**开发文档**

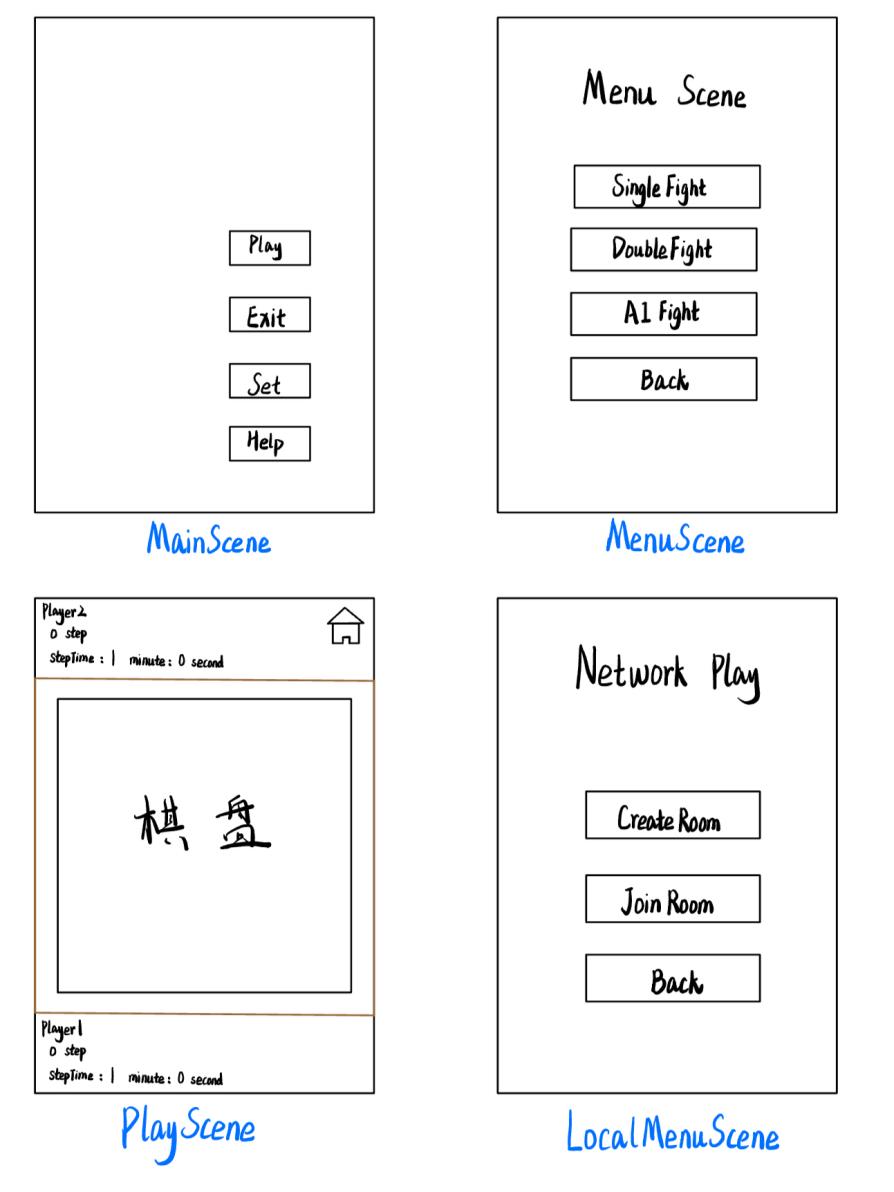
**软件名称：**

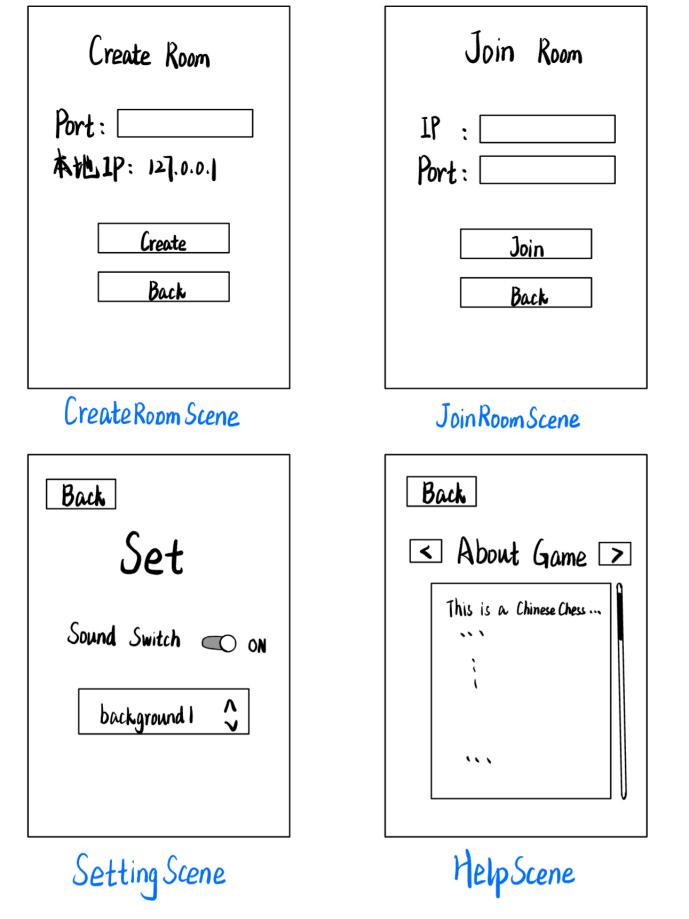
中国象棋-云雨星项目

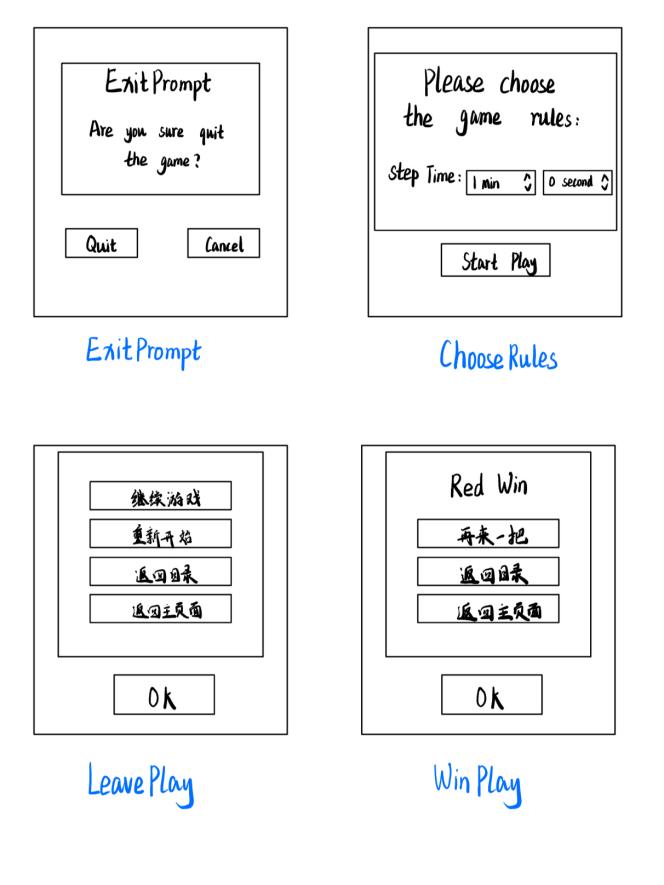
**软件作者：**杨云博、代鑫、唐雨

**软件原型图：**

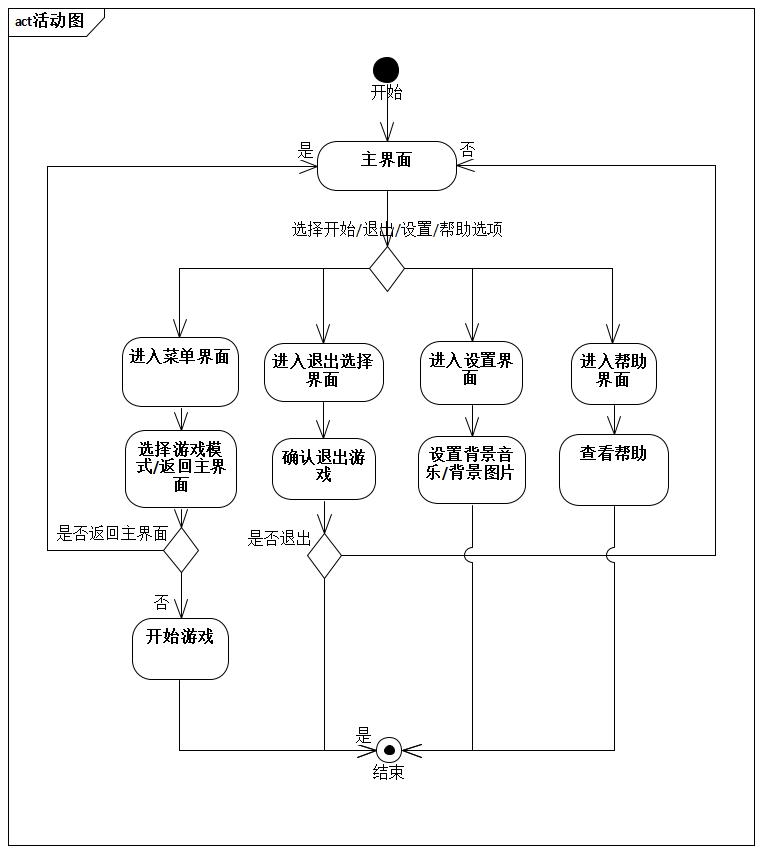
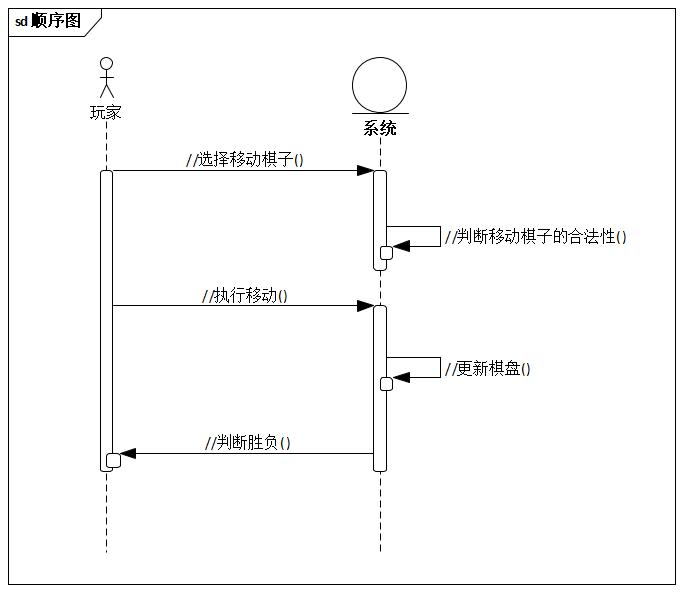
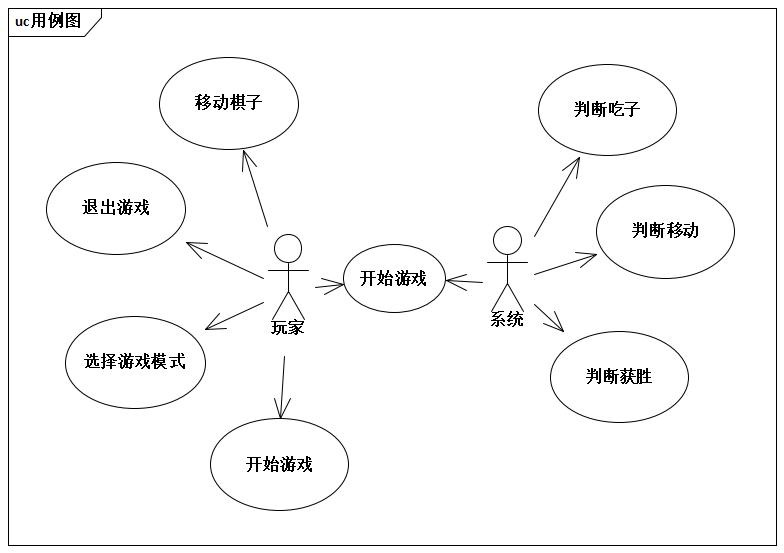
本项目的基本窗口布局如下图所示：







**软件模型图：**



**总体设计思路：**

**项目总体结构：**

头文件和源文件是进行网络编程使用的；

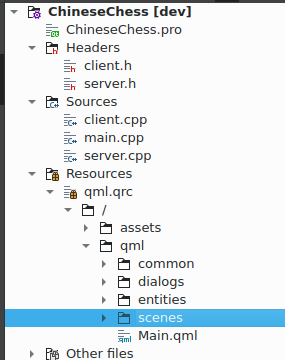
QML文件全部放在资源文件夹qrc中，其中assets文件包含所有项目中使用的图片和音效，音乐；

QML文件夹中common文件夹是项目中一些基类组件，我们将按钮和场景和棋子等设置了一个父组件，然后在场景中对这些父组件进行复用；

dialogs文件夹包含所有的对话框组件，通过在场景中设置对话框的visible来显示和销毁对话框；

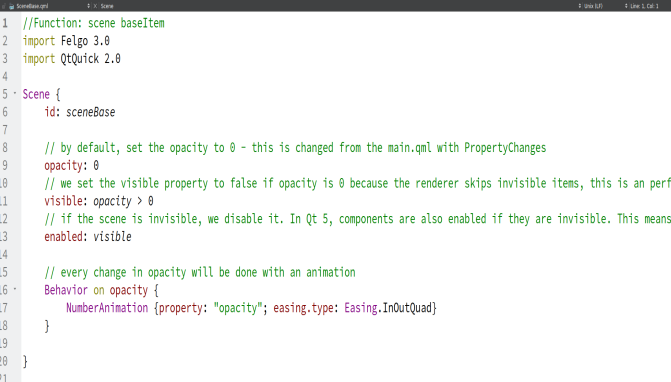
entites文件夹包含项目中所有的实体类，其实也就是全部棋子和玩家的组件，相当于泛化了common文件夹中的棋子和玩家组件；

Scenes文件夹中包含所有的场景文件，场景的布局便是在这些组件中设计的。

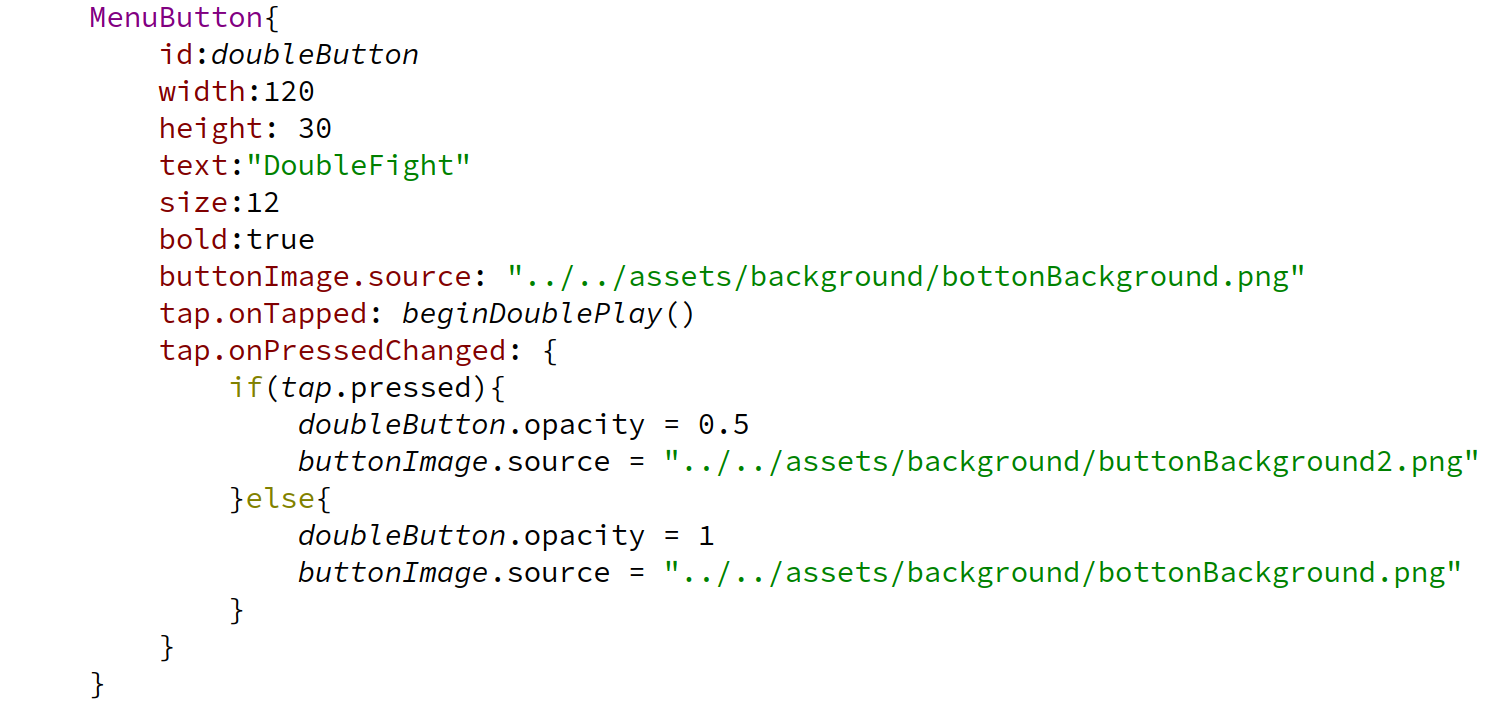


**具体设计思路：**

**场景切换：**通过将所有的场景放置在main.qml中，然后类似于状态机，通过将每个页面设计成一个状态，然后通过点击场景中的按钮触发场景的槽函数，调用槽函数切换状态来达到切换页面的效果，所有场景都是基于common中的SceneBase.qml父组件，这个父组件中设计了一个动画效果，通过改变场景的透明度的变化达到场景切换淡入淡出的效果。



**按钮处理：**我们设计了两个按钮父组件，MenuButton.qml和ButtonBase.qml，前一个是用在场景中的按钮，后一个是用在对话框中的按钮，其实做完之后感觉就设计成一个都可以了，当时不晓得为啥想着这样分。按钮主要就是处理他的TapHandler事件，通过处理事件调用场景或者对话框的信号槽。其中按钮有通过处理按钮按下和释放的事件改变按钮的透明度和背景图片的高亮来判断确认该按钮被点击，呈现出不同的状态效果。



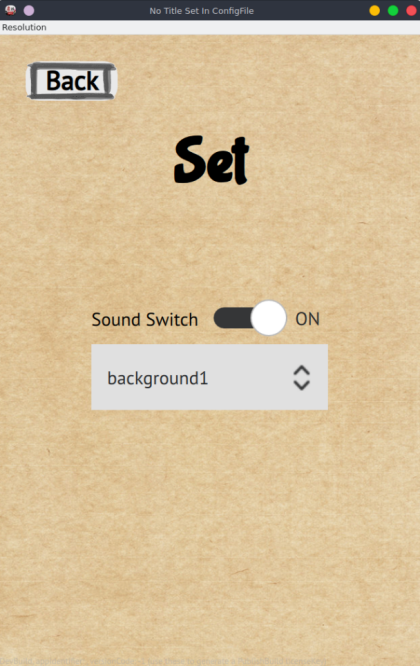
**主页面设计：**主页面的主要构造是由一张背景图片和四个按钮组成，其实现文件的具体结构如下图所示：



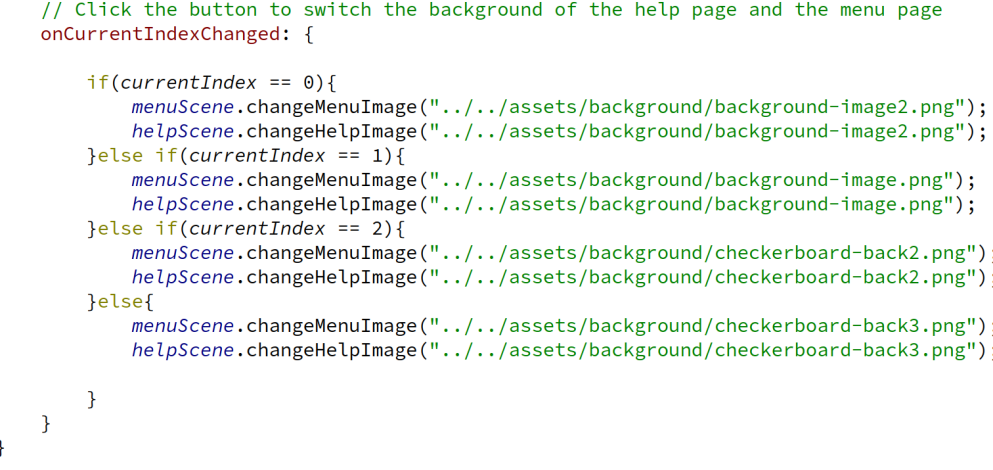
**菜单页面设计：**菜单页面的主要构造是由一张背景图片，一个文字标题和四个菜单按钮所组成，其实现的效果如下图所示：



**设置页面的设计：**设置界面主要由一张背景图片、一个文字标题、一个返回按钮、一个switch开关按钮、和一个comboBox下拉菜单组成。通过开关按钮我们能够实现背景音乐的播放功能，通过下拉菜单我们可以选择切换全局的背景图片，其实现文件效果如下图所示：



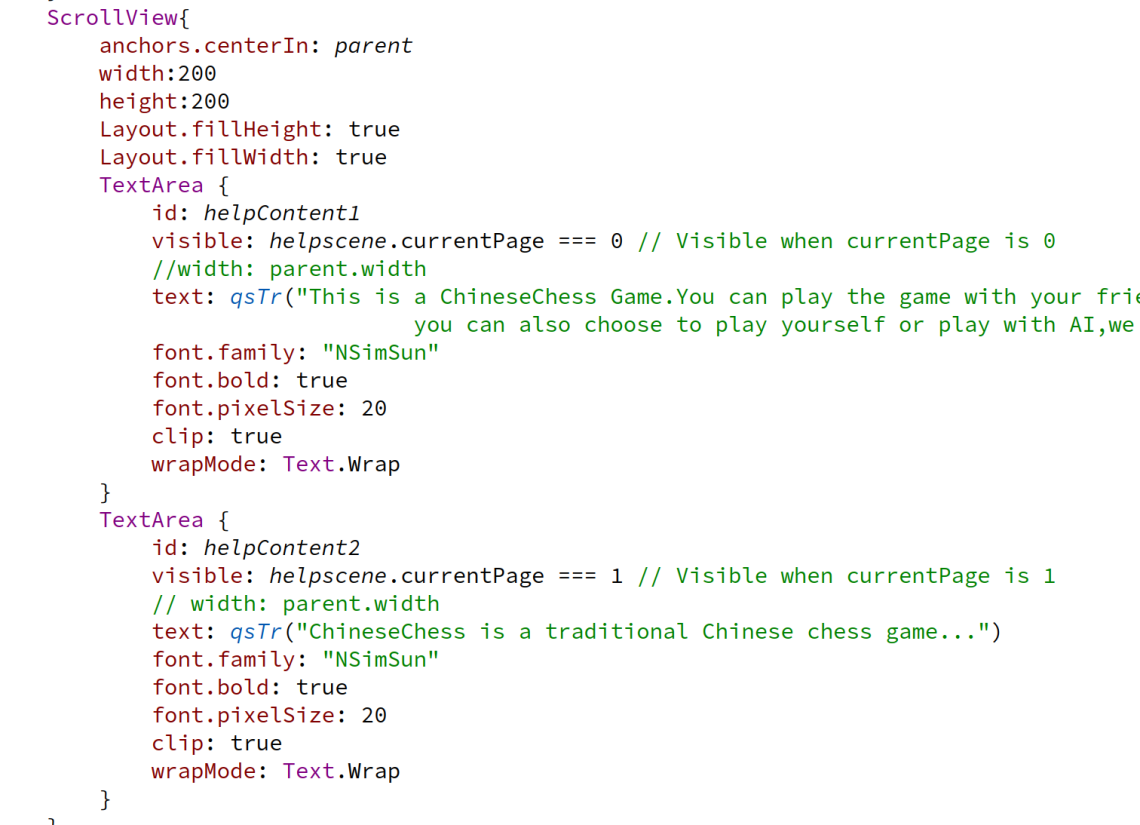
切换全局背景图片是通过处理combox当前值的槽函数来调用其他页面的函数，使其他页面的背景图片路径更改。



**帮助界面的设计：**帮助界面主要由一张背景图片、一个返回按钮、两个切换按钮、一个文字标题、和一段帮助说明和一和滑动条组成。其中两个切换按钮可以实现切换说明内容，滑动条可以实现页面内容的滚动访问。其实现文件的效果如下图所示：



页面点击左右按钮切换中心内容是通过点击左右按钮后改变当前组件的一个属性值currentPage,然后在滑动视图中判断当前页面的值来使某个页面呈现。



**游戏界面设计：**

首先加载游戏界面时会先跳出一个对话框供选择玩家的局时，选择完成后确认即可将玩家的局时时间设置成相应的时间，然后将棋盘的棋子初始化到相应的位置上去。

我是将游戏界面和游戏逻辑分离开的，游戏界面中只是放置了界面的背景图片，玩家1和玩家2的相应的局时，棋盘图片，棋子图片，退出游戏对话框的按钮和所有的对话框，对游戏界面中的一切处理都是通过加载到游戏界面中的gameLogic中实现处理的。而GameLogic是一个单独的组件，里面包含了许多函数，以供逻辑处理。

我将棋盘表示成一个以棋盘左上角的坐标轴，首先我通过截图软件计算出每一个格子的长度，然后将每一个格子作为坐标1来判定，左上角那个点的坐标是（0,0），然后划分棋盘，棋盘分为（0,0）～（9,8）。初始化棋子时，就将每个棋子x,y值通过坐标换算（坐标乘以格子大小）表示。

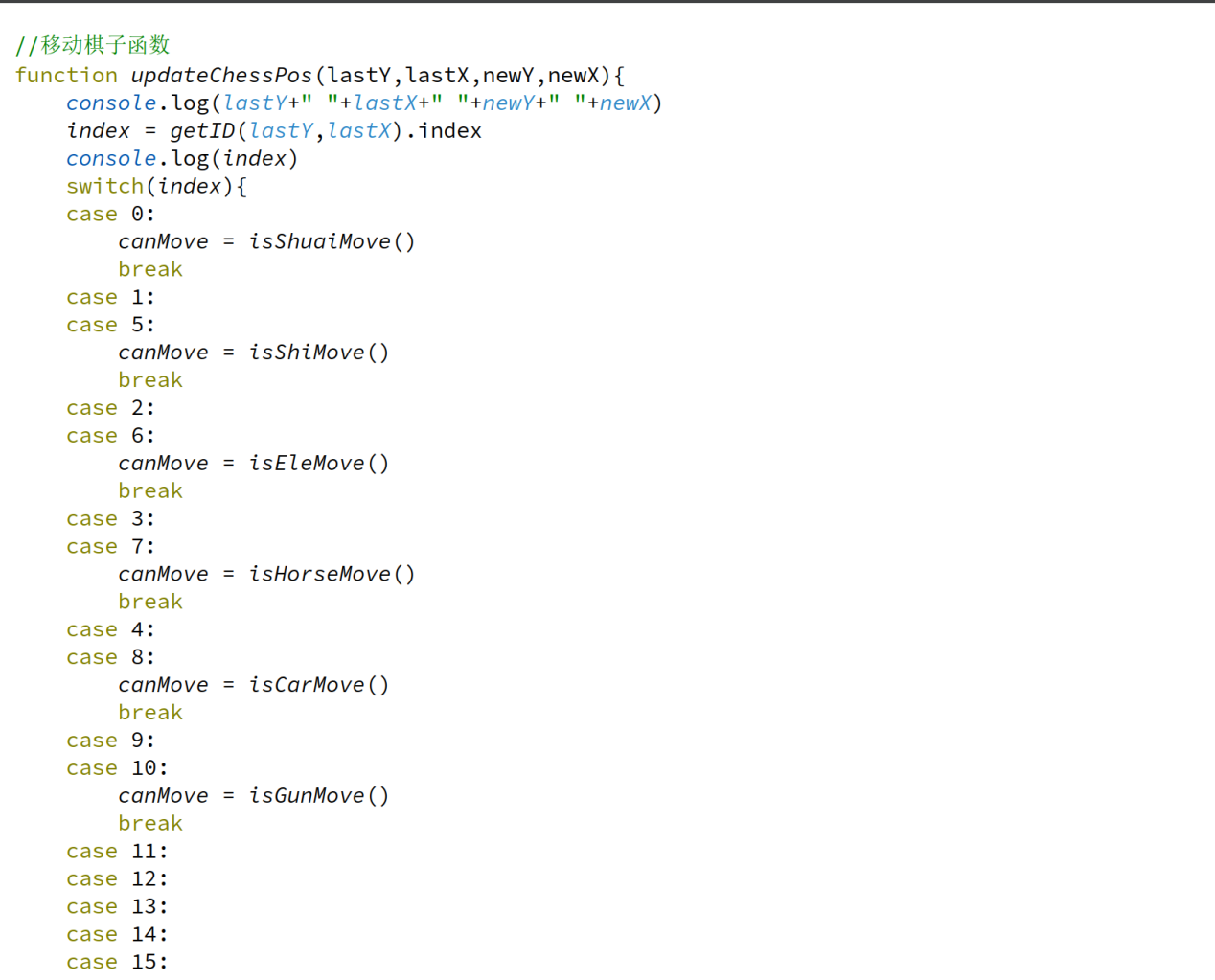
棋子的移动我是通过处理棋盘的TapHandler事件获取点击的坐标，然后通过换算成坐标轴的点位，在gameLogic.qml中设计了lastRow,lastCol,newRow,newCol这四个值来记录第一次点击的坐标和第二次点击的坐标，mouseCount记录点击次数

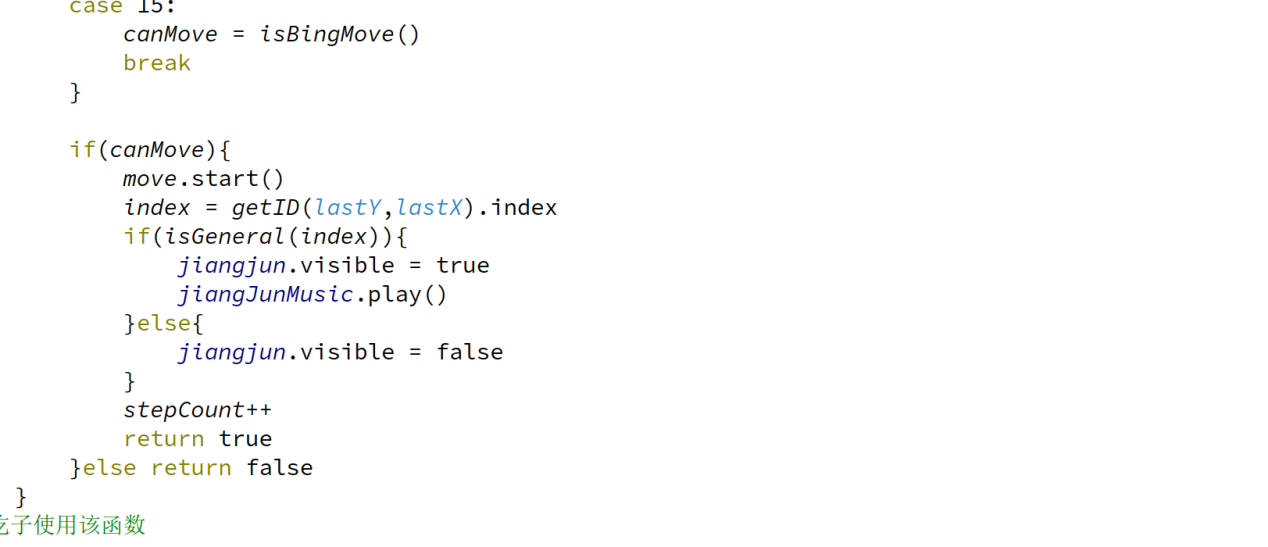
选择棋子移动的逻辑如下：首先先判断是否是第一次点击再判断点击处是否是棋子，不是则不记录坐标，第一次点击选中了棋子才会保存选中的坐标值到lastRow,lastCol,并保存选中的棋子阵营，若第一次点击不是当前阵营回合会弹出不能移动的对话框提示；第二次点击先判断是否是点击同阵营的棋子，若是同阵营则更改之前保存的第一次点击的坐标值，点击次数又改为一次；若不是同阵营则第二次点击是否有棋子，没有棋子执行移动棋子的函数，有棋子执行吃棋子的函数，执行完后将点击次数设置为0。



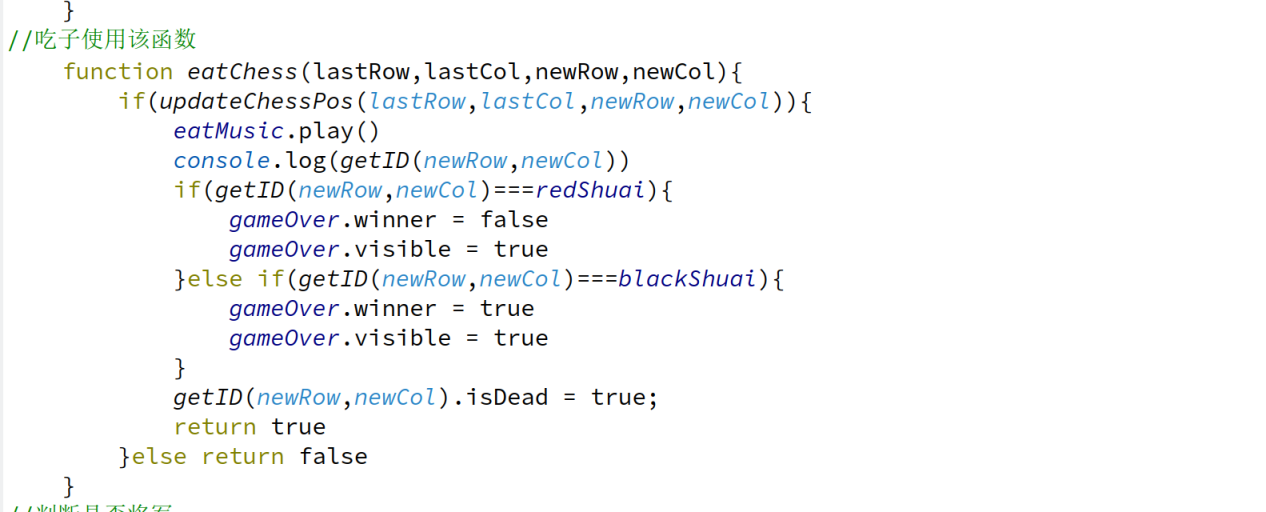


**移动棋子的逻辑如下：**首先先保存需要移动棋子的类型值（就是判断这个棋子是车、马、士、炮、帅等棋子类型），然后通过类型值去调用不同的判断当前类型能否移动到相应点去，若判定无法移动，就不能动，可以移动才会调用一个动画序列来改变棋子坐标从而移动棋子到相应坐标上去。移动过去之后再判断棋子当前坐标与对方大王的坐标，看棋子能否移动到对方大王的位置去从而达到是否将军。

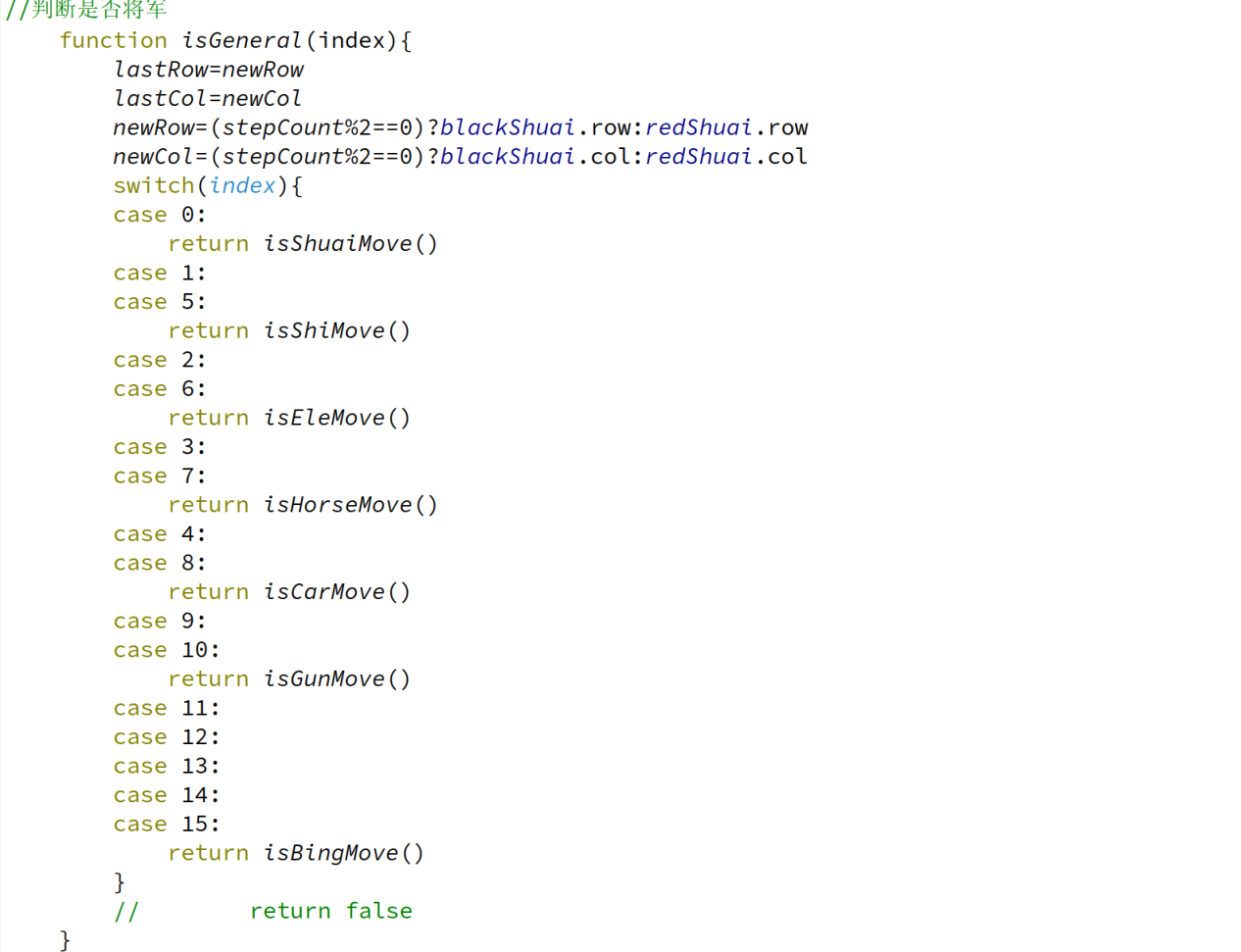




**移动棋子吃子的逻辑：**其实就是移动棋子，然后将那个棋子的属性isDead设置为true表示该棋子死亡，死亡就将他的可视性变为false。其中若是红方大王死亡或者是黑方大王死亡，就会导致游戏结束的对话框显示。



**将军的逻辑：**就是将原来的坐标修改为当前坐标，并将需要移动的坐标设置成红/黑方大王的坐标，然后根据类型判断当前棋子能否移动到大王的坐标去，能就将军，不能就不能将军。这种逻辑只是简单将军逻辑，而且会因为更换阵营移动后若不是为了不被将军的移动，还会导致将军消失（其实本质上还是将军着的）



以下函数是判断棋子类型是否能够移动的函数：

**判断大王能否移动**我是将大王可移动的值限定到九宫格内，并且每次移动只能横向移动一格或纵向移动一格，斜着移动是不行的，是先判断需要移动坐标和棋子坐标是否是在一条线上的，再来判断是否移动的是一格。

**判断士的移动**是限制在九宫格内，然后每次移动是横坐标和纵坐标都移动一格才能移动。

**判断象能否移动**是先判断这个象的阵营，再通过阵营将象的移动范围限制在自己阵营方的河内，然后每次移动是横坐标和纵坐标都移动三格才能移动。

**判断马能否移动**是将马能移动的八个点标记出来，然后判断需要移动的坐标和棋子坐标的差来判断是走哪一个点，再判断要走那一个点会不会遇到崴脚，遇到崴脚就不能移动，崴脚点没有棋子就可以移动。

**判断小兵移动**是先判断他的阵营，然后在自己阵营内只能向前移动一格，过了河后可以横向移动一格也可以继续向前移动一格。

**判断车和判断炮的移动**都会涉及到一个函数getChessCountAtLine（），这个函数是判断移动的坐标和棋子坐标中间有多少个其他棋子，炮的移动需要中间只有一个棋子才能翻山，车的移动需要中间一个棋子都没有才能移动。



**一个简单AI的实现逻辑：**其实移动的逻辑跟一般移动的逻辑大致相同，但是当时想的是把每一个场景的实现逻辑区分开，但是其实本身逻辑也差不多，分开了到感觉是代码重用了，不过因为AI玩家移动一次后就让系统自己移动一次，所以当时还是分开写的，主要是让我自己当时写的时候逻辑更清晰。

然后AI实现是将棋子定义了分数，每个类型的棋子都有一个自己的分数，通过计算场面中红方棋子总分和黑方棋子总分，然后相减，fakeMove和unFakeMove是将棋子假设移动，因为AI需要循环每一个棋子的移动方式来获取每一次移动的局面分，从而选择局面分最高的那次移动，所以需要假装移动和还原移动。

如果所有棋子循环完都不能移动，则随机选一个棋子想下移动一格（因为当时不想再搞复杂了，只是想实现一个简单AI,所以就直接设置了一个改变规则的算法了）。







**网络对战连接逻辑：**为实现这一功能，我们设计了服务端和客户端。通过服务端和客户端的连接，再进行数据传输，即可实现双人在不同主机上进行象棋对战。

首先，我们编写了两个类，Server类（服务端）、Client类（客户端），再在两个类中编写程序实现相应功能。这两个类的主要目的为实现客户端和服务端的连接和实现客户端与服务端的数据传输。

在最基本的服务端与客户端连接实现中，我们主要运用Socket编程接口实现网络连接，网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个socket。在连接过程中主要分为以下三个步骤：

1、服务器监听：是服务器端套接字并不定位具体的客户端套接字，而是处于等待连接的状态，实时监控网络状态。

2、客户端请求：是指由客户端的套接字提出连接请求，要连接的目标是服务器端的套接字。为此，客户端的套接字必须首先描述它要连接的服务器的套接字，指出服务器端套接字的地址和端口号，然后就向服务器端套接字提出连接请求。

3、连接确认：是指当服务器端套接字监听到或者说接收到客户端套接字的连接请求，它就响应客户端套接字的请求，建立一个新的线程，把服务器端套接字的描述发给客户端，一旦客户端确认了此描述，连接就建立好了。

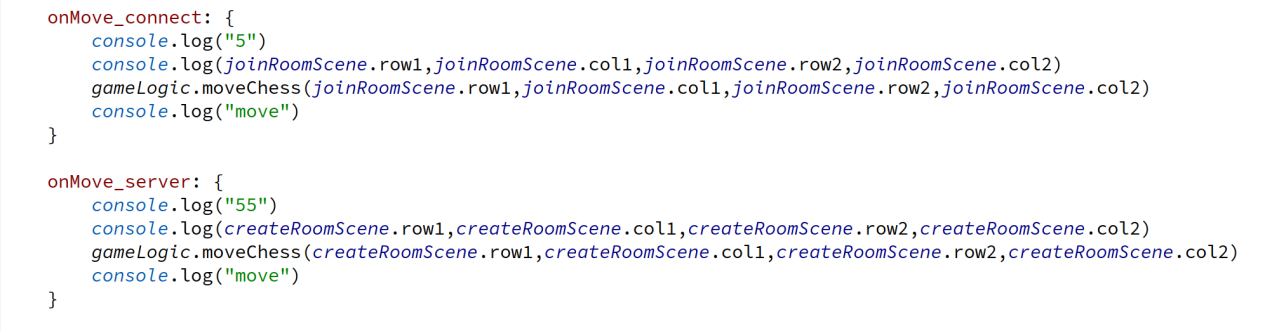
我们在Server类设置了两个主要变量ip、port，Client类中设置主要变量port,通过ip地址和端口号的配对实现连接。

服务端与客户端数据传输功能实现则基于本项目的下棋基本逻辑实现。为实现双人对战，我们需要将qml与所创建的Server类与Client类进行连接，为区分用户谁为服务端，谁为客户端，我们在选择双人对战模式后通过创建房间和加入房间两种选项实现角色选择，即进入创建房间页面，输入连接端口，创建房间，充当服务端;进入加入房间页面，输入要连接的ip地址和端口号，进入该ip地址用户所创建的房间，充当客户端。当两个用户进入了同一个房间，即实现了连接，接下来便可通过逻辑的数据传输实现双人对战。

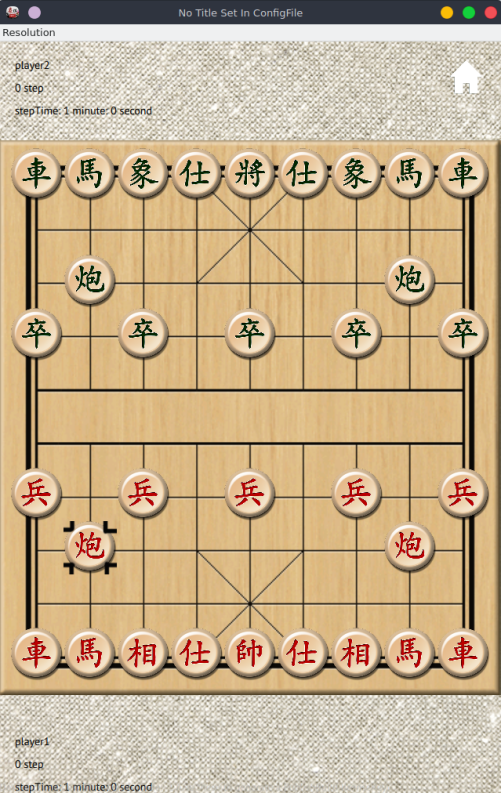
**网络对战传输数据逻辑：**连接成功后将客户端的界面阵营设置成黑方，服务端的界面设置成红方，方便逻辑处理中判断是否该哪一方阵营移动。每次移动棋子后，客户端向服务端或者服务端向客户端调用sendMes槽函数传输棋子坐标和移动坐标数据，客户端或服务端收到数据后则调用游戏的移动棋子函数，因为逻辑写的时候可能针对一个功能时挺清晰的，但是复用逻辑时总会有冲突，所以就只能重写了一个移动棋子的函数专用与服务端或客户端发送数据后，对方移动棋子。







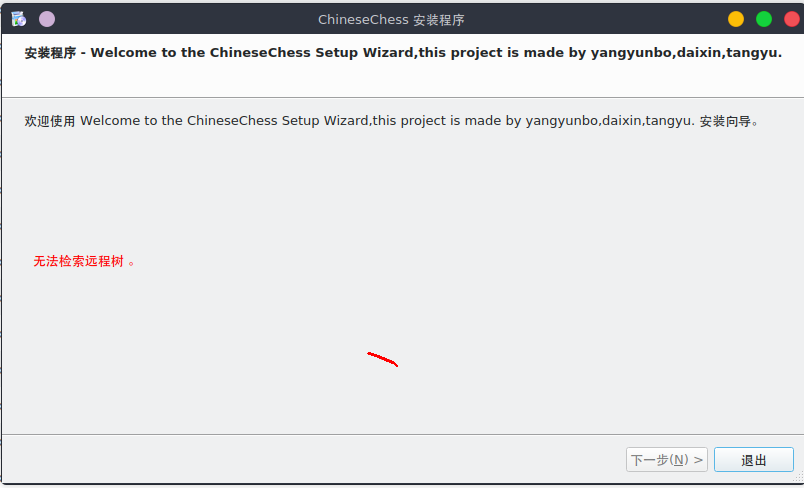




**打包项目：**

1. 首先改变项目的构建配置--Release构建
2. 安装QtInstallerFramework-linux-x64
3. 在安装这里面找到tutorial这个例子，将该例子中的config.xml文件修改成我们项目所需的信息，主要修改了文件中title和Name区域
4. 将release生成的可执行文件和一些头文件等放入tutorial文件夹中的data文件夹中，然后通过一个脚本自动将全部的依赖库拉取到文件夹内
5. 再修改package文件夹中的meta文件夹中的package.xml
6. 最后在Binarycreator所在目录下执行如下命令：./binarycreator -c config/config.xml -p packages ChineseChess,生成ChineseChess可执行文件

可是执行这个文件就无法检索远程树，我也不清楚是啥原因。



**开发参考资料**

**项目打包参考资料：**

<https://doc.qt.io/qtinstallerframework/ifw-tutorial.html>

<https://blog.csdn.net/casun_li/article/details/72821316>

**象棋逻辑参考资料**

**<https://github.com/XMuli/ChineseChess>**