网络对战国际象棋

计83 饶淙元 2017011285

1. 功能效果

1.1 界面展示

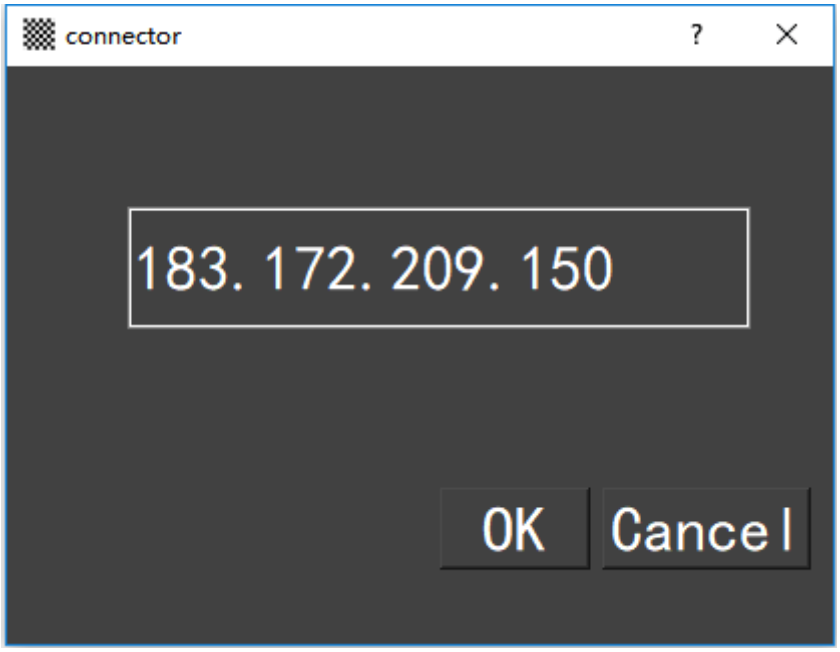
1.1.1 启动器

程序主界面独立存在，可以选择建立主机、连接主机或退出游戏。

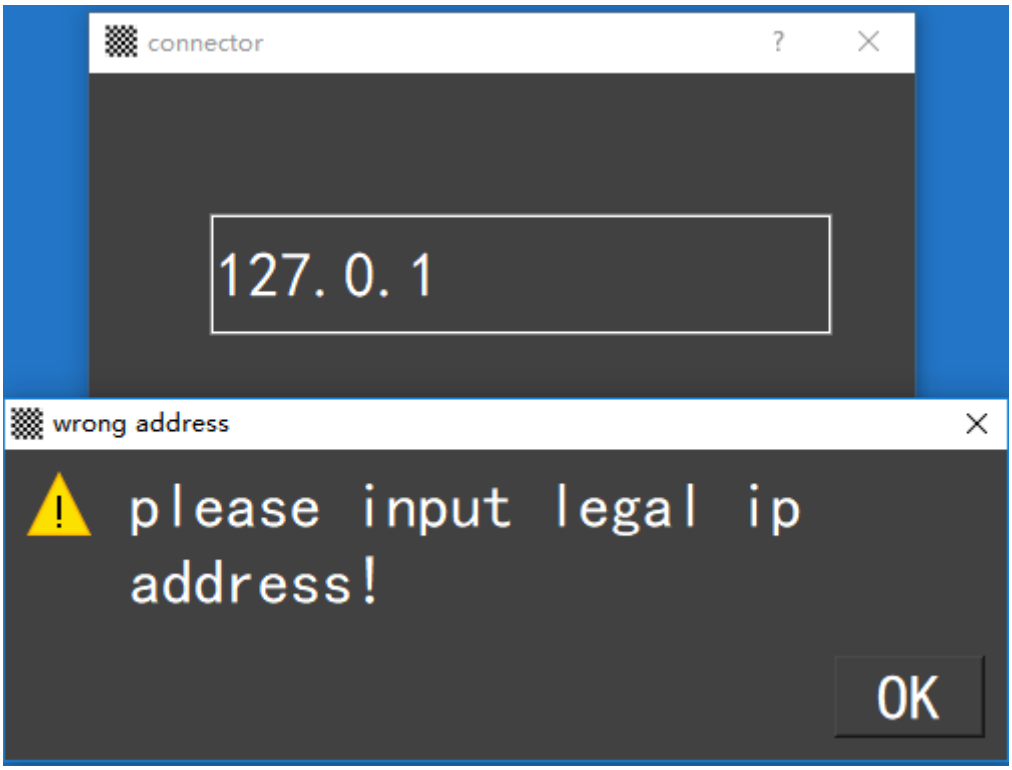
其中标题字是一个`QLabel`，下面三个`QPushButton`，采用布局控制位置，样式表控制字体大小，保证在界面缩放之后还能比较正常地显示。

1.1.2 连接器

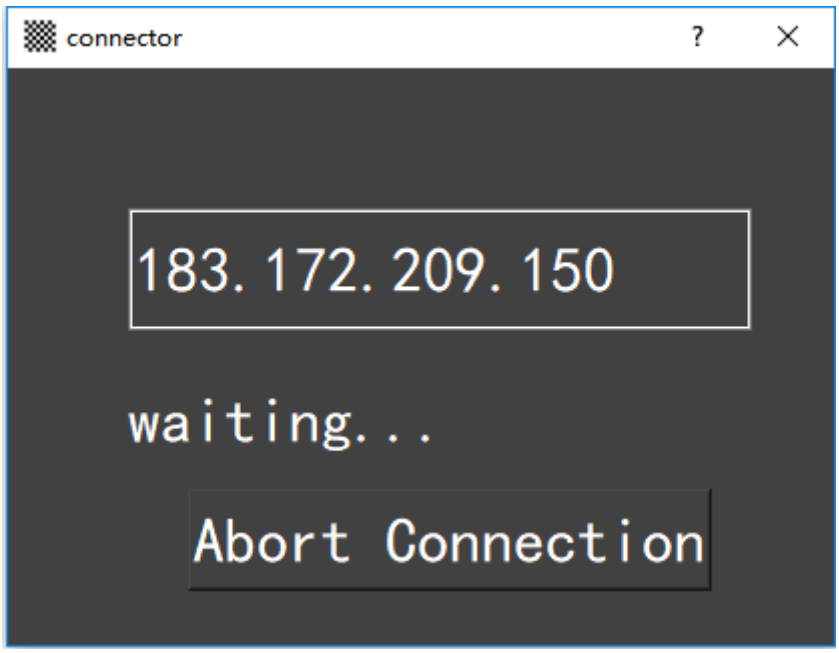
点击建立主机或者连接主机后就会弹出连接器界面，由于本程序的设计理念是服务端和客户端共用一个软件，因此两者的区分不是非常明显。例如此界面对于服务端与客户端完全相同，差别仅在于客户端需要从输入框中输入目标IP，点击OK后尝试连接，服务端这个编辑框是只读里，里面会显示本机可用的IPV4地址(如果存在的话)。



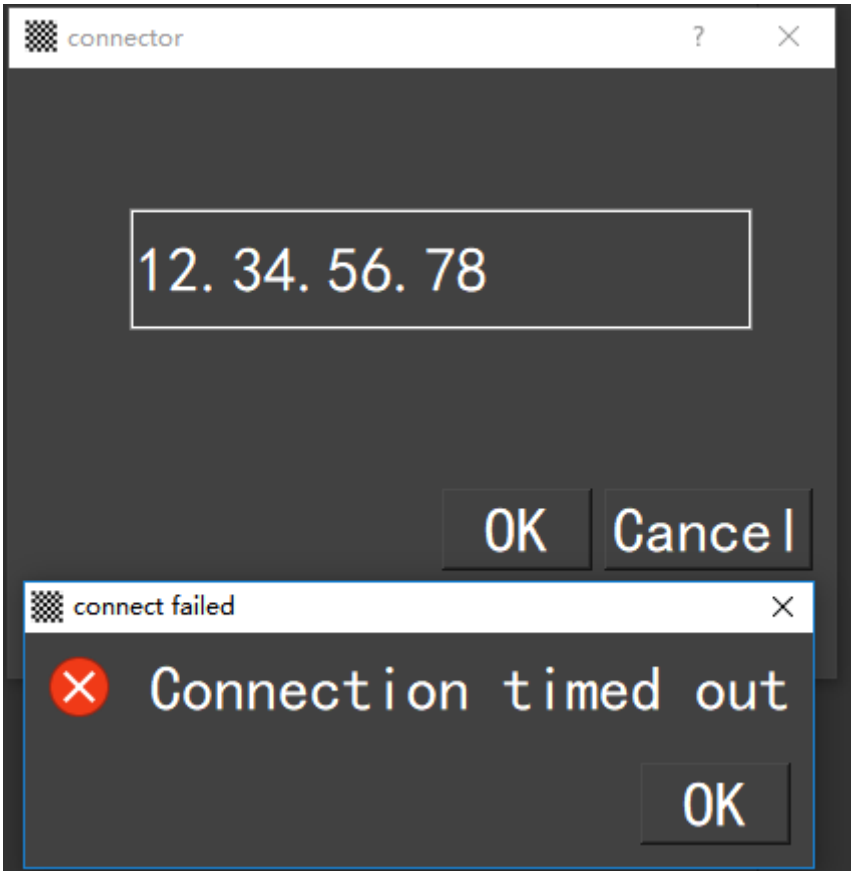
客户端输入的IP地址需要确保合法，不合法的IP将被驳回。



当服务端在等待连接或客户端在尝试连接时，会进入等待状态，此时可以放弃连接返回上一个界面：

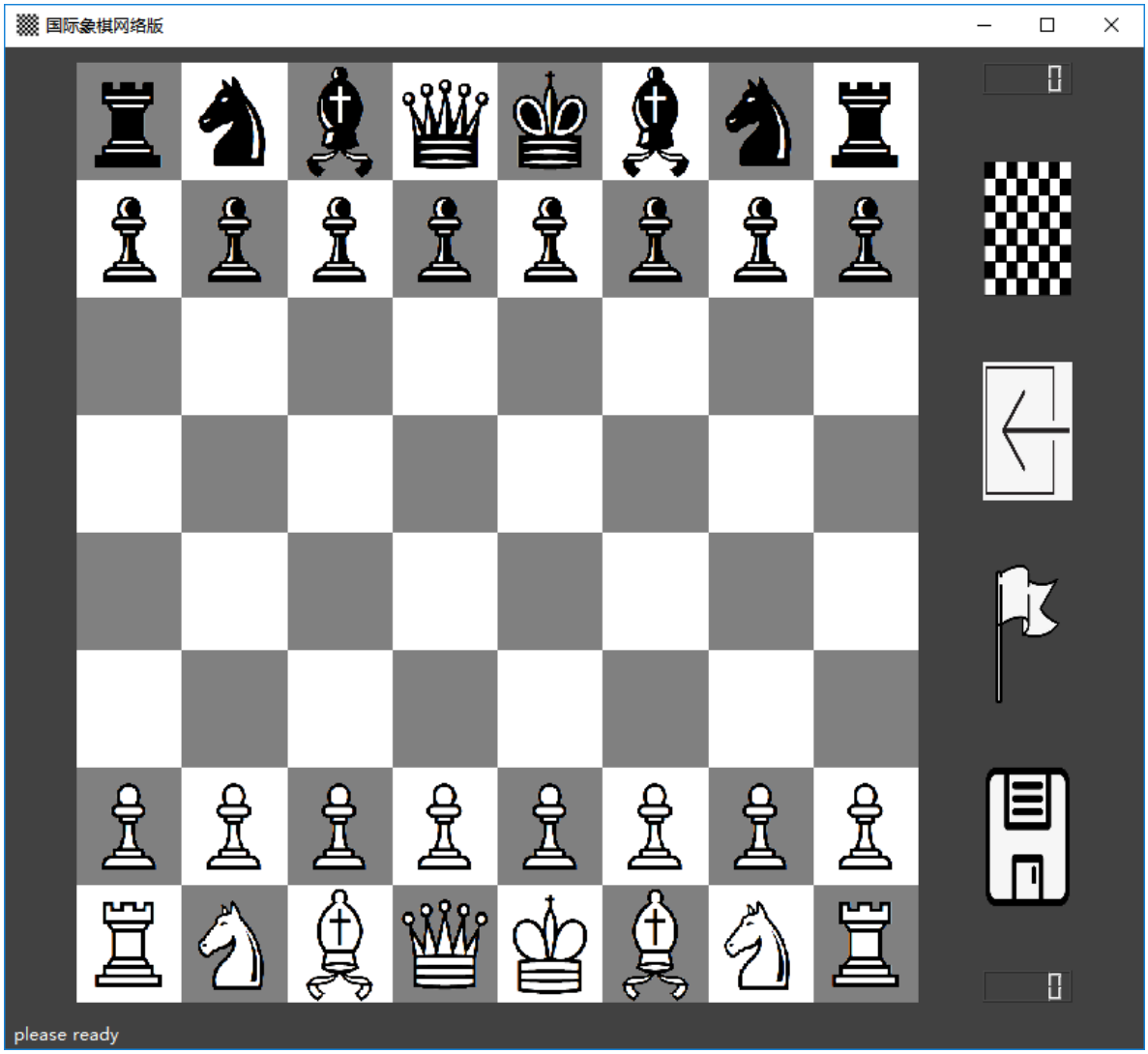


对服务端来说，如果没有客户端前来连接，可以一直等待下去，但是对客户端来说如果被拒绝访问或连接超时等，会得到弹窗报错，例如下图是随便输入一个IP后连接超时（大约45秒），弹窗报错并自动Abort的例子。



1.1.3 游戏界面

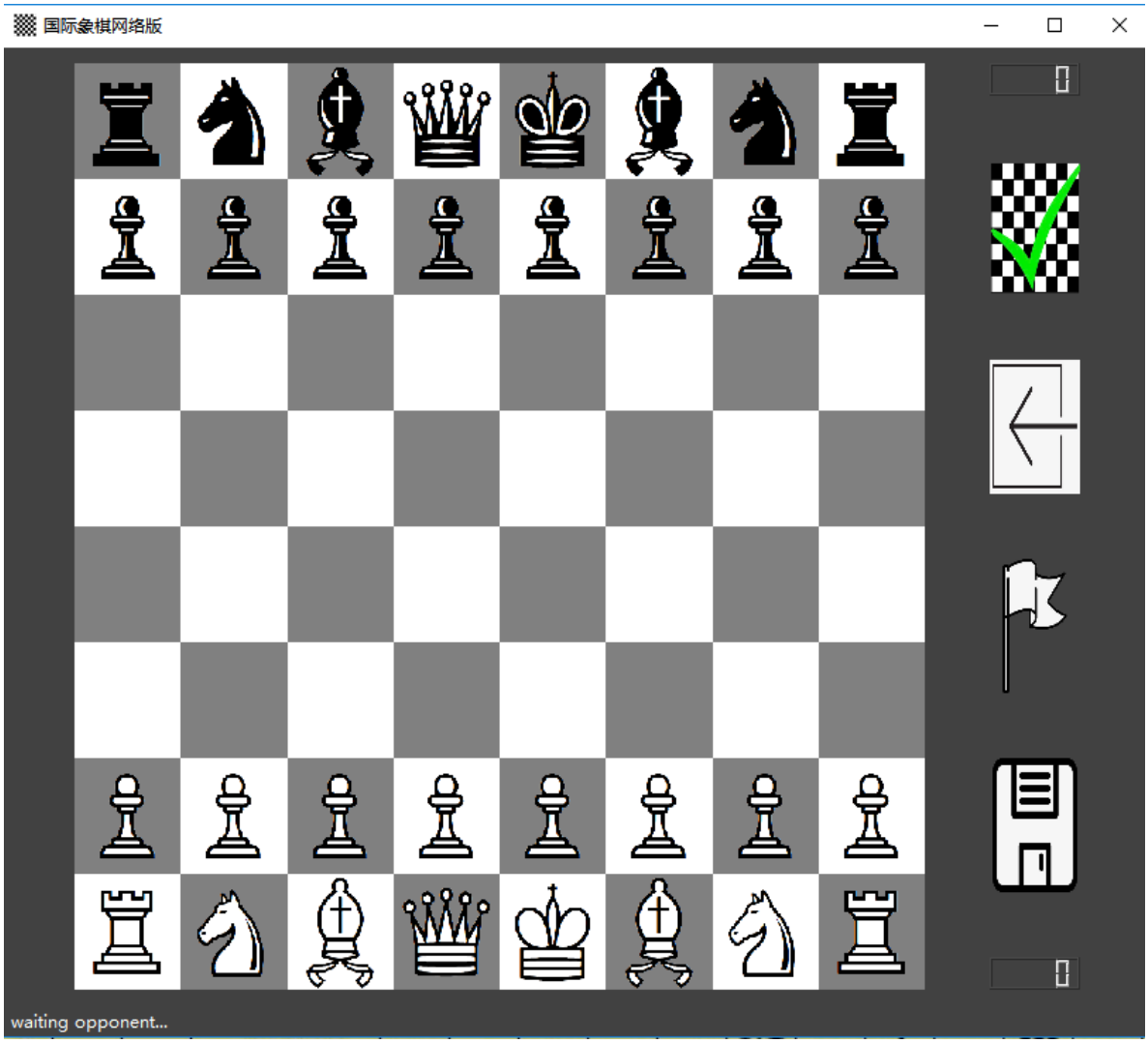
完成连接后，双方都会进入游戏界面，如下是服务端玩家的界面（客户端玩家界面的差异仅在于“载入”按钮为灰色，下文解释）。由于设定里钦点为服务端白棋，客户端黑棋，因此界面上客户端也被设定为黑棋在下，与服务端看到的游戏画面中心对称。



1.2 游戏功能

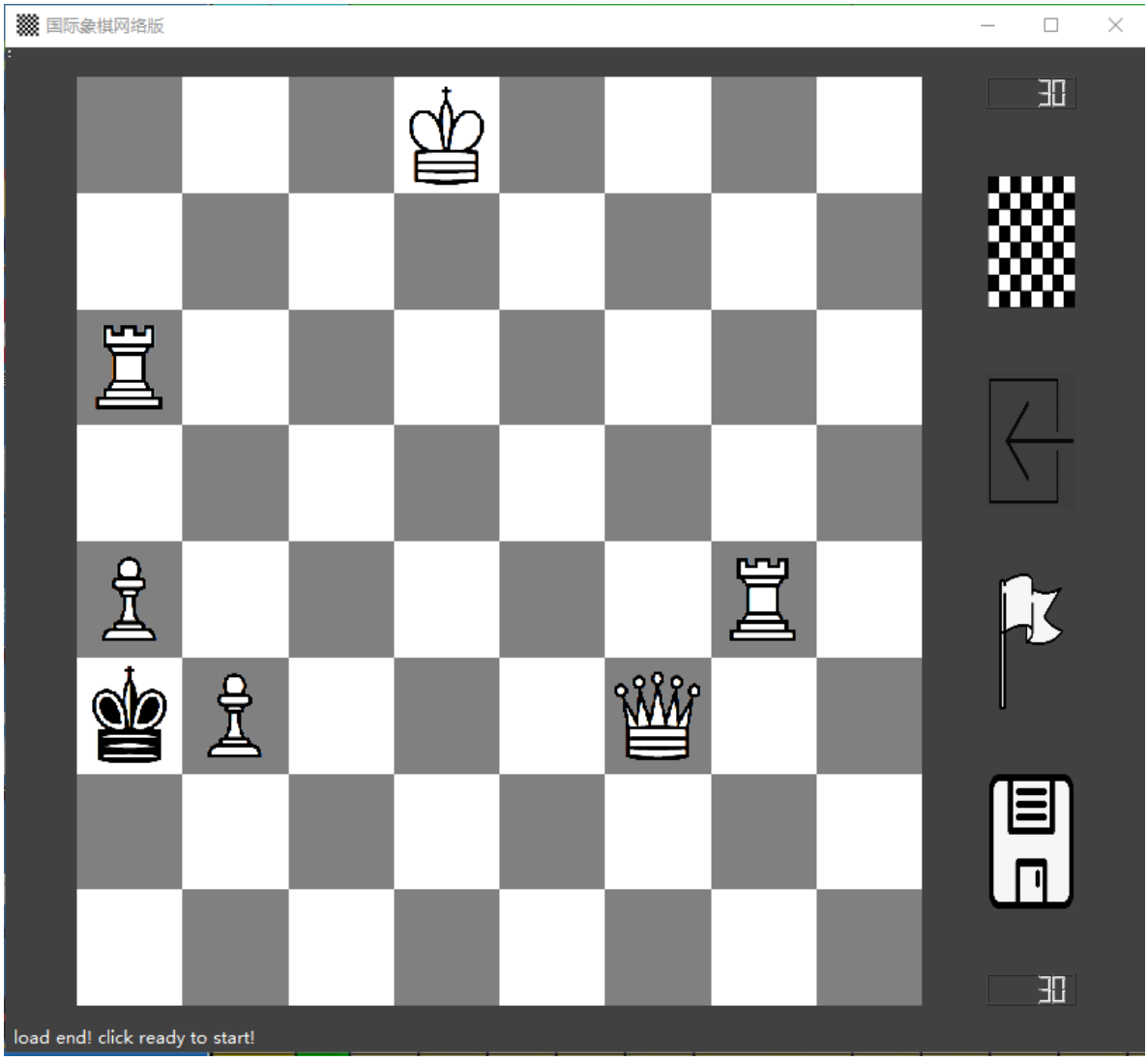
1.2.1 开始游戏

游戏主界面右边有四个图标按钮，分别是“准备/取消准备”、“载入”、“投降”、“保存”。左下角状态栏会提示准备，因为此时还处于未准备状态，双方都准备后才能开始游戏。



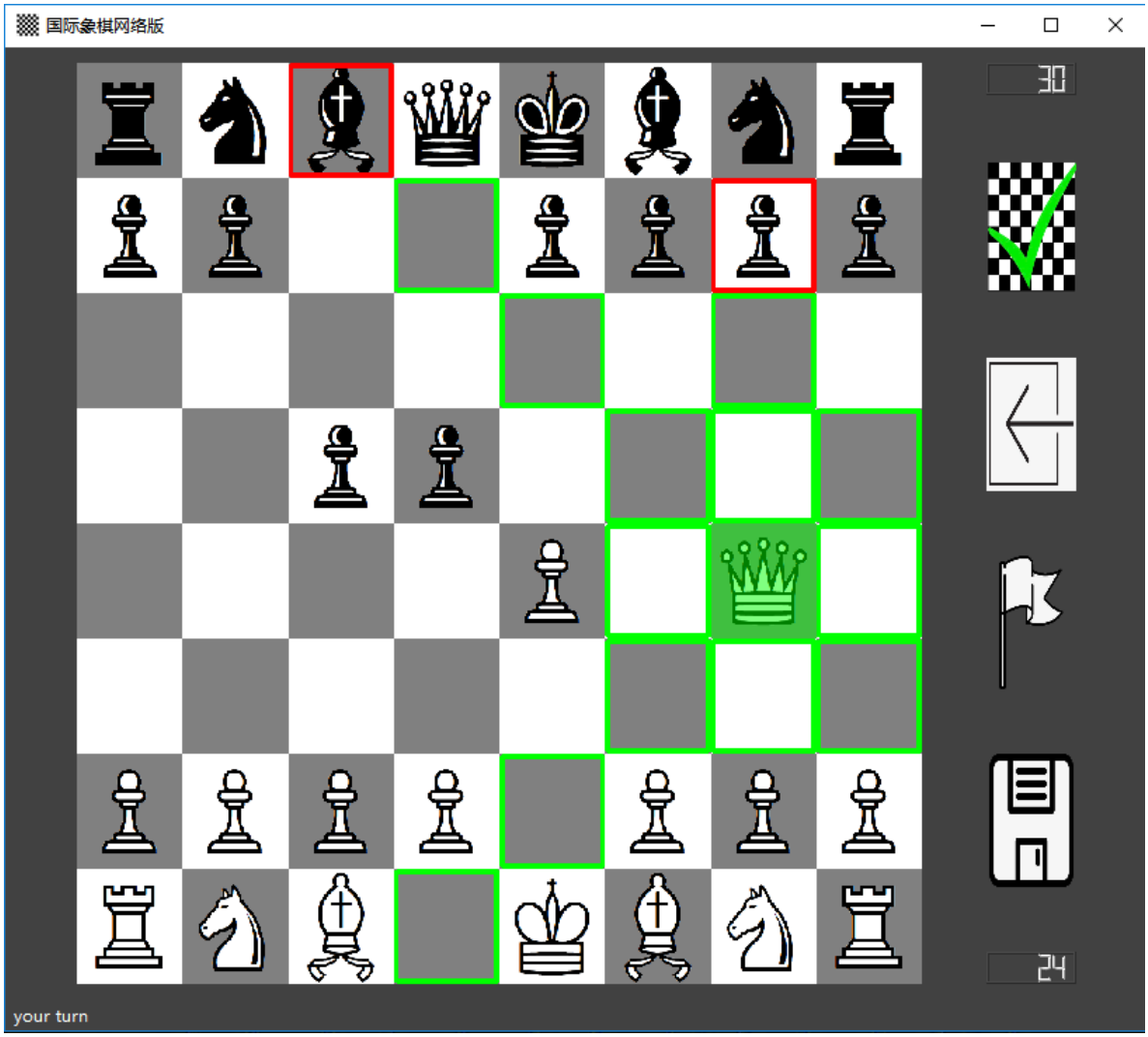
此时点击准备后可以准备按钮会出现一个勾，并且左下角提示等待对方准备，对方准备后倒计时开始，正式进入游戏。

点击载入可以选择残局并加载，此功能只有主机可用，载入残局后双方均需要重新准备游戏。对客户端来说载入残局按钮是灰色不可点状态，如果主机载入残局，客户端之前的准备状态会取消，左下角会有提示。



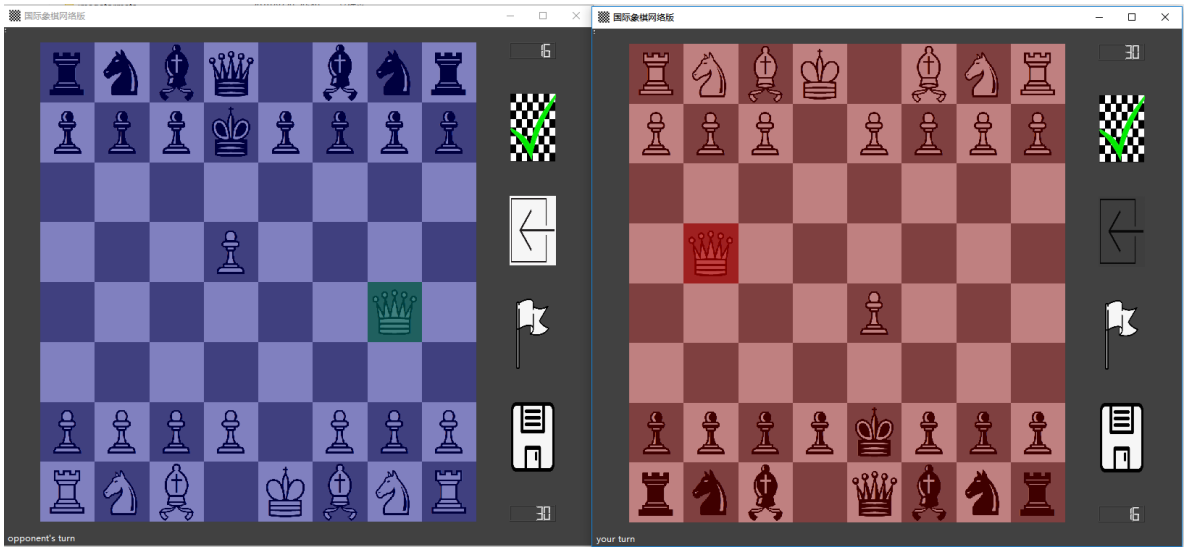
1.2.2 进行游戏

双方均准备后方可以开始游戏，行棋时点击己方棋子，会自动标记可以移动、攻击的区域。

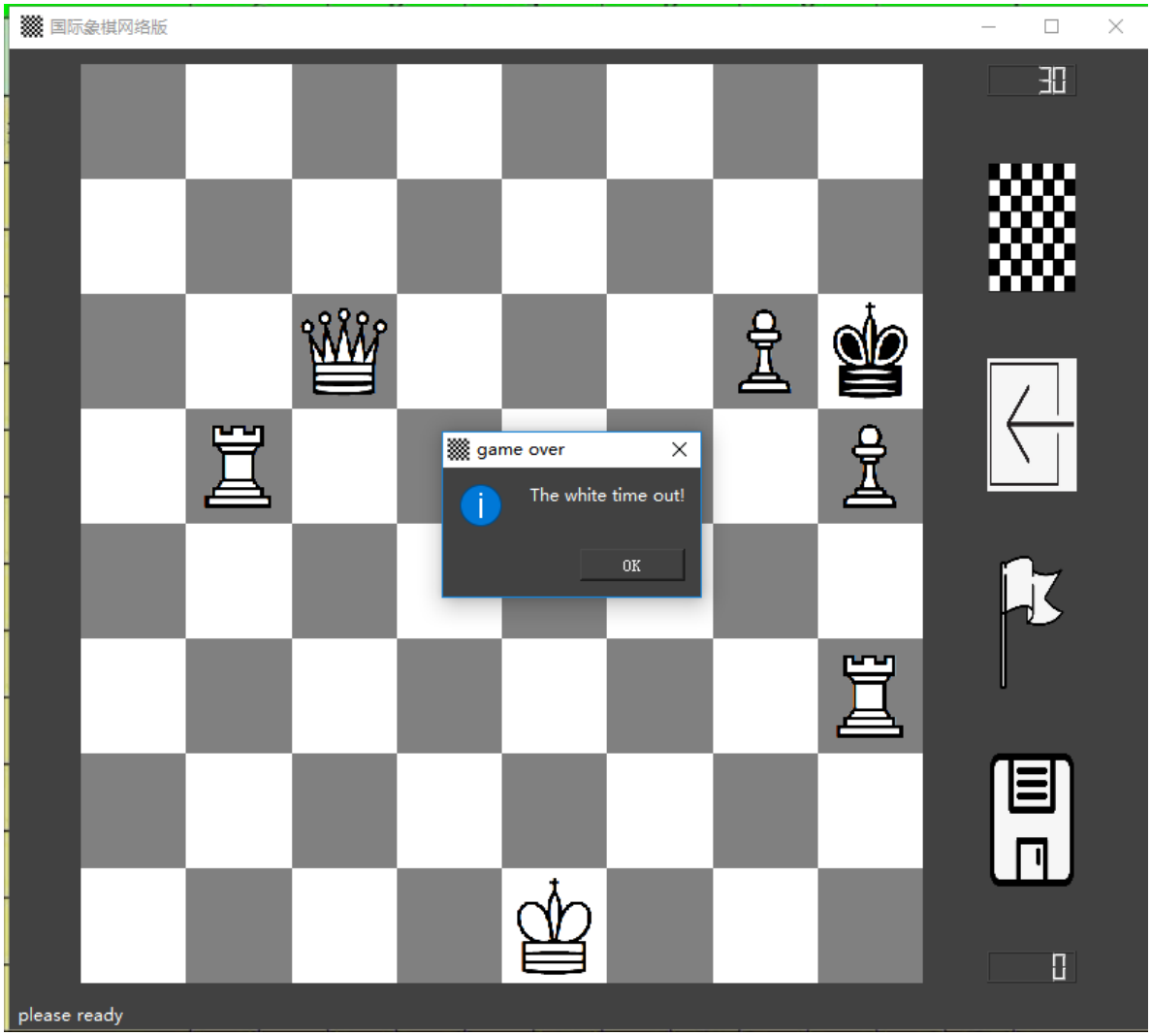


对于特殊的兵升变场面在中间给出额外区域供点击，此时会额外画选择区域，鼠标点击该区域以外的棋子均无效，例如下图是黑棋左上角兵升变的场面。

此外在被将军时会有一层半透明红色做提醒，而将军对方会有半透明蓝色做提醒，下面是将军时双方的场面。



需注意游戏时左下角会提示是谁的回合，并且右下角会显示己方倒计时，右上角显示对方倒计时，每步棋有30秒。如果超时会提示超时并自动结束游戏（如果希望继续游戏，主机可以立刻保存残局、载入残局，双方重新准备，继续游戏）。



1.2.3 结束游戏

游戏结束包括吃掉王、投降、超时、逼和这四种正常情况，另外还有掉线的不正常情况，例如服务端掉线（如断网、强行退出等）时客户端会收到如下提示：

游戏正常结束后战局会暂时保留最后一步行棋后的场面，直到手动点击了准备或者主机载入残局，点击准备后会刷新为标注开局场面，载入残局会自动让双方读取残局。

2. 程序设计

2.1 游戏逻辑

2.1.1 逻辑架构

由于我采用了非常清晰的“逻辑与界面先行，通信后动”的开发顺序，因此逻辑架构较为清晰。

游戏逻辑主要用一个继承自`QWidget`的`Board`类作为棋盘管理器，负责存储棋盘状态，并处理鼠标点击事件。在这里面我准备了一个简单的`Chess`类，仅用于记录一个棋子的种类和颜色，`Board`类有一个`Chess* board[64]`变量管理所有棋子。

玩家点击己方棋子所在区域时时，我首先映射到对应的棋子后，根据该棋子种类试探该棋子所有可能行走的区域，标级给一个Mask后交付绘图函数进行绘制，玩家试图移动该棋子时只需要比对Mask即可判断移动合法性，如果可以移动就在`board`中进行一通指针交换即可。

但是在设计逻辑的时候我已经考虑到了通信的存在，因此鼠标点击移动时不会直接执行，而是由`Board`类发出一个`operated(Instruction ins)`信号，然后`Board`类再设置该信号的槽函数`execute(instruction ins)`进行处理，这为后面通信埋下了伏笔。

在处理玩家移动指令时，如果处理发现没有兵升变且没有结束游戏，那么改变行动方。如果兵升变，那么不改变行动方，仍然是己方行动，然后调用绘图函数在场景中间绘制四种可升变的棋，并且鼠标点击这四个棋以外的区域无效。玩家选择升变目标后，也会作为一个`Insruction`发出信号进行处理。

此外在每次行动完时，除了检查兵升变、吃王之外，还会检查将军、逼和。对于将军，我的处理方法准备一个`bool danger[64]`存储危险区，遍历所有敌方棋子的攻击范围，标记危险区（注：相比可移动区域，危险区将可以攻击到的“自己人”也纳入了区域之中，如果己方王吃了这种位置的敌方棋，会立刻被另一个敌方棋吃掉），检查己方王是否在危险区。我没有采用直接遍历棋子是否能攻击到己方王，这是为后面检测逼和和王车易位做准备。

2.1.2 王车易位

王车易位的内容我写在王的移动区域检测逻辑之中，逻辑中一些固定位置的检测比较简单，关于路径安全的问题，由于为了检测将军我已经存储了危险区域，因此只需要检查王是否在危险区，王到其目标位置的路是否在危险区。

如果符合条件，那么王车易位后王可以到达的位置将会成为可行走区域，因此王车易位没有特别的按钮，只需要将王行走到易位后的位置即可。

从逻辑的执行上来说，王车易位只是王的移动指令，因此执行移动指令时我需要特殊判断移动棋子是否是王，如果是王且移动距离大于1，那么触发了王车易位，将移动方向上的车移到正确的位置。

2.1.3逼和

逼和逻辑比较复杂，我担心在检测时复杂度较高而引起性能瓶颈，一直在考虑简化。鉴于我已经完成了危险区判定、可移动区域检测，我设计了如下一个在我看来比较简单的逼和检测算法：

1. 首先已经获取到了危险区，假定此时已经是逼和场面，然后遍历己方棋子，考察是否能证明未逼和，如果均不能证明，则确实逼和。
2. 将己方棋子分类三类：王；与王在横竖斜(即白皇后移动的方向)上共线的棋，下称内围棋；与王在横竖斜上不共线的棋，下称外围棋。对于不存在可移动区域的棋子无需考虑，一定无法对未逼和做贡献。
3. 对于王，如果王在危险区，则王被将军，证明未逼和；若王不在危险区，遍历王的可移动区域，若存在一格不在危险区，则王可以动，证明未逼和。
4. 对于外围棋，只要有可移动区域，则移动后一定不会导致王被将军，证明未逼和。
5. 对于内围棋，检测在与王共线的位置上前后一格（共两个）是否有可移动区域，如果有，则移动后一定不会导致王被将军，证明未逼和；若无，则在这一条线上一定不存在可移动区域，将它向可移动区域内任意一个区域移动，检测移动后王是否在危险区，若不在则可以移动，证明未逼和；若在，那么它像任何区域移动后都会导致王被将军，不能移动它。
6. 若将上述算法完成后仍然未证明未逼和，那么说明逼和局面形成。

在这个算法下，复杂度主要在于内围棋，最坏情况下需要将所有内围棋都尝试移动一步后检查王是否在危险区，但这是后期才会出现的情况，在前期基本上随便选取一个外围棋，发现他有可移动区域，即可结束判定，不会造成太多消耗。

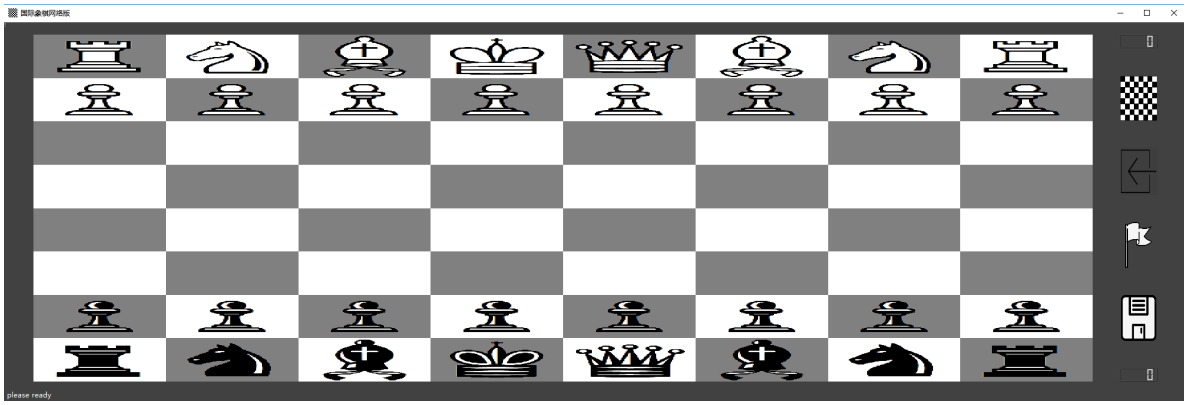
我和室友下了几局，实测这个算法没有让人感到任何延时，并且对于我们测试的情形可以准确地报告逼和。我个人认为这个算法是准确无误的，但是既没有查询相关资料（不会查）也没有给出严格的证明（不会证）。

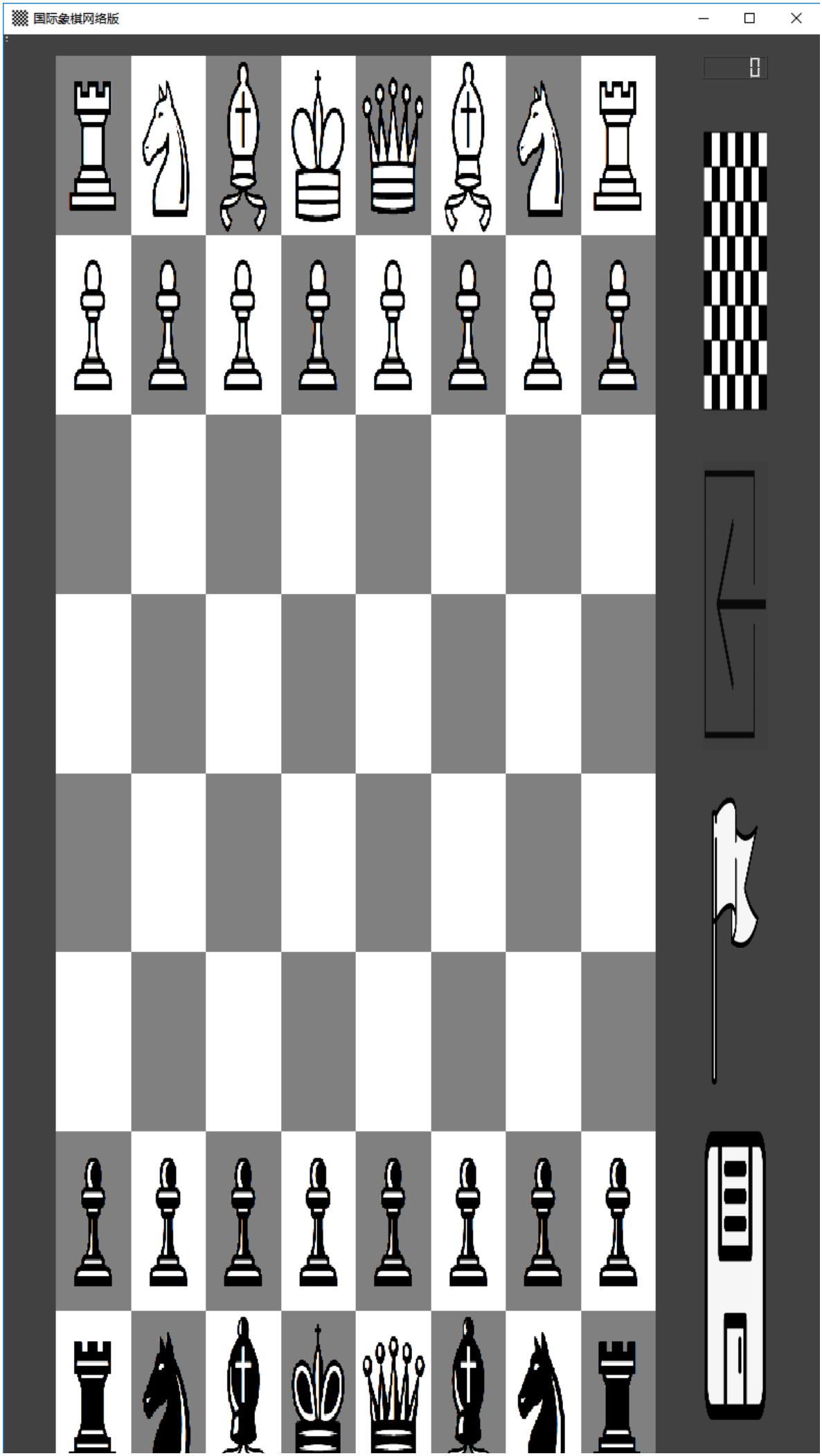
2.2 界面绘图

前文已经提到，棋盘是继承自`QWidget`的`Board`类，而在设计时我为这个类重载了`paintEvent(QPaintEvent*)`和`mousePressEvent(QMouseEvent*)`以完成用户操作和界面显示。

paintEevnt我写得很简单，里面仅仅是一个drawPixmap而已，为了完成绘图，我准备了三个QPixmap，分别是background,image,masked，其中backgroud只画一次，即黑白格棋盘，image是棋盘画了棋子后样子，每次执行指令后更新一次，而masked更新比较频繁，每次点击棋子显示可移动区域，以及将军时填色和兵升变的选择框都会更会masked。

此外为了保证绘图简单而缩放时不必重新从backgroud画起，我在内存里把这三张图的尺寸全部定为 1000×1000，实际画图时会自动填满Board，窗口缩放时Board会随之缩放，进而图片也被缩放，如下图所示：




国际象棋网络版

2.3 通信结构

2.3.1 基本框架

通信我采用的课上样例中给出的QTcpServer和QTcpSocket进行通信，总的来说是简单的C/S模式，一个玩家建立主机后，server开始监听11235端口，另一个玩家试图用QTcpServer连接主机的该端口，连接好后通过QTcpSocket进行长连接通信，直到连接断开。

由于事件驱动已经足以应付需求，因此我没有涉及到线程与进程，完全利用Qt的信号和槽机制完成了相关功能。

2.3.2 工作流程

从流程上讲，主机建立后会监听本机所有IP（以满足同时可以通过127.0.0.1、公网IPV4地址，公网IPV6地址访问）,然后等待被连接，直到收到QTcpServer::newConnection信号后用nextPendingConnection获取到socket，此时用QTcpServer::pauseAccepting函数停止监听，确保不会再被别的客户端连上。

对客户端来说，输入指定IP调用QTcpSocket::connectToHost函数进行连接，并且关注QTcpSocket::SocketError和QTcpSocket::connected信号，如果收到前者，则弹窗报错连接失败，给出错误信息，关闭socket，如果是后者则完成连接，准备接受信号。

再完成连接后，对于服务端和客户端没有什么差别，双边的操作都是关注QTcpSocket::readyRead和QTcpSocket::disconnected信号，一旦收到前者就调用QTcpSocket::readAll函数读取数据并解析，收到后者则提示断开连接，退出游戏界面。

当己方有需要传输的操作时，将信息转化成QByteArray后调用QTcpSocket::write函数将数据传给对方。

如果一方退出游戏，则会调用QTcpSoket::close，这会自动向另一方发出消息，接收方会收到QTcpSocket::disconnect信号，从而进行处理。

2.3.3 通信格式

TCP通信已经有了一套自己的协议，这里不再赘述。但是TCP通信是采用流的模式，一方面传输对象是二进制数据流，无法直接传输具体类型，另一方面还可能出现粘包的问题，因此我需要给出一定的自定义内容解决这些实际问题。

首先是数据转二进制的问题，我将数据分为五类，分别是操作、文件、结局、准备、倒计时，其中操作我转化成了一个int，结局、准备、倒计时本就可以是一个int，唯独文件（残局）需要传输内容有所不同，因此大体上我传输内容的主体分为两种，传int的和传文件的。

对于传int的，我采用双int传法，第一个int存储指令类型（本质上是enum），第二个int就是数据；对于传文件，我在第一个int存储指令类型后，第二个int显示文件长度，然后后面是整个残局文件的数据。

粘包问题最初我没有意识到，直到周三早上做展示时有同学提到后，我当晚就经历了粘包。尽管再上面的机制下我可以直接根据前八个字节的数据判断包长，但是为了让我的通信方法更具有普适性和鲁棒性，我决定在前面再加上一个int记录包长，每次加载数据时直接根据包长进行截断，对剩下的数据递归调用解析接口。

拆包问题我没有考虑，因为我一次传输数据不到256字节，我觉得在现代网络下没理由会出现拆包的情况，若真的出现了可能会出现解析失败的情况。