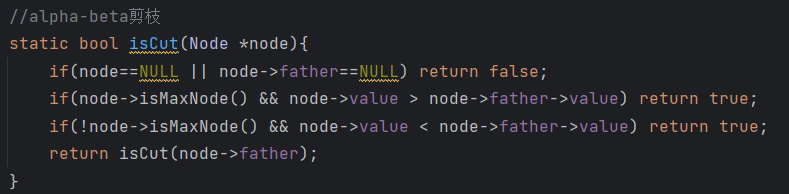
expandChildren函数会根据get\_empty\_pos函数获得的所有潜在子节点对树进行扩张，将其作为当前节点的子节点，并返回该节点子节点的个数。

代码如下：



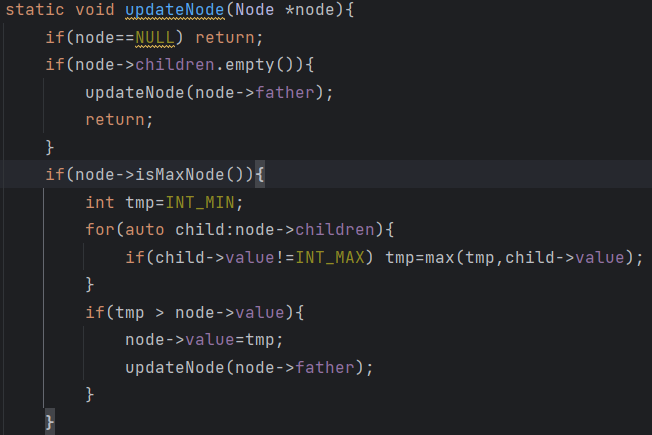
isCut函数用于判断是否能够alpha-beta剪枝，返回值为布尔型。

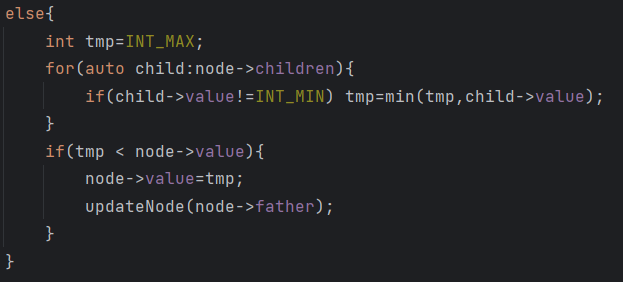
代码如下：



updateNode函数用于更新节点的值，更新时会从叶节点开始向上直到父节点依次进行，分为max节点和min节点两部分，原理类似，在上文已经提到。

代码如下：





nextNode函数只需要选择子节点中价值最大的节点即可。

1. 执行部分

game函数用于连接各个函数并运行。

程序中使用队列openTable用于保存当前的所有节点，每次出队一个节点进行搜索。搜索流程如下：

判断当前局面是否已经结束，如已有一方获胜则会直接返回

将根节点推入空队列

接下来进行循环，结束条件为已经没有剩余可搜索节点：

循环中每次出队一个节点，先判断是否能被剪枝，再判断是否在最大允许搜索深度内（此处设置为5轮，即depth为9），如果不能被剪枝且在允许范围内，则会对树进行扩张（expandChildren函数，该函数返回值为扩张得到的子节点数量），并获得子节点个数。当搜索到叶节点或者已经到达最大搜索范围后，停止扩张树，并对局面进行评估，并从下到上更新树的节点的权重。这是每一个节点都进行的操作。当搜索完毕，会根据权重选择最合适的节点作为下一步，电脑记录局面。而由于每一次的根节点都是根据局面获得的，因此下一次执行该函数时就相当于从此局面继续搜索，估价，从而达到对弈的效果。

代码如下



这里game的返回值决定了是否要结束程序。

1. draw

窗口大小为600\*600。

绘制帧为60。

绘图都是基于一张图片反复进行操作。该图片初始为600\*600的棕色纯色图片，并在20-580之间以40为间隔进行黑色直线的绘制画出棋盘。对应进行相应格点坐标的计算并储存在pointx和pointy中，调用时即可获得格点（如（7,7）（9,10））的坐标。

整型变量flag则用于控制绘制棋子，存在的状态有0,1,2,3,4。

flag为1：即轮到电脑进行计算并落子，根据当前局面建树，并调用play函数。play函数中会调用game获得得到下一步落子的点，并修改局面。接下来获得落子的点坐标用于绘制黑子。同时修改flag为2

flag为2：会在循环结束时将flag修改为0，而在玩家按下鼠标之前，flag又会变回2，如此反复，从而给玩家思考的时间。一旦玩家按下鼠标左键落子便会立刻获得坐标并修改局面和绘制白子，同时修改flag为3

flag为3：会在循环结束时将flag修改为1，下一回合轮到电脑执黑子。

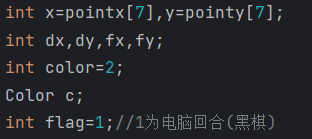
flag为0：开始判断鼠标左键是否按下，同flag为2中所述

flag为4：游戏结束

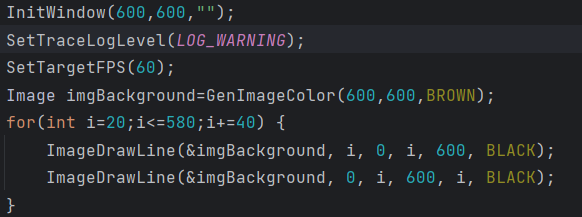
当flag不为4时，程序会根据获得的坐标以及落子方绘制相应位置相应颜色的棋子在图片上，接下来转化为texture并绘制，结束后将texture释放。

当整个循环结束，即窗口将要关闭时，释放image，紧接着关闭窗口，退出程序。

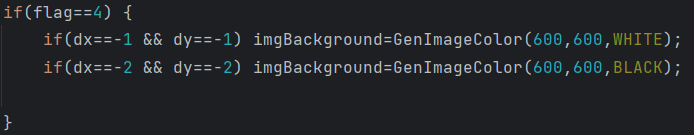
变量如下：

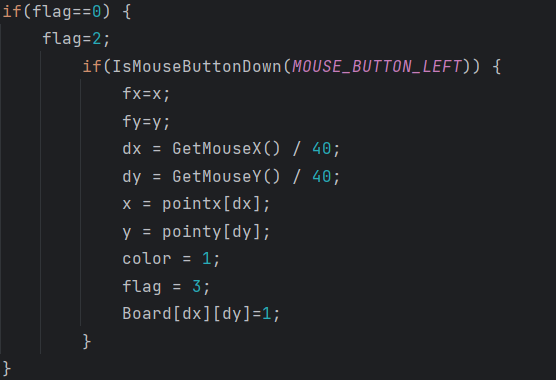


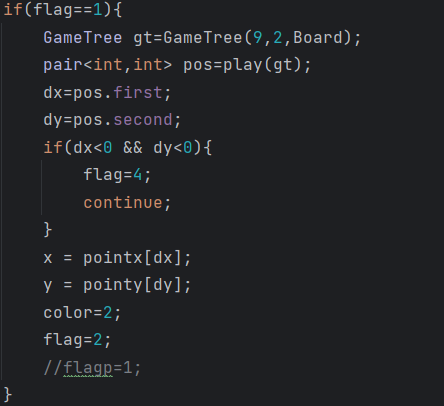
绘制准备工作如下：

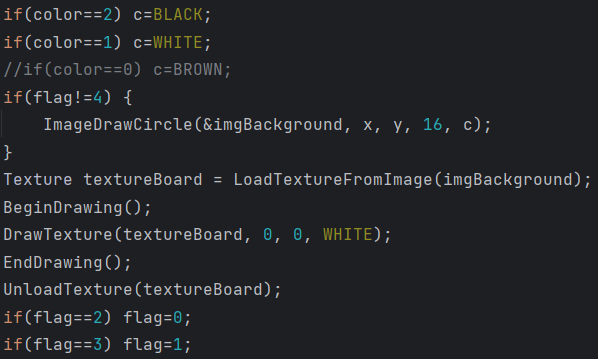


绘制工作如下：

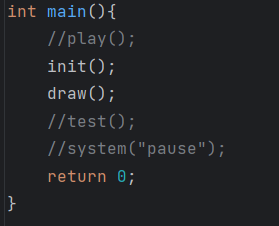








最后主程序如下：



#### 四、其他

该程序是初学者现学现写，存在诸多不足。

首先是程序的功能并不强大，只有简陋的手绘界面以及不足够聪明的电脑玩家。经常出现显而易见的“双三”而漏掉。但是人也会失误，不仔细也很容易输给电脑。至少在复杂程度大的局面下，电脑战胜我的概率会很大。

其次是draw函数的拐弯抹角和冗余实在是太多，我也很不喜欢这样子的代码，但是我的水平有限以及时间也不充足，就采用这样的方式来控制绘制。Raylib并不是一个好的绘图软件，它更适合做动画（吧）。而且它对于初学者相当不友好，网站没有详细的教学，国内使用的教程也并不多，最后还是根据别的使用raylib的程序里的代码猜这个函数的用处（好在命名很到位）。

最后则是源代码中存在着大量注释掉甚至未被注释掉的函数，均为测试用，我并没有直接删去，毕竟在没有raylib的条件下可以用它们在console中进行对弈。

这个五子棋是我写过的最大的程序，也给我带来了很多很多收获，第一次去使用类定义，去分装这么多个函数，把他们组合在一起最后成功运行真的是一件非常开心的事情。如果说计算部分是现学现用照葫芦画瓢，那手绘加上将计算部分与绘制部分相连接真的就是花费了很多时间和精力的努力成果了。从一知半解到完全理解这个模型，完全吸收了每一行代码，对我来说已经是一件很意义重大的事了。