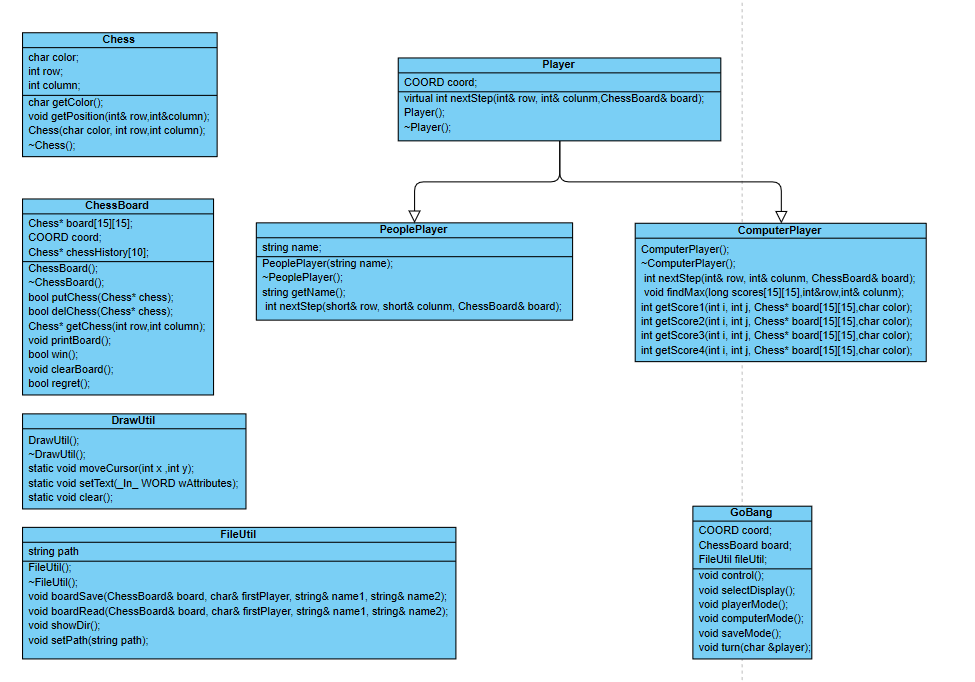
# 五子棋

类图：



GoBang类：

整个游戏的主控类，连接所有类。整个游戏的主逻辑在GoBang类中实现。主逻辑在control函数中，不同模式的实现分别在playerMode computerMode saveMode中实现。

Chess类：

棋子类，包含棋子的颜色和棋子所在的行和列的信息

ChessBoard类：

棋盘类。棋盘类里面有一个15\*15的数组用以存储棋局信息。同时有对应的下棋（putchess）、悔棋（regret）、判断胜负（win）、打印空棋盘（printBoard）等函数。

Player类：

Player类里面定义了一个nextStep虚函数，用以得到下一步棋子的行和列值，在父类中不进行具体实现，用以交与子类PeoplePlayer和ComputerPlay分别重载实现。

DrawUtil类：

控制台画图工具类，里面中有移动光标（moveCursor）、清屏（clear）、设置字体颜色（setText）等静态函数

FileUtil类：

文件工具类，用以实现文件流操作，来进行存档（saveBoard）和读档（readBoard）操作。

下面对一些主要的实现逻辑进行说明，具体可参考源程序，源程序中有充分的注释。

主控部分：



人机对战模式：

人机对战的机器算法实现，考虑到计算的复杂性，因此在程序中只采用了简单的方法进行计算得出最终的结果。核心思想是，只有在有棋子的附近的格子的分数才要计算。

如图中的棋局，只有蓝色椭圆所标记的格子的分数才需要计算，其他格子是不用进行计算的。

分数计算过程：

对一个格子进行评定，需要对该格子的横向情况，纵向情况，上斜情况，和下斜情况四种情况的棋局都进行判断，因此，需要对每一个valueGrid的附近的棋局进行扫描。

横向扫描：

先向左，若遇到己方棋子，则count++；

若遇到空格，则停止扫描。

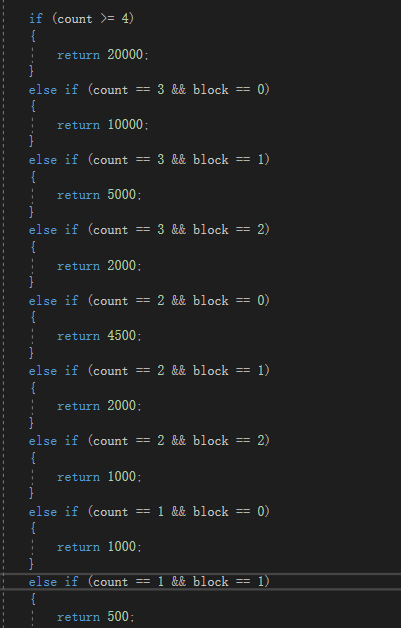
若遇到对方棋子，则，block++，并停止扫描。

再向右，进行同样的扫描的操作。

向左向右扫描完后，检测count和block的值。Count值为n的意思是左右两边共有n个相邻的同色棋子，block为m的意思是左右共有m个对方的棋子。若count的值为4且block的值为0，此时说明，左右两边共有4个相邻的同色的棋子且无对方的棋子，此时在此格中下棋的话，就能得到5子的情况，故此时得到的分数应该很大（程序中用的20000）。若此时count为3，block为0，此时说明两边共有3子的情况，且无对方棋子，此时在此格中下棋可得到单4子的情况，因此此时分数应该也比较高。程序即是对不同的情况进行对应的评分，最终得到横纵上下斜四个方向的总方向。

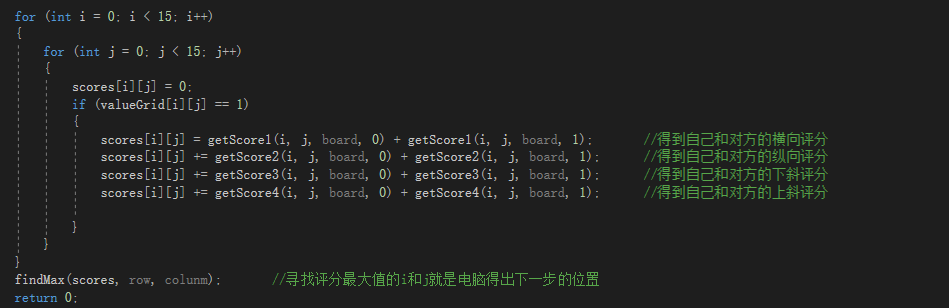
纵向、下斜、上斜的操作与横向的类似，具体实现请参考源程序。

## 评分表：



说明：计算分数时，不应该只考虑自己的分数，还应该考虑到对方的分数，不然电脑将只会攻击而不会防守，会导致一些极端的情况。

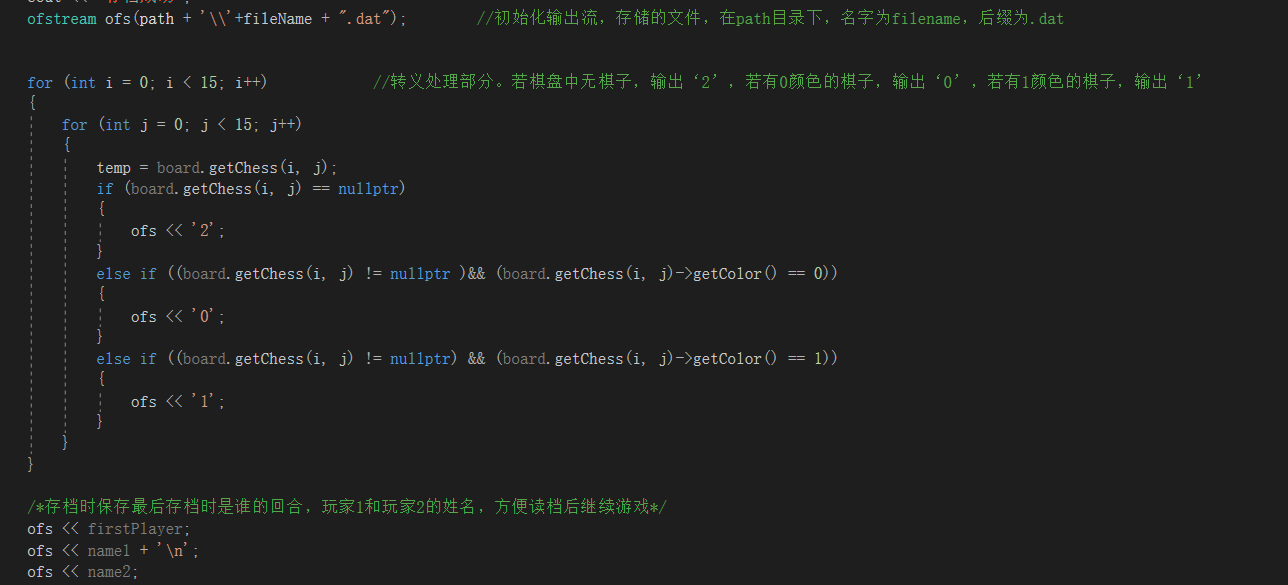
最终分数计算



最终返回的行和列值是分数最大的那个格子的i和j值。

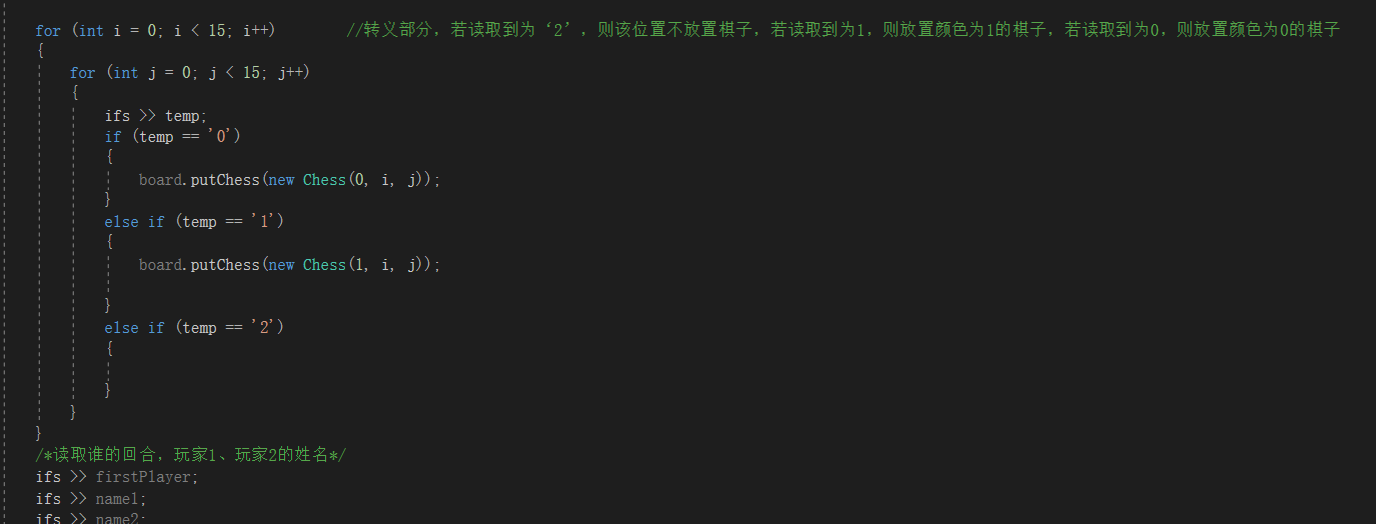
### 读档存档模式：

由于个人能力有限，不知道如何将C++的对象序列化并直接将对象保存到本地，源程序中存档操作是通过转义的操作。通过自己设定的协议，将对象中的信息转义后通过文件流的方式写到本地文件中。



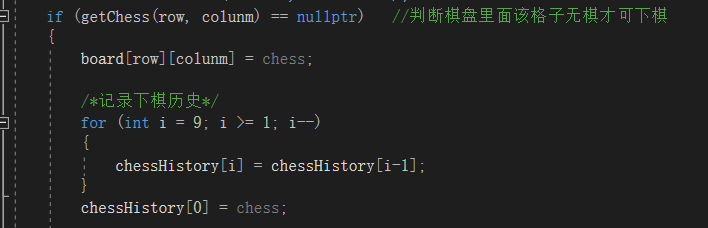
具体转义方式，参考源程序中的注释。

读档操作也是根据存档时规定的协议进行读取。

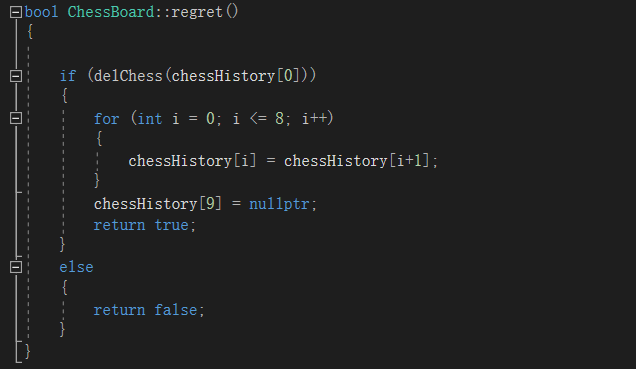


#### 悔棋操作

在每次下棋时（putChess），会把此时所放置的棋子的信息放到chessHistory[10]数组中，因为源程序中数组长度仅设为10，故最多只能悔10步棋

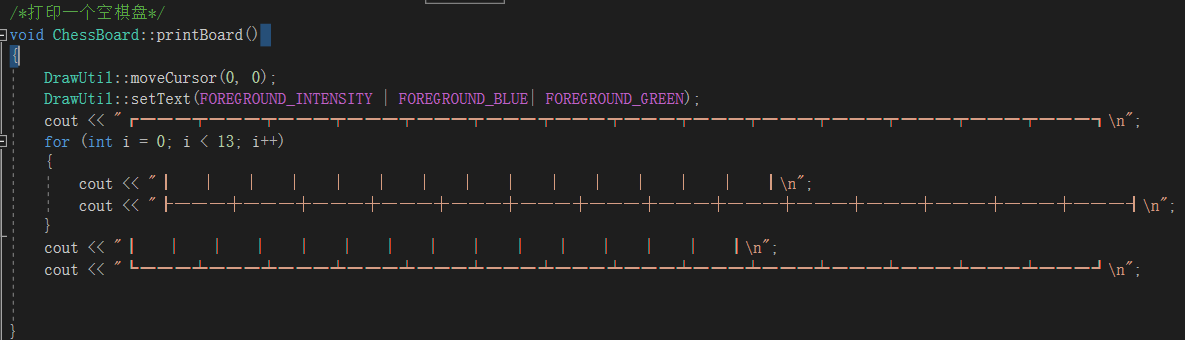


保存后悔棋操作即将最新的历史的棋子删除即可（chessHistory[0]）



#### 棋盘绘制

在对应位置输出对应的符号即可



下棋时，先把光标移到对应位置（moveCursor），再输出

