个人信息及环境

个人信息

杨雅儒，计85，2017011071。

环境

QT Creator 4.9.1(Enterprise)，QT 5.12.4 (MinGW 7.3.0 64-bit)。

程序设计

棋盘设定

界面

位置的表示

按照数学常规定义的二维坐标轴，左下角格子为原点，将a、b、c等字母转成对应1、2、3等数字进行表示。

棋子类型

棋子有三个属性：type、color和pos。其中type如下（程序中有相应常变量如TYPEKING）：

```
type    id
king    1
queen   2
bishop  3
knight  4
rook    5
pawn    6
```

另外color为颜色，黑色为1，白色为0 。

通信协议

通信类的定义

定义Communication类用于通信，其成员变量为 QTcpSocket* tcpSocket，每次实例化Communication时传入一个QTcpSocket *，利用这个 QTcpSocket 进行通信包的传输。

包的定义

定义包格式如下：

```
CHESSPACK YYR
<PACK LENGTH>
<PACK CONTENT>
```

封包和拆包的实现

实现pack和unpack函数用于封包和拆包，封包直接字符串拼接即可，拆包时如果无正常包头或者为非完整的包则返回空串，另外传入了一个引用int pos，用于返回包结尾位置的下一位置索引。

封包代码如下：

```
QString Communication::pack(QString s){
    return packHead + "\n" + QString::number(s.length()) + "\n" + s;
}
```

拆包代码如下：

```
QString Communication::unpack(QString s, int &pos){
    //拆包成功则返回原信息，否则返回空串。pos用于返回字符串s中包结尾的下一位置
    QStringList strList = s.split('\n');
    pos=-1;
    if(strList.length()<3) return "" ;
    bool ok;
    int contentLen=strList.at(1).toInt(&ok),
headLen=strList.at(0).length()+1+strList.at(1).length()+1; //包头和包内容的长度
    if(strList.at(0)!=packHead || !ok) return ""; //不是以包头作为开头
    if(headLen+contentLen>s.length()) return ""; //非完整包
    pos = headLen+contentLen ;
    return s.mid(headLen, contentLen) ;
}
```

而当QTcpSocket接收到信息之后执行如下代码：

```
void Communication::handleRead(){
    mainWindow->debug("READ MESSAGE");
    readBuffer += tcpSocket->readAll() ;
    int pos;
    QString tmp;
    if((tmp=unpack(readBuffer, pos))!=""){
        mainWindow->debug("THE MESSAGE IS A PACK");
        readBuffer.remove(0, pos) ;
        packages.append(tmp) ;
        emit(readyReadPack()) ;
    }
}
```

消息接收机制

在Communication类中实现了类似QTcpSocket的消息接收机制，只不过这里是以一个完整的包内容作为单位的。

实现的函数有：bool hasNextPack(); QString nextPack() ; void close(); 还定义了信号readyReadPack()。

于是在主窗口类中将communication的readyReadPack()信号同this的handleReadPack()连接起来即可。

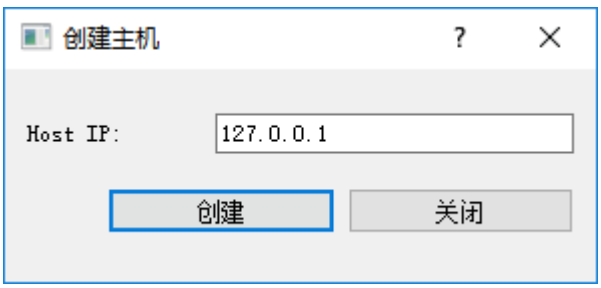
通信逻辑

首先固定端口为12345，存下作为静态变量。

连接过程——服务器端

界面

创建主机界面如下：



默认输入127.0.0.1，可以进行更改，点击“创建主机”即可开始等待连接。

逻辑

新建了DialogCreateHost类，带有成员变量QTcpServer *tcpServer以及QTcpSocket *tcpSocket。

在主界面选择了创建主机之后，DialogCreateHost类实例化的对象dialogCreateHost，并且使用tcpServer监听IP和固定端口，而点击取消连接则会停止监听。

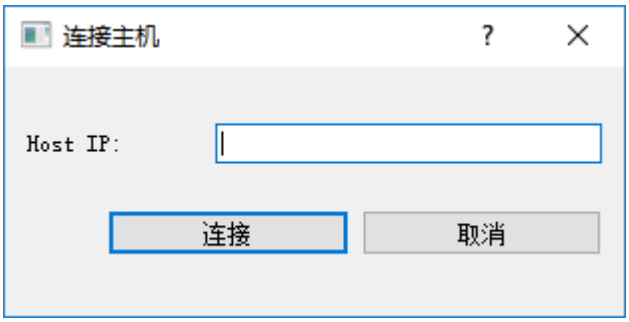
在监听到连接之后便回调主窗口类中的startOnlineGame函数，监听成功的代码如下：

```
tcpSocket = tcpServer->nextPendingConnection();
tcpServer->close();
mainWindow->startOnlineGame(tcpSocket, 0) ; //0表示己方为白方
this->close();
```

连接过程——客户端

界面

连接主机界面如下：



在输入IP地址之后点击“连接主机”即开始尝试连接，连接成功之前点击“取消连接”即可停止连接。

逻辑

新建了DialogConnectToHost类，带有成员变量QTcpSocket *tcpSocket 以及 bool isConnecting。

在点击“连接主机”或“取消连接”这个按钮时，isConnecting进行01切换，并且tcpSocket进行连接或者停止连接。在连接成功后，回调MainWindow的函数 startOnlineGame(tcpSocket, 1)。

点击按钮的代码如下：

```
void DialogConnectToHost::on_pushButtonConnect_clicked()
{
    isConnecting^=1;
    if(isConnecting){
        QString s = ui->lineEdit->text();
        QHostAddress addr(s) ;
        if(!DialogCreateHost::checkIP(s)){
            QMessageBox::critical(this, "错误", "请输入正确ip格式");
            isConnecting=false;
            return;
        }
        tcpSocket->connectToHost(addr, DialogCreateHost::PORT);
        ui->pushButtonConnect->setText("取消连接") ;
    } else{
        tcpSocket->close();
        ui->pushButtonConnect->setText("连接") ;
    }
}
```

连接成功的代码如下：

```
void DialogConnectToHost::handleConnected()
{
    mainWindow->startOnlineGame(tcpSocket, 1) ;
    this->close();
}
```

IP检查

IP检查使用正则表达式进行，这部分代码如下：

```
bool DialogCreateHost::checkIP(QString s)
{
    QRegExp regExp("((\\d{1,2})|(1\\d{2})|(2[0-4]\\d)|(25[0-5])).){3}((\\d{1,2})|(1\\d{2})|
(2[0-4]\\d)|(25[0-5]))");
    return regExp.indexIn(s)!=-1 && regExp.matchedLength()==s.length();
}
```

传输内容

在两端连接上之后，便会在MainWindow类对象中使用communication成员变量进行相互通信。

连接成功时代码如下：

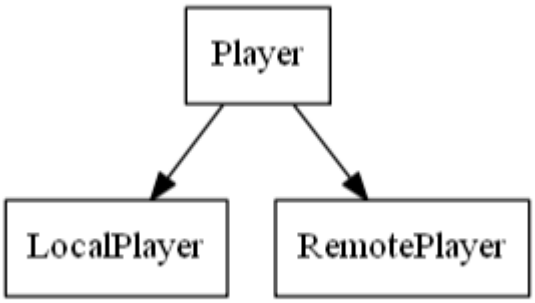
```
void MainWindow::startOnlineGame(QTcpSocket *tcpSocket, int color){
    communication = new Communication(this, tcpSocket, this) ;
    connect(communication, SIGNAL(readyReadPack()), this, SLOT(handleReadPack()));
    isPlayingOnline=true;
    player[color] = localPlayer[color] ;
    remotePlayer->setColor(color^1) ;
    player[color^1] = remotePlayer ;
    if(!color){
        sendMessage(getChessStr()) ;
        player[nowColor]->play();
    } else{
        nowColor=0 ;
        player[nowColor]->play();
    }
}
```

每次传输时直接按照读取残局的格式进行传输，这样会很方便，并且对于残局读取和普通移动以及兵升变等可以一并处理。

另外定义了字符串QString MESSAGEWHITEWIN, MESSAGEBLACKWIN, MESSAGETIE，传输这些特定字符串时表示游戏结束。

玩家类

定义Player作为基类，再定义LocalPlayer和RemotePlayer类，结构如下：



基类

在Player类中，定义了受保护的成员变量颜色color，响应的函数getColor()、setColor(int)，以及虚函数play()和gameEnd(int)，play()表示轮到该玩家下子了，gameEnd(int)表示通知玩家游戏已经结束。另外还定义了MainWindow *mainWindow便于回调。

LocalPlayer类

这个类定义的play()函数和gameEnd(int)函数如下：

```
void LocalPlayer::play(){
    mainWindow->nowChoose = QPoint(-1,-1) ;
    mainWindow->nowColor = color ;
    mainWindow->setStatus(MainWindow::STATUSMYTURN) ;
    mainWindow->checkGameStatus() ;
}

void LocalPlayer::gameEnd(int status){
    ;
}
```

因为是本地玩家，而游戏结束的弹窗是在setStatus函数中进行的，所以在gameEnd函数中并不需要特别做什么事情。

RemotePlayer类

这个类定义的play()函数和gameEnd(int)函数如下：

```
void RemotePlayer::play()
{
    mainWindow->nowChoose = QPoint(-1,-1) ;
    mainWindow->nowColor = color ;
    mainWindow->setStatus(MainWindow::STATUSOPPTURN) ;
}

void RemotePlayer::gameEnd(int status)
{
    if(status==MainWindow::STATUSTIE){
        mainWindow->sendMessage(mainWindow->MESSAGETIE) ;
    } else if(status==MainWindow::STATUSWHITEWIN){
```

```
        mainWindow->sendMessage(mainWindow->MESSAGEWHITEWIN) ;
    } else if(status==MainWindow::STATUSBLACKWIN){
        mainWindow->sendMessage(mainWindow->MESSAGEBLACKWIN) ;
    }
}
```

即游戏结束时需要告知远程玩家。

规则实现

首先使用 QList<Chessman> nowChessman 记录了当前场上的棋子有哪些。

普通移动规则

规则在MainWindow类中实现，通过QList<QPoint> dir[i][j]记录了颜色为i，种类为j的棋子能够移动的方向，canWalkMore[j]记录了这个棋子能否沿这些方向移动多步。

在MainWindow类的构造函数中初始化dir和canWalkMore：

```
memset(canWalkMore,0,sizeof(canWalkMore)) ;
    canWalkMore[2]=canWalkMore[3]=canWalkMore[5]=true;

    //king
    dir[0][1].append(QPoint(-1,0)) ;
    dir[0][1].append(QPoint(1,0)) ;
    dir[0][1].append(QPoint(0,-1)) ;
    dir[0][1].append(QPoint(0,1)) ;
    dir[0][1].append(QPoint(-1,-1)) ;
    dir[0][1].append(QPoint(1,1)) ;
    dir[0][1].append(QPoint(-1,1)) ;
    dir[0][1].append(QPoint(1,-1)) ;

    //queen
    dir[0][2] = dir[0][1] ;

    //bishop
    dir[0][3].append(QPoint(-1,-1)) ;
    dir[0][3].append(QPoint(1,1)) ;
    dir[0][3].append(QPoint(-1,1)) ;
    dir[0][3].append(QPoint(1,-1)) ;

    //knight
    dir[0][4].append(QPoint(-2,1));
    dir[0][4].append(QPoint(2,-1));
    dir[0][4].append(QPoint(-1,2));
    dir[0][4].append(QPoint(1,-2));
    dir[0][4].append(QPoint(2,1));
    dir[0][4].append(QPoint(-2,-1));
    dir[0][4].append(QPoint(1,2));
    dir[0][4].append(QPoint(-1,-2));


    //rook
```

```
    dir[0][5].append(QPoint(-1,0)) ;
    dir[0][5].append(QPoint(1,0)) ;
    dir[0][5].append(QPoint(0,-1)) ;
    dir[0][5].append(QPoint(0,1)) ;

    //pawn
    dir[0][6].append(QPoint(0,1)) ;

    //黑方的棋子能走的dir（实际上只有pawn同白方有区别）
    for(int j=1;j<=TYPENUM;++j){
        for(int k=0;k<dir[0][j].length();++k){
            QPoint p = dir[0][j].at(k) ;
            p.setY(-p.y());
            dir[1][j].append(p) ;
        }
    }
```

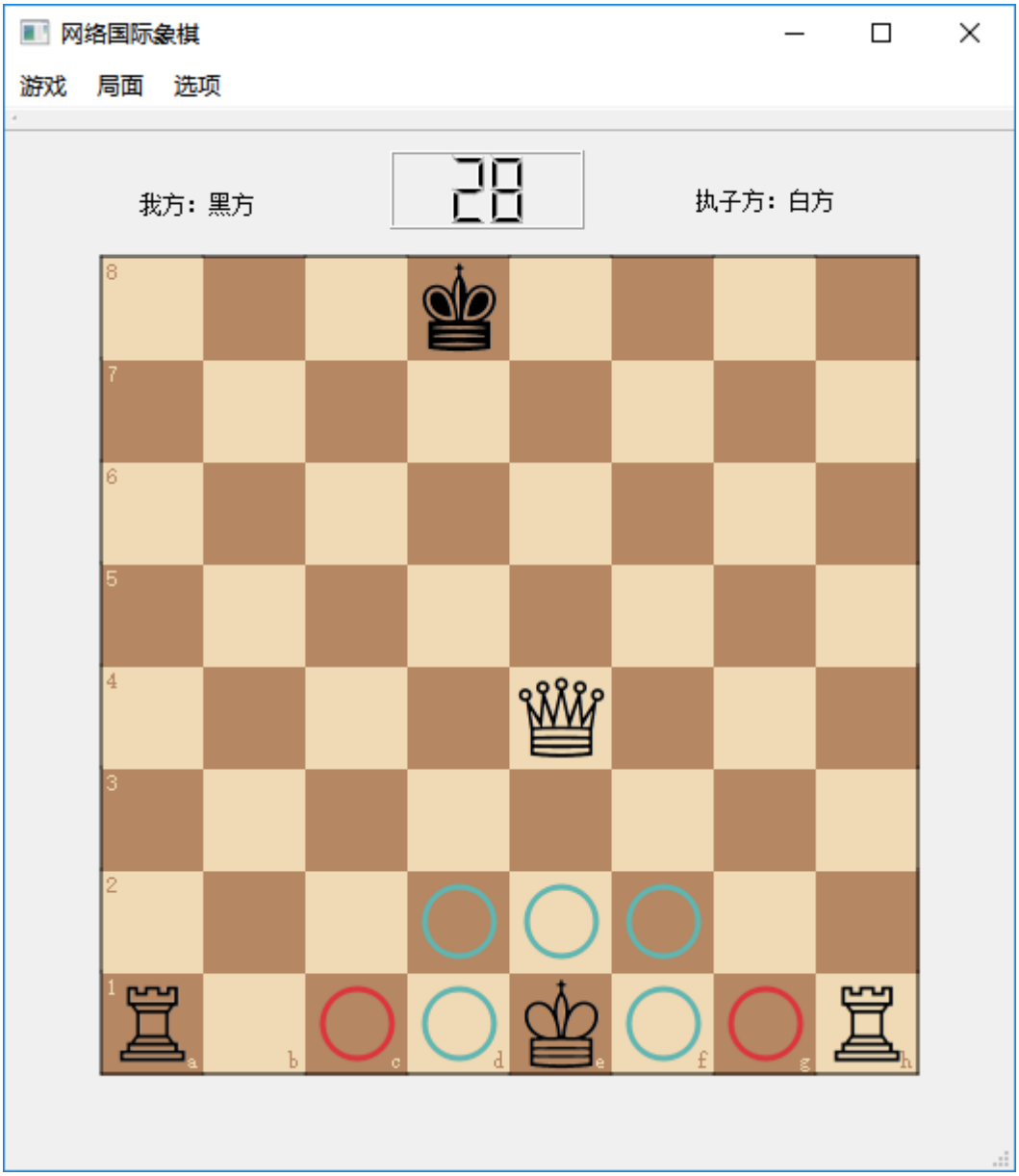
这样一来，在之后枚举移动规则时便会非常方便。

选中和移动

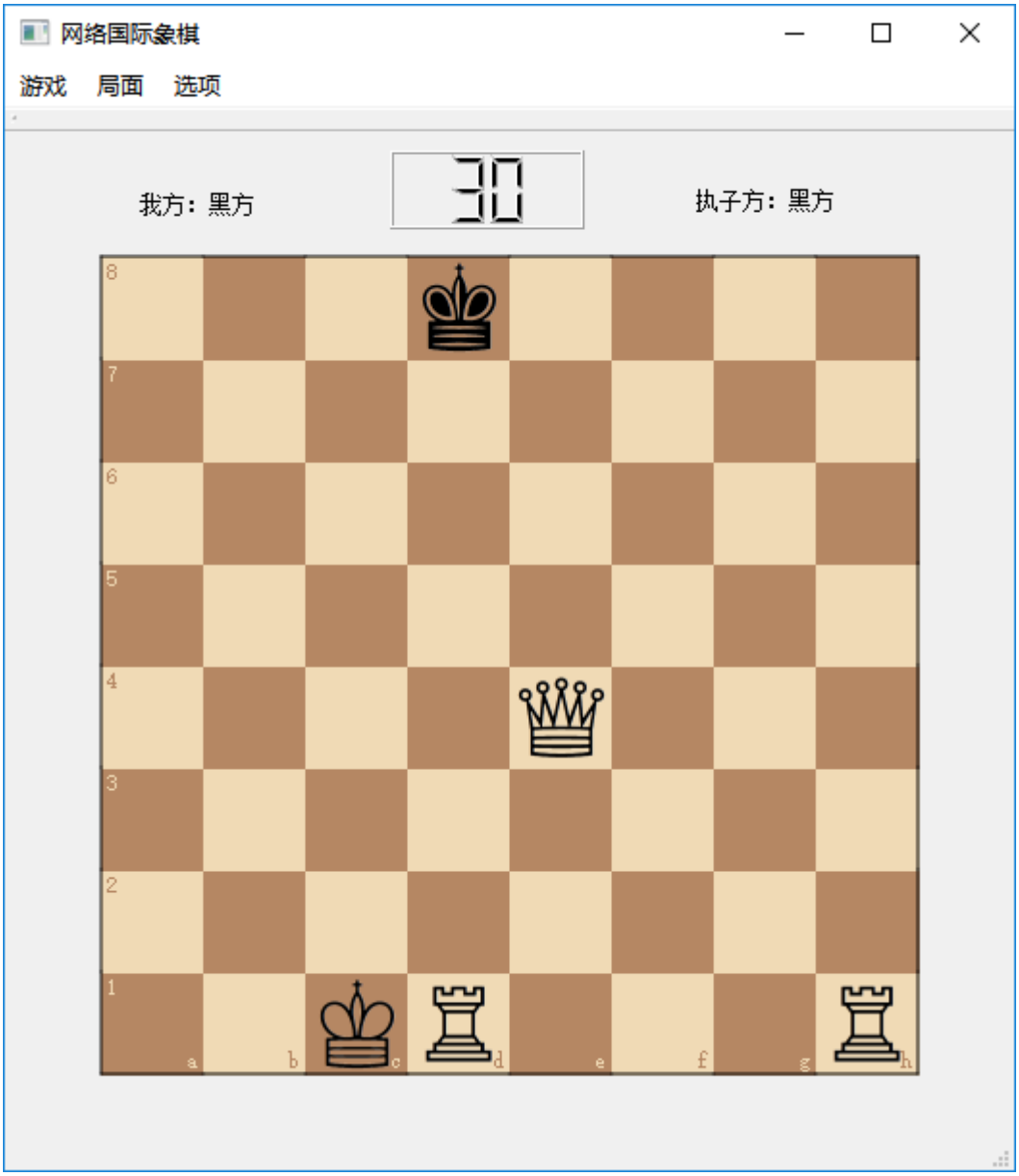
当轮到自己的回合时，鼠标点击一个棋子便会将其能走的位置用圈标记出来，注意一般来说棋子不能跨越其它棋子移动。这一部分在mousePressEvent中实现，效果如下：

再次点击带圈的格子便可以进行移动。另外这一部分还特殊处理了王车易位和兵升变。

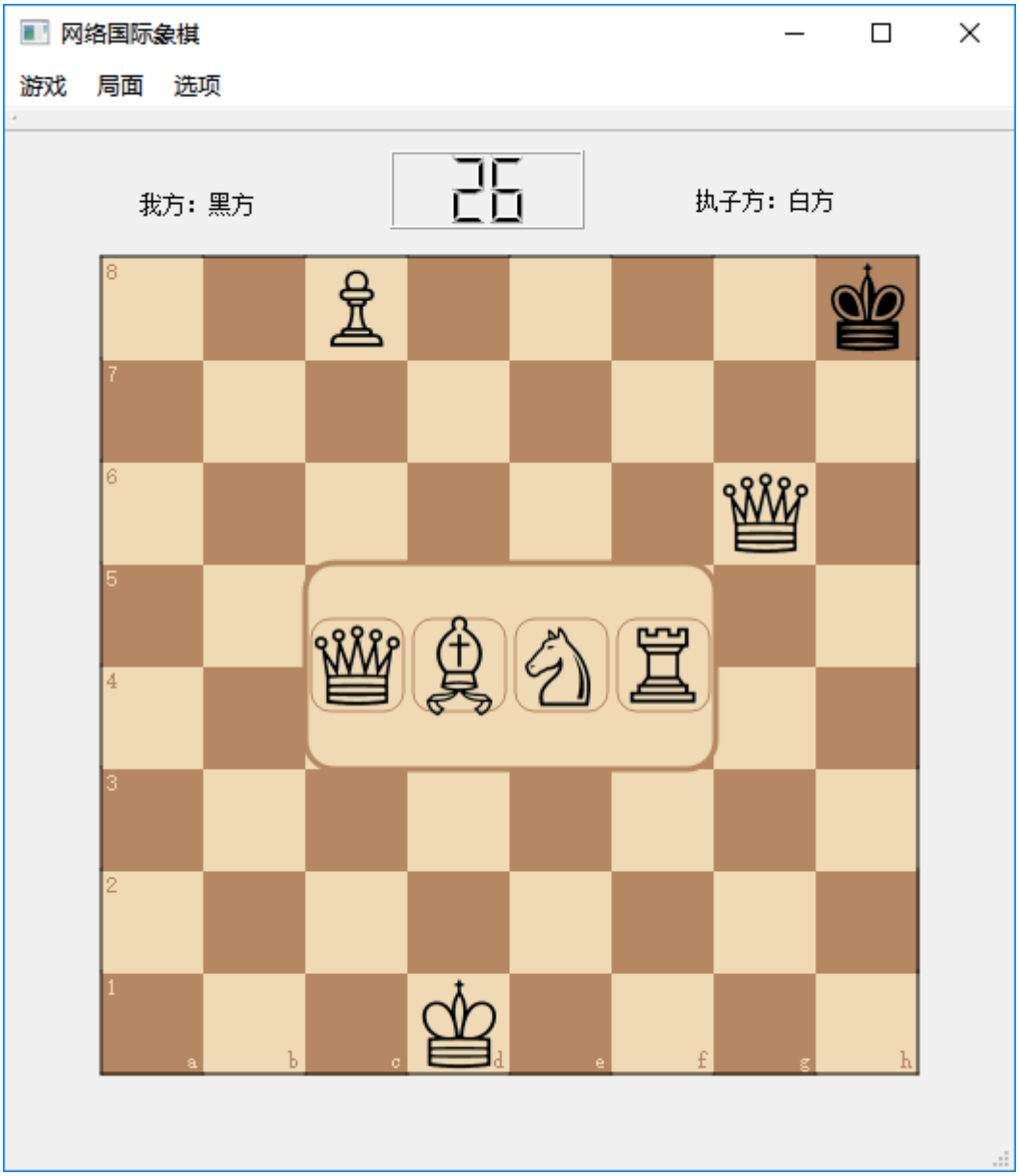
王车易位如下：



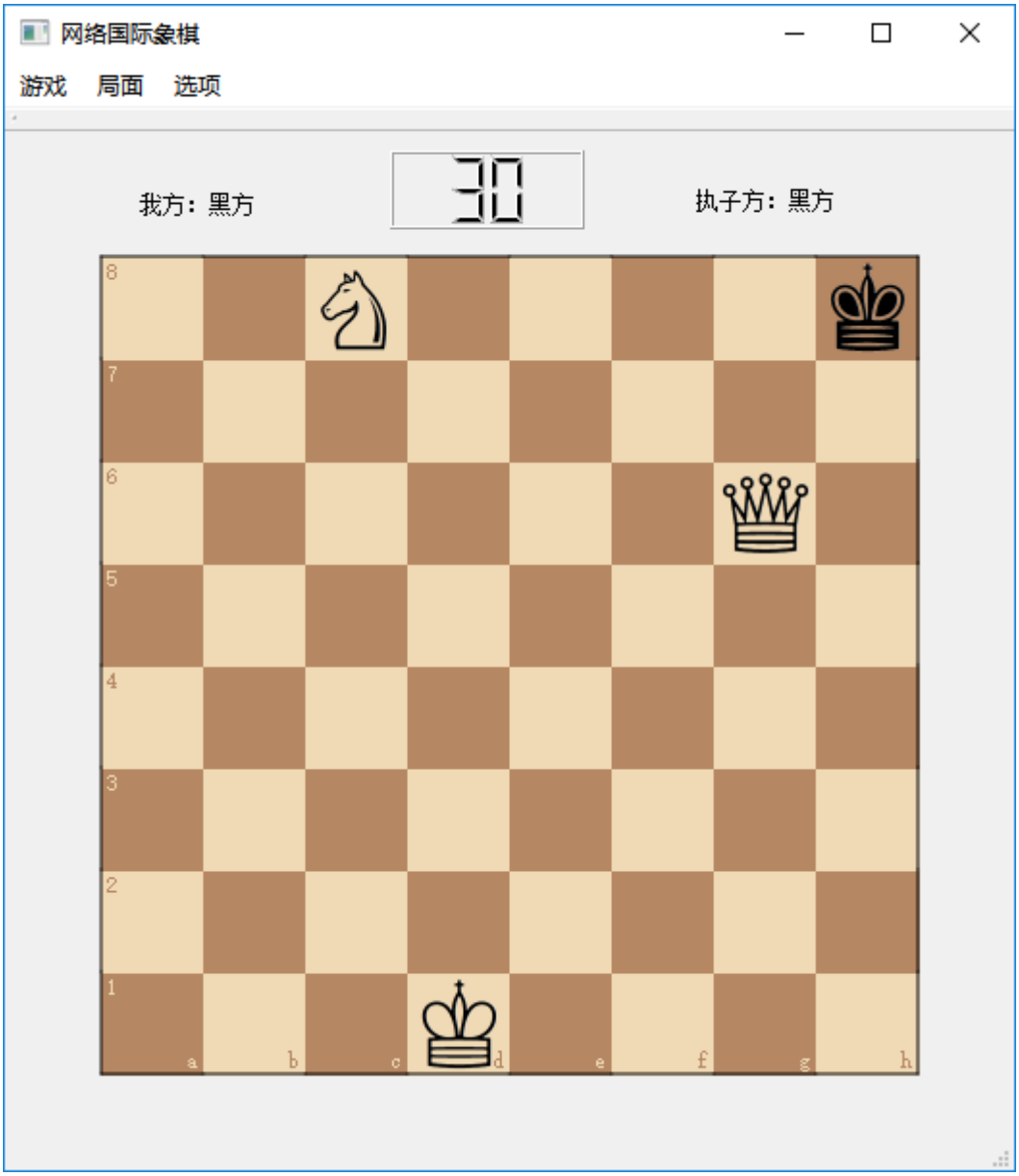
其中点击红色圈表示进行王车易位，例如点击左边的红圈：



兵升变如下：



例如变成马：



这一部分代码较长，在MainWindow类的mousePressEvent函数中实现，可以在最后附的代码中找到。

另外在确定候选可行位置时，实现了代码QList<QPoint> MainWindow::getCandidatePos(Chessman man)和QList<QPoint> MainWindow::getCandidatePosWithCheck(Chessman man)，分别可以求出man棋子在不带将军检查和带将军检查情况下的移动候选位置。

其中前者代码如下：

```
QList<QPoint> MainWindow::getCandidatePos(Chessman man){
    //计算棋子下一步能走的所有位置，注意这里是不考虑将军的情况下合法能走到的位置
    QList<QPoint> list;
    int type = man.type, color = man.color;
    QPoint pos = man.pos ;
    for(int i=0;i<dir[color][type].length();++i){
        QPoint d = dir[color][type].at(i) ;
        int up = (canWalkMore[type] ? 7 : 1) ;
        for(int j=1;j<=up;++j){
            QPoint newPos = pos + d*j ;

            if(outGridRange(newPos)) break ;
```

```
            int tmpInd = getChessmanIndOnPos(newPos) ;
            if(tmpInd==-1 || nowChessman.at(tmpInd).color!=color){ //可移动
                if(type==TYPEPAWN && tmpInd!=-1) //pawn不可直接吃
                    continue;
                list.append(newPos) ;
            }
            if(tmpInd!=-1) break ; //被遮挡
        }
    }
    if(type==TYPEPAWN){ //特殊处理pawn
        QPoint d = dir[color][type].at(0) , newPos = pos+2*d;
        if(getPawnStatus(man)==PAWNINI && getChessmanIndOnPos(pos+d)==-1 &&
getChessmanIndOnPos(newPos)==-1 && !outGridRange(newPos)){
            list.append(newPos);
        }
        //吃子
        newPos = pos+d+QPoint(1,0);
        if(getChessmanIndOnPos(newPos)!=-1 &&
nowChessman.at(getChessmanIndOnPos(newPos)).color!=color){
            list.append(newPos);
        }
        newPos = pos+d+QPoint(-1,0);
        if(getChessmanIndOnPos(newPos)!=-1 &&
nowChessman.at(getChessmanIndOnPos(newPos)).color!=color){
            list.append(newPos);
        }
    }

    return list;
}
```

后者代码如下：

```
QList<QPoint> MainWindow::getCandidatePosWithCheck(Chessman man){
    //计算棋子下一步能走的所有位置，考虑不能被将军
    QList<QPoint> list = getCandidatePos(man) ;
    int ind = nowChessman.indexOf(man) ;
    QList<QPoint>::iterator it = list.begin();
    while(it!=list.end()){
        QPoint newPos = *it;
        QList<Chessman> tmpNowChessman = nowChessman; //备份nowChessman
        moveChessman(ind, newPos) ;
        int ck = isCheck() ;
        nowChessman = tmpNowChessman ;
        if(ck & (man.color ? CHECKBLACK : CHECKWHITE)){
            it = list.erase(it) ;
        } else {
            ++ it ;
        }
    }


    //王车易位
```

```
    if(man.type==TYPEKING){
        int y = (man.color ? 8 : 1);
        if(man.pos == QPoint(5,y) && !(isCheck() & (man.color ? CHECKBLACK : CHECKWHITE))){
            bool canLongCastling=false, canShortCastling=false;
            Chessman rook1, rook2 ;
            for(int i=0;i<nowChessman.length();++i){
                Chessman rook = nowChessman.at(i);
                if(rook.color==man.color && rook.type==TYPEROOK){
                    if(rook.pos == QPoint(1,y)){
                        canLongCastling=true;
                        rook1 = rook;
                    }
                    else if(rook.pos == QPoint(8,y)){
                        canShortCastling=true;
                        rook2 = rook;
                    }
                }
            }
            if(getChessmanIndOnPos(QPoint(4,y))!=-1 || getChessmanIndOnPos(QPoint(3,y))!=-1)
canLongCastling=false;
            if(getChessmanIndOnPos(QPoint(6,y))!=-1 || getChessmanIndOnPos(QPoint(7,y))!=-1)
canShortCastling=false;
            if(canLongCastling){
                QList<Chessman> tmpNowChessman = nowChessman ;
                moveChessman(ind, QPoint(4,y)) ;
                if(isCheck() & (man.color ? CHECKBLACK : CHECKWHITE)){
                    canLongCastling=false;
                }
                moveChessman(ind, QPoint(3,y)) ;
                moveChessman(nowChessman.indexOf(rook1), QPoint(4,y)) ;
                if(isCheck() & (man.color ? CHECKBLACK : CHECKWHITE)){
                    canLongCastling=false;
                }
                nowChessman = tmpNowChessman ;
                if(canLongCastling){
                    list.append(QPoint(3,y));
                }
            }

            if(canShortCastling){
                QList<Chessman> tmpNowChessman = nowChessman ;
                moveChessman(ind, QPoint(6,y)) ;
                if(isCheck() & (man.color ? CHECKBLACK : CHECKWHITE)){
                    canShortCastling=false;
                }
                moveChessman(ind, QPoint(7,y)) ;
                moveChessman(nowChessman.indexOf(rook2), QPoint(6,y)) ;
                if(isCheck() & (man.color ? CHECKBLACK : CHECKWHITE)){
                    canShortCastling=false;
                }
                nowChessman = tmpNowChessman ;
                if(canShortCastling){

                    list.append(QPoint(7,y));
```

```
                }
            }
        }
    }

    return list;
}
```

将军、将杀和逼和

用CHECKNEITHER=0, CHECKWHITE=1, CHECKBLACK=2, CHECKBOTH表示哪一方出现了这样的情况。

判断将军直接判断是否有一方棋子可以一步吃掉另一方王即可，代码如下：

```
int MainWindow::isCheck(){
    //判断哪个颜色被将军，0为没有被将军，1为白色被将军，2为黑色被将军，3为同时被将军，用常变量表示
    int ret=0;
    QPoint kingPos[2];
    for(int i=0;i<nowChessman.length();++i){
        Chessman man = nowChessman.at(i) ;
        if(man.type==TYPEKING) kingPos[man.color]=man.pos;
    }
    for(int i=0;i<nowChessman.length();++i){
        //枚举走哪个棋子
        Chessman man = nowChessman.at(i) ;
        QList<QPoint> list = getCandidatePos(man) ;
        if(list.indexOf(kingPos[man.color^1])!=-1){
            ret |= (man.color ? CHECKWHITE : CHECKBLACK) ;
        }
    }
    return ret;
}
```

再写一个函数isStuck()判断哪方无法移动棋子，实现如下：

```
int MainWindow::isStuck(){
    //判断哪个颜色无法走子，0为没有，1为白色，2为黑色，3为两者用和isCheck一样的常变量进行表示
    int ret=3 ;
    for(int i=0;i<nowChessman.length();++i){
        Chessman man = nowChessman.at(i) ;
        QList<QPoint> list = getCandidatePosWithCheck(man) ;
        if(list.length()>0) {
            ret &= (man.color ? (~CHECKBLACK) : (~CHECKWHITE)) ;
        }
    }
    return ret;
}
```

于是判断将杀就很容易了：

```
int MainWindow::isCheckMate(){
    //判断哪个颜色被将杀，0为没有被将杀，1为白色被将杀，2为黑色被将杀，用和isCheck一样的常变量表示
    return (isCheck()&isStuck()) ;    //正被将军且无法移动的一方被将杀
}
```

判断逼和也很容易：

```
int MainWindow::isStaleMate(){
    //判断哪个颜色被逼和，0为没有被逼和，1为白色被逼和，2为黑色被逼和，用和isCheck一样的常变量表示
    return ((~isCheck())&isStuck()) ; //未被将军且无法移动的一方被逼和
}
```

附主要代码

这里只列出主要的.cpp文件代码。

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "player.h"
#include "localplayer.h"
#include "remoteplayer.h"
#include "communication.h"
#include <QPainter>
#include <QFile>
#include <QDebug>
#include <QColor>
#include <QMessageBox>
#include <QFileDialog>
#include <cmath>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    timeLim=timeRes=30; //30秒时间限制
    gridSize=53;
    tagSize=gridSize/8 ;
    circleR=gridSize/3;
    col=row=8;
    leftUp = QPoint(50,100) ;
    upgradeLeftUp=getPoint(3,6) ;
    groundColor[0] = QColor(240,218,181) ;
    groundColor[1] = QColor(181,135,99) ;
    circleColor = QColor(99,181,176);
    castlingColor = QColor(219,54,62) ;
    iniChessmanStr = QString("white\nking 1 e1\nqueen 1 d1\nbishop 2 c1 f1\nknight 2 b1 g1\nrook
```

```
2 a1 h1\npawn 8 a2 b2 c2 d2 e2 f2 g2 h2\nblack\nking 1 e8\nqueen 1 d8\nbishop 2 c8 f8\nknight 2
b8 g8\nrook 2 a8 h8\npawn 8 a7 b7 c7 d7 e7 f7 g7 h7") ;

    textBrowser = new QTextBrowser(this) ;
    textBrowser->setGeometry(QRect(getPoint(9,9)+QPoint(30,0),getPoint(9,1)+QPoint(300,0))) ;
    textBrowser->hide();
    ui->actionDebug->setChecked(true) ;
    on_actionDebug_triggered();
    MESSAGEWHITEWIN = "WHITE WIN" ;
    MESSAGEBLACKWIN = "BLACK WIN" ;
    MESSAGETIE = "TIE" ;
    isPlayingOnline=false;
    on_actionLoadInit_triggered();
    upgradingInd=-1;
    for(int i=0;i<MAXM;++i) label[i] = new QLabel(this) ;
    for(int i=0;i<4;++i) upgradeLabel[i] = new QLabel(this);
    nowChoose = QPoint(-1,-1);
    ui->lcdNumber->setDigitCount(2);
    ui->lcdNumber->display(timeRes) ;
    playTimer = new QTimer(this) ;
    connect(playTimer , SIGNAL(timeout()), this, SLOT(passOneSec())) ;
    localPlayer[0] = new LocalPlayer(this, 0) ;
    localPlayer[1] = new LocalPlayer(this, 1) ;
    remotePlayer = new RemotePlayer(this, 0);
    setStatus(STATUSNOTRUN) ;

    memset(canWalkMore,0,sizeof(canWalkMore)) ;
    canWalkMore[2]=canWalkMore[3]=canWalkMore[5]=true;

    //king
    dir[0][1].append(QPoint(-1,0)) ;
    dir[0][1].append(QPoint(1,0)) ;
    dir[0][1].append(QPoint(0,-1)) ;
    dir[0][1].append(QPoint(0,1)) ;
    dir[0][1].append(QPoint(-1,-1)) ;
    dir[0][1].append(QPoint(1,1)) ;
    dir[0][1].append(QPoint(-1,1)) ;
    dir[0][1].append(QPoint(1,-1)) ;

    //queen
    dir[0][2] = dir[0][1] ;

    //bishop
    dir[0][3].append(QPoint(-1,-1)) ;
    dir[0][3].append(QPoint(1,1)) ;
    dir[0][3].append(QPoint(-1,1)) ;
    dir[0][3].append(QPoint(1,-1)) ;

    //knight
    dir[0][4].append(QPoint(-2,1));
    dir[0][4].append(QPoint(2,-1));
    dir[0][4].append(QPoint(-1,2));

    dir[0][4].append(QPoint(1,-2));
```

```
    dir[0][4].append(QPoint(2,1));
    dir[0][4].append(QPoint(-2,-1));
    dir[0][4].append(QPoint(1,2));
    dir[0][4].append(QPoint(-1,-2));

    //rook
    dir[0][5].append(QPoint(-1,0)) ;
    dir[0][5].append(QPoint(1,0)) ;
    dir[0][5].append(QPoint(0,-1)) ;
    dir[0][5].append(QPoint(0,1)) ;

    //pawn
    dir[0][6].append(QPoint(0,1)) ;

    //黑方的棋子能走的dir（实际上只有pawn同白方有区别）
    for(int j=1;j<=TYPENUM;++j){
        for(int k=0;k<dir[0][j].length();++k){
            QPoint p = dir[0][j].at(k) ;
            p.setY(-p.y());
            dir[1][j].append(p) ;
        }
    }
}

void MainWindow::paintEvent(QPaintEvent *event){
    QPainter painter(this) ;
    painter.setRenderHint(QPainter::Antialiasing, true); // 抗锯齿

    //网格及边线
    for(int i=1;i<=col;++i){
        for(int j=1;j<=row;++j){
            QPoint p1=getPoint(i,j+1), p2=getPoint(i+1,j) ;
            painter.setPen(Qt::NoPen);
            painter.setBrush(QBrush(groundColor[getGroundType(i,j)], Qt::SolidPattern)) ;
            painter.drawRect(QRect(p1,p2)) ;
        }
    }
    painter.setPen(Qt::SolidLine);
    for(int i=1;i<=col;++i) {
        painter.setPen(Qt::black);
        painter.drawLine(getPoint(i, 1), getPoint(i+1, 1)) ;
        painter.drawLine(getPoint(i, row+1), getPoint(i+1, row+1)) ;
        QPoint tmp = getPoint(i+1, 1) ;
        painter.setPen(groundColor[getGroundType(i,1)^1]);
        QFont font = painter.font();
        font.setPointSize(tagSize) ;
        painter.drawText(tmp.x()-tagSize-3, tmp.y()-3, ind2char(i)) ;
    }
    for(int j=1;j<=row;++j) {
        painter.setPen(Qt::black);
        painter.drawLine(getPoint(1, j), getPoint(1, j+1)) ;
        painter.drawLine(getPoint(col+1, j), getPoint(col+1, j+1)) ;

        QPoint tmp = getPoint(1, j+1) ;
```

```
        painter.setPen(groundColor[getGroundType(1,j)^1]);
        QFont font = painter.font();
        font.setPointSize(tagSize) ;
        painter.drawText(tmp.x()+3, tmp.y()+tagSize+6, QString::number(j)) ;
    }
    QPoint tmp = getPoint(col+1,1);
    if(!debugOn){
        this->setMinimumSize(tmp.x()+50, tmp.y()+50);
        this->setMaximumSize(tmp.x()+50, tmp.y()+50);
    } else{
        this->setMinimumSize(tmp.x()+350, tmp.y()+50);
        this->setMaximumSize(tmp.x()+350, tmp.y()+50);
    }

    //棋子图片
    //debug(QString("nowChessman.length(): %1").arg(nowChessman.length())) ;
    for(int i=0;i<nowChessman.length();++i){
        Chessman man = nowChessman.at(i) ;
        QString path = ":/new/prefix1/res/" + QString(man.color ? "black_" : "white_") +
ind2type(man.type) + ".png";
        QPoint pos = man.pos ;
       // debug("path:"+path+QString("   pos:(%1,%2)").arg(pos.x()).arg(pos.y())) ;
        label[i]->setPixmap(QPixmap(path)) ;
        label[i]->setGeometry(QRect(getPoint(pos.x(),pos.y()+1), getPoint(pos.x()+1,pos.y()))) ;
        label[i]->setScaledContents(true) ;
        label[i]->lower();
        label[i]->show();
    }
    for(int i=nowChessman.length();i<MAXM;++i)
        label[i]->hide();

    //当前候选位置
    for(int i=0;i<myNextCandidate.length();++i){
        QPoint p=myNextCandidate.at(i) ;
        QPen pen = painter.pen();
        if(nowChessman.at(getChessmanIndOnPos(nowChoose)).type==TYPEKING
                && std::abs(nowChoose.x()-p.x())>=2){
            //王车易位
            pen.setColor(castlingColor) ;
        }
        else pen.setColor(circleColor);
        pen.setWidth(circleR/5) ;
        painter.setPen(pen) ;
        int margin=(gridSize-2*circleR)/2;

 painter.drawArc(QRect(getPoint(p.x(),p.y()+1)+QPoint(margin,margin),getPoint(p.x()+1,p.y())+QPo
int(-margin,-margin)),0,360*16) ;
    }

    //升变
    if(upgradingInd!=-1){
        for(int i=0;i<nowChessman.length();++i){

            QPoint pos = nowChessman.at(i).pos ;
```

```
            if(pos.x()>=3 && pos.x()<=6 && pos.y()>=4 && pos.y()<=5)
                label[i]->hide();
        }
        int color = nowChessman.at(upgradingInd).color;
        QPen pen = painter.pen();
        pen.setColor(groundColor[1]) ;
        pen.setWidth(3);
        painter.setPen(pen);
        painter.setBrush(QBrush(groundColor[0],Qt::SolidPattern));

 painter.drawRoundedRect(QRect(upgradeLeftUp,upgradeLeftUp+QPoint(4*gridSize,2*gridSize)),
                                15,15) ;
        for(int i=2;i<=5;++i){
            QString path = ":/new/prefix1/res/" + QString(color ? "black_" : "white_") +
ind2type(i) + ".png";
            QPoint pos = getPoint(3+i-2,6) + QPoint(0,gridSize/2) ;
            upgradeLabel[i-2]->setPixmap(QPixmap(path)) ;
            upgradeLabel[i-2]->setGeometry(QRect(pos, pos+QPoint(gridSize,gridSize))) ;
            upgradeLabel[i-2]->setScaledContents(true) ;
            upgradeLabel[i-2]->lower();
            upgradeLabel[i-2]->show();
            QPen pen = painter.pen();
            pen.setColor(groundColor[1]) ;
            pen.setWidth(1);
            painter.setPen(pen);
            painter.setBrush(Qt::NoBrush);
            QPoint margin=QPoint(3,3);
            painter.drawRoundedRect(QRect(pos+margin, pos+QPoint(gridSize,gridSize)-
margin),10,10) ;
        }
    } else{
        for(int i=0;i<4;++i)
            upgradeLabel[i]->hide();
    }
}

void MainWindow::moveChessman(int ind, QPoint p){
    //移动操作，将索引为ind的棋子移动到p位置，注意可能只是作为测试移动而被其它函数调用
    int tmpInd = getChessmanIndOnPos(p) ;
    Chessman man = nowChessman.at(ind) ;
    man.pos = p ;
    nowChessman.replace(ind, man);
    if(tmpInd!=-1){ //注意删除后会导致索引改变
        nowChessman.removeAt(tmpInd) ;
    }
}

bool MainWindow::outGridRange(QPoint pos){
    return pos.x()<1||pos.y()<1||pos.x()>col||pos.y()>row;
}

int MainWindow::getPawnStatus(Chessman man){

    //分为在起始位置、普通位置以及升变位置三种情况
```

```
    int y = man.pos.y();
    if(man.color==1) y=col+1-y; //统一黑白
    if(y==2) return PAWNINI ;
    else if(y==8) return PAWNUPGRADE;
    else return PAWNNORMAL;
}

QList<QPoint> MainWindow::getCandidatePos(Chessman man){
    //计算棋子下一步能走的所有位置，注意这里是不考虑将军的情况下合法能走到的位置
    QList<QPoint> list;
    int type = man.type, color = man.color;
    QPoint pos = man.pos ;
    for(int i=0;i<dir[color][type].length();++i){
        QPoint d = dir[color][type].at(i) ;
        int up = (canWalkMore[type] ? 7 : 1) ;
        for(int j=1;j<=up;++j){
            QPoint newPos = pos + d*j ;
            if(outGridRange(newPos)) break ;
            int tmpInd = getChessmanIndOnPos(newPos) ;
            if(tmpInd==-1 || nowChessman.at(tmpInd).color!=color){ //可移动
                if(type==TYPEPAWN && tmpInd!=-1) //pawn不可直接吃
                    continue;
                list.append(newPos) ;
            }
            if(tmpInd!=-1) break ; //被遮挡
        }
    }
    if(type==TYPEPAWN){ //特殊处理pawn
        QPoint d = dir[color][type].at(0) , newPos = pos+2*d;
        if(getPawnStatus(man)==PAWNINI && getChessmanIndOnPos(pos+d)==-1 &&
getChessmanIndOnPos(newPos)==-1 && !outGridRange(newPos)){
            list.append(newPos);
        }
        //吃子
        newPos = pos+d+QPoint(1,0);
        if(getChessmanIndOnPos(newPos)!=-1 &&
nowChessman.at(getChessmanIndOnPos(newPos)).color!=color){
            list.append(newPos);
        }
        newPos = pos+d+QPoint(-1,0);
        if(getChessmanIndOnPos(newPos)!=-1 &&
nowChessman.at(getChessmanIndOnPos(newPos)).color!=color){
            list.append(newPos);
        }
    }

    return list;
}

QList<QPoint> MainWindow::getCandidatePosWithCheck(Chessman man){
    //计算棋子下一步能走的所有位置，考虑不能被将军
    QList<QPoint> list = getCandidatePos(man) ;

    int ind = nowChessman.indexOf(man) ;
```

```
    QList<QPoint>::iterator it = list.begin();
    while(it!=list.end()){
        QPoint newPos = *it;
        QList<Chessman> tmpNowChessman = nowChessman; //备份nowChessman
        moveChessman(ind, newPos) ;
        int ck = isCheck() ;
        nowChessman = tmpNowChessman ;
        if(ck & (man.color ? CHECKBLACK : CHECKWHITE)){
            it = list.erase(it) ;
        } else {
            ++ it ;
        }
    }

    //王车易位
    if(man.type==TYPEKING){
        int y = (man.color ? 8 : 1);
        if(man.pos == QPoint(5,y) && !(isCheck() & (man.color ? CHECKBLACK : CHECKWHITE))){
            bool canLongCastling=false, canShortCastling=false;
            Chessman rook1, rook2 ;
            for(int i=0;i<nowChessman.length();++i){
                Chessman rook = nowChessman.at(i);
                if(rook.color==man.color && rook.type==TYPEROOK){
                    if(rook.pos == QPoint(1,y)){
                        canLongCastling=true;
                        rook1 = rook;
                    }
                    else if(rook.pos == QPoint(8,y)){
                        canShortCastling=true;
                        rook2 = rook;
                    }
                }
            }
            if(getChessmanIndOnPos(QPoint(4,y))!=-1 || getChessmanIndOnPos(QPoint(3,y))!=-1)
canLongCastling=false;
            if(getChessmanIndOnPos(QPoint(6,y))!=-1 || getChessmanIndOnPos(QPoint(7,y))!=-1)
canShortCastling=false;
            if(canLongCastling){
                QList<Chessman> tmpNowChessman = nowChessman ;
                moveChessman(ind, QPoint(4,y)) ;
                if(isCheck() & (man.color ? CHECKBLACK : CHECKWHITE)){
                    canLongCastling=false;
                }
                moveChessman(ind, QPoint(3,y)) ;
                moveChessman(nowChessman.indexOf(rook1), QPoint(4,y)) ;
                if(isCheck() & (man.color ? CHECKBLACK : CHECKWHITE)){
                    canLongCastling=false;
                }
                nowChessman = tmpNowChessman ;
                if(canLongCastling){
                    list.append(QPoint(3,y));
                }

            }
```

```
            if(canShortCastling){
                QList<Chessman> tmpNowChessman = nowChessman ;
                moveChessman(ind, QPoint(6,y)) ;
                if(isCheck() & (man.color ? CHECKBLACK : CHECKWHITE)){
                    canShortCastling=false;
                }
                moveChessman(ind, QPoint(7,y)) ;
                moveChessman(nowChessman.indexOf(rook2), QPoint(6,y)) ;
                if(isCheck() & (man.color ? CHECKBLACK : CHECKWHITE)){
                    canShortCastling=false;
                }
                nowChessman = tmpNowChessman ;
                if(canShortCastling){
                    list.append(QPoint(7,y));
                }
            }
        }
    }

    return list;
}

void MainWindow::passOneSec(){
    if(!ui->actionPauseTimer->isChecked()){
        if(timeRes>0){
            if(--timeRes == 0 && nowStatus==STATUSMYTURN){
                on_actionGiveIn_triggered();
            }
            ui->lcdNumber->display(timeRes) ;
            timeout=0;
        } else{
            if(++timeout==3){
                setStatus(nowColor ? STATUSWHITEWIN : STATUSBLACKWIN) ;
            }
        }
    }
}

void MainWindow::setStatus(int status){
    //棋盘状态改变
    //debug("TTT,status:"+QString::number(status)) ;
    nowStatus = status;
    nowChoose = QPoint(-1,-1);
    myNextCandidate.clear();
    timeRes = timeLim ;
    ui->lcdNumber->display(timeRes) ;
    playTimer->stop() ;
    upgradingInd = -1;
    if(isRunning()){
        if(ui->actionDebug->isChecked()==false){
            ui->actionLoadInit->setEnabled(false);

            ui->actionLoadFromFile->setEnabled(false);
```

```
        } else if(isPlayingOnline && nowColor == remotePlayer->getColor()){
            ui->actionLoadInit->setEnabled(false);
            ui->actionLoadFromFile->setEnabled(false);
        } else{
            ui->actionLoadInit->setEnabled(true);
            ui->actionLoadFromFile->setEnabled(true);
        }
        ui->actionPVP->setEnabled(false);
        ui->actionConnectHost->setEnabled(false);
        ui->actionCreateHost->setEnabled(false);
        playTimer->start(1000) ;
        ui->labelMyColor->setText(QString("我方：") + (remotePlayer->getColor() ? "白方" : "黑
方")) ;
        ui->labelNowColor->setText(QString("执子方：") + (nowColor ? "黑方" : "白方")) ;
    } else{
        ui->actionLoadInit->setEnabled(true);
        ui->actionLoadFromFile->setEnabled(true);
        ui->actionPVP->setEnabled(true);
        ui->actionConnectHost->setEnabled(true);
        ui->actionCreateHost->setEnabled(true);
        if(status!=STATUSNOTRUN) for(int i=0;i<=1;++i) player[i]->gameEnd(status) ;
        if(isPlayingOnline) communication->close() ;
        isPlayingOnline=false ;
    }
    if(status==STATUSMYTURN){
        ui->actionGiveIn->setEnabled(true);
    } else{
        ui->actionGiveIn->setEnabled(false);
    }
    if(status==STATUSWHITEWIN){
        QMessageBox::information(this, "游戏结束", "白方胜利!") ;
    } else if(status==STATUSBLACKWIN){
        QMessageBox::information(this, "游戏结束", "黑方胜利!") ;
    } else if(status==STATUSTIE){
        QMessageBox::information(this, "游戏结束", "和棋!") ;
    }
    update() ;
}

int MainWindow::isCheck(){
    //判断哪个颜色被将军，0为没有被将军，1为白色被将军，2为黑色被将军，3为同时被将军，用常变量表示
    int ret=0;
    QPoint kingPos[2];
    for(int i=0;i<nowChessman.length();++i){
        Chessman man = nowChessman.at(i) ;
        if(man.type==TYPEKING) kingPos[man.color]=man.pos;
    }
    for(int i=0;i<nowChessman.length();++i){
        //枚举走哪个棋子
        Chessman man = nowChessman.at(i) ;
        QList<QPoint> list = getCandidatePos(man) ;
        if(list.indexOf(kingPos[man.color^1])!=-1){

            ret |= (man.color ? CHECKWHITE : CHECKBLACK) ;
```

```
        }
    }
    return ret;
}

int MainWindow::isStuck(){
    //判断哪个颜色无法走子，0为没有，1为白色，2为黑色，3为两者用和isCheck一样的常变量进行表示
    int ret=3 ;
    for(int i=0;i<nowChessman.length();++i){
        Chessman man = nowChessman.at(i) ;
        QList<QPoint> list = getCandidatePosWithCheck(man) ;
        if(list.length()>0) {
            ret &= (man.color ? (~CHECKBLACK) : (~CHECKWHITE)) ;
        }
    }
    return ret;
}

int MainWindow::isCheckMate(){
    //判断哪个颜色被将杀，0为没有被将杀，1为白色被将杀，2为黑色被将杀，用和isCheck一样的常变量表示
    return (isCheck()&isStuck()) ;    //正被将军且无法移动的一方被将杀
}

int MainWindow::isStaleMate(){
    //判断哪个颜色被逼和，0为没有被逼和，1为白色被逼和，2为黑色被逼和，用和isCheck一样的常变量表示
    return ((~isCheck())&isStuck()) ; //未被将军且无法移动的一方被逼和
}

void MainWindow::mousePressEvent(QMouseEvent *event)
{
    if(event->button() == Qt::LeftButton){
        int x=event->x(), y=event->y() ;
        if(upgradingInd!=-1){
            //兵升变
            QPoint p = upgradeLeftUp + QPoint(0, gridSize/2) ;
            x=static_cast<int>(std::floor(static_cast<double>(x-p.x())/gridSize)+1);
            y=static_cast<int>(std::floor(static_cast<double>(y-p.y())/gridSize)+1);
            debug(QString("Press: (%1,%2)").arg(x).arg(y)) ;
            if(y==1 && x>=1 && x<=4){
                int type=x+1 ;
                Chessman man = nowChessman.at(upgradingInd) ;
                man.type = type;
                nowChessman.replace(upgradingInd, man) ;
                upgradingInd = -1;
                update();
                nextPlayer();
                sendMessage(getChessStr()) ;
            }
            return ;
        }
        x=static_cast<int>(std::floor(static_cast<double>(x-leftUp.x())/gridSize)+1);
        y=static_cast<int>(std::floor(static_cast<double>(y-leftUp.y())/gridSize)+1);

        y=row+1-y;
```

```
        debug(QString("Press: (%1,%2)").arg(x).arg(y)) ;
        if(x<1||y<1||x>col||y>row)  return ;
        if(nowStatus==STATUSMYTURN){
            if(nowChoose == QPoint(x,y)){
                //选中了上次选中的棋子
                nowChoose = QPoint(-1,-1);
            } else{
                int ind = getChessmanIndOnPos(QPoint(x,y)) ;
                if(nowChoose == QPoint(-1,-1)){
                    //判断是否新选中了棋子
                    if(ind!=-1 && nowChessman.at(ind).color==nowColor){
                        nowChoose = QPoint(x,y);
                    }
                } else{
                    //之前已经选中过某个棋子
                    //移动
                    bool suc=false ;
                    for(int i=0;i<myNextCandidate.length();++i){
                        if(myNextCandidate.at(i) == QPoint(x,y)){
                            suc=true;
                            int ind = getChessmanIndOnPos(nowChoose);
                            moveChessman(ind, QPoint(x,y)) ; //可能导致吃子进而索引改变

                            //处理兵升变
                            ind = getChessmanIndOnPos(QPoint(x,y));
                            Chessman man = nowChessman.at(ind) ;
                            if(man.type==TYPEPAWN && getPawnStatus(man)==PAWNUPGRADE){
                                upgradingInd = ind ;
                                nowChoose = QPoint(-1,-1) ;
                                break ;
                            }

                            //处理王车易位
                            if(man.type==TYPEKING && std::abs(man.pos.x()-nowChoose.x())==2){
                                int y = (man.color ? 8 : 1) ;
                                int x = ((man.pos.x()<nowChoose.x()) ? 1 : 8) ;
                                int rookInd =
nowChessman.indexOf(Chessman(TYPEROOK,man.color,QPoint(x,y))) ;
                                moveChessman(rookInd, QPoint(((x==1) ? 4 : 6), y)) ;
                            }

                            nextPlayer() ;
                            nowChoose = QPoint(-1,-1) ;
                            sendMessage(getChessStr()) ;
                            break;
                        }
                    }

                    //重选
                    if(!suc && ind!=-1 && nowChessman.at(ind).color==nowColor){
                        nowChoose = QPoint(x,y);
                    }

                }
```

```
            }

            if(nowChoose != QPoint(-1,-1)){
                int ind = getChessmanIndOnPos(nowChoose) ;
                myNextCandidate = getCandidatePosWithCheck(nowChessman.at(ind));
            } else{
                myNextCandidate.clear();
            }
        }
        //debug("test4");
        update();
        //debug("test5");
    }
}

int MainWindow::getChessmanIndOnPos(QPoint pos){
    //返回某个位置上的Chessman索引
    int ret=-1 ;
    for(int i=0;i<nowChessman.length();++i){
        if(nowChessman.at(i).pos == pos){
            ret = i ;
            break ;
        }
    }
    return ret;
}

void MainWindow::debug(QString s){
    if(debugOn){
        qDebug() << s ;
        QStringList list = s.split('\n') ;
        for(int i=0;i<list.length();++i)
            textBrowser->append(list.at(i)) ;
    }
}

bool MainWindow::isRunning()
{
    return nowStatus==STATUSMYTURN || nowStatus==STATUSOPPTURN ;
}

void MainWindow::checkGameStatus(){
    //接下来是本地玩家着子，判断是否已经输了或者逼和
    if(isCheckMate() & (nowColor ? MainWindow::CHECKBLACK : MainWindow::CHECKWHITE)){
        setStatus(nowColor ? MainWindow::STATUSWHITEWIN : MainWindow::STATUSBLACKWIN) ;
    } else if(isStaleMate() & (nowColor ? MainWindow::CHECKBLACK : MainWindow::CHECKWHITE)){
        setStatus(STATUSTIE) ;
    }
}

QPoint MainWindow::getPoint(int x, int y){
    y = row+2-y;

    return QPoint((x-1)*gridSize+leftUp.x(), (y-1)*gridSize+leftUp.y());
```

```
}

int MainWindow::char2ind(QChar s){
    return s.toLatin1()-'a'+1;
}

QChar MainWindow::ind2char(int a){
    return QChar(a+'a'-1) ;
}

int MainWindow::type2ind(QString s){
    if(s=="king"){
        return 1;
    } else if(s=="queen"){
        return 2;
    } else if(s=="bishop"){
        return 3;
    } else if(s=="knight"){
        return 4;
    } else if(s=="rook"){
        return 5;
    } else if(s=="pawn"){
        return 6;
    } else{
        return -1;
    }
}

QString MainWindow::ind2type(int a){
    if(a==1){
        return "king" ;
    } else if(a==2){
        return "queen" ;
    } else if(a==3){
        return "bishop" ;
    } else if(a==4){
        return "knight" ;
    } else if(a==5){
        return "rook" ;
    } else if(a==6){
        return "pawn" ;
    } else{
        return "" ;
    }
}

QPoint MainWindow::str2pos(QString s){
    assert(s.length()==2);
    return QPoint(char2ind(s.at(0)), s.mid(1,1).toInt()) ;
}

QString MainWindow::pos2str(QPoint pos){

    return ind2char(pos.x()) + QString::number(pos.y()) ;
```

```
}

QList< Chessman> MainWindow::str2chessman(QString s, int color){
    //将一行字符串转换为若干chessman
    QList< Chessman> list;
    QStringList strList = s.split(' ') ;
    int type = type2ind(strList.at(0)) ;
    for(int i=2;i<strList.length();++i){
        if(strList.at(i).trimmed()=="") continue ;
        QPoint pos = str2pos(strList.at(i)) ;
        list.append( Chessman(type, color, pos)) ;
    }
    return list;
}

QString MainWindow::chessman2str(QList< Chessman> &list){
    //将若干相同类型的chessman转成字符串
    if(list.length()==0) return QString("") ;
    QString ret = ind2type(list.at(0).type) + " " + QString::number(list.length()) + " ";
    for(int i=0;i<list.length();++i){
        ret = ret + pos2str(list.at(i).pos) + ((i==list.length()-1) ? "" :" ");
    }
    return ret;
}

QString MainWindow::getChessStr(){
    //把当前局面转化为字符串
    QString ret = (nowColor ? "black\n" : "white\n") ;
    QList< Chessman> tmpList[COLORNUM][TYPENUM+1] ;
    for(int i=0;i<nowChessman.length();++i){
         Chessman man = nowChessman.at(i) ;
        tmpList[man.color][man.type].append(man) ;
    }
    for(int i=1;i<=TYPENUM;++i){
        QString tmp = chessman2str(tmpList[nowColor][i]) ;
        if(tmp!="") ret = ret + tmp + "\n" ;
    }
    ret = ret + (nowColor ? "white\n" : "black\n") ;
    for(int i=1;i<=TYPENUM;++i){
        QString tmp = chessman2str(tmpList[nowColor^1][i]) ;
        if(tmp!="") ret = ret + tmp + "\n" ;
    }
    return ret;
}

void MainWindow::loadChessStr(QString chessStr){
    int color=0 ;
    QStringList strList = chessStr.split('\n') ;
    QList<Chessman> list ;
    int now = 0 ;
    while(now < strList.length()){
        QString s = strList.at(now) ;

        //debug("test:" + s + " " + QString::number(now)) ;
```

```
        if(s.trimmed()=="white"){
            color=0 ;
        } else if(s.trimmed()=="black"){
            color=1 ;
        } else {
            list.append(str2chessman(s, color)) ;
        }
        ++now ;
    }
    nowChessman = list ;
    nowColor = color^1;
    if(isRunning()){
        player[nowColor]->play();
    }
    update() ;
}

int MainWindow::getGroundType(int x, int y){
    //返回地面颜色（黑色为1，白色为0）
    return (x+y)%2^1 ;
}

void MainWindow::nextPlayer(){
    player[nowColor^1]->play();
}

void MainWindow::sendMessage(QString s){
    if(isPlayingOnline){
        debug("SEND PACK!") ;
        communication -> sendMessage(s);
    }
}

void MainWindow::handleReadPack(){
    debug("READ PACK!") ;

    while(communication->hasNextPack()){
        QString s = communication->nextPack();
        debug("GETPACK! Length:" + QString::number(s.length())) ;
        if(s==MESSAGEWHITEWIN){
            setStatus(STATUSWHITEWIN) ;
        } else if(s==MESSAGEBLACKWIN){
            setStatus(STATUSBLACKWIN) ;
        } else if(s==MESSAGETIE){
            setStatus(STATUSTIE) ;
        } else if(nowStatus==STATUSOPPTURN){
            loadChessStr(s) ;
        }
    }
}

void MainWindow::closeEvent(QCloseEvent *event){

    if(isPlayingOnline){
```

```
        on_actionGiveIn_triggered();
        event->accept();
    }
}

void MainWindow::startOnlineGame(QTcpSocket *tcpSocket, int color){
    communication = new Communication(this, tcpSocket, this) ;
    connect(communication, SIGNAL(readyReadPack()), this, SLOT(handleReadPack()));
    isPlayingOnline=true;
    player[color] = localPlayer[color] ;
    remotePlayer->setColor(color^1) ;
    player[color^1] = remotePlayer ;
    if(!color){
        sendMessage(getChessStr()) ;
        player[nowColor]->play();
    } else{
        nowColor=0 ;
        player[nowColor]->play();
    }
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_actionLoadInit_triggered()
{
    loadChessStr(iniChessmanStr) ;
    if(isPlayingOnline){
        sendMessage(iniChessmanStr);
    }
    update();
}

void MainWindow::on_actionLoadFromFile_triggered()
{
    QString filePath = QFileDialog::getOpenFileName(this, "打开文件", "./", "All Files(*.*)") ;
    if(filePath=="") return ;
    QFile file(filePath);
    if(!file.open(QIODevice::ReadOnly | QIODevice::Text)){
        QMessageBox::critical(this, "错误", "打开文件失败");
        return ;
    }
    QTextStream in(&file);
    QString s = in.readAll();
    loadChessStr(s);
    if(isPlayingOnline){
        sendMessage(s);
    }
    file.close();
}
```

```
void MainWindow::on_actionSaveChess_triggered()
{
    QString filePath = QFileDialog::getSaveFileName(this, "保存文件", "./", "All Files(*.*)") ;
    if(filePath=="") return ;
    QFile file(filePath);
    if(!file.open(QIODevice::WriteOnly | QIODevice::Text)){
        QMessageBox::critical(this, "错误", "保存文件失败");
        return ;
    }
    QTextStream out(&file);
    out << getChessStr();
    file.close();
}

void MainWindow::on_actionPVP_triggered()
{
    setStatus(STATUSMYTURN) ;
    player[0] = dynamic_cast<Player*>(localPlayer[0]);
    player[1] = dynamic_cast<Player*>(localPlayer[1]);
    nowColor = 0;
    if(nowColor!=player[0]->getColor()){
        std::swap(player[0],player[1]);
    }
    player[0]->play();
}

void MainWindow::on_actionGiveIn_triggered()
{
    if(isPlayingOnline){
        int myColor = 0;
        if(myColor == remotePlayer->getColor()) myColor^=1;
        setStatus(myColor ? STATUSWHITEWIN : STATUSBLACKWIN) ;
    }
    else setStatus(nowColor ? STATUSWHITEWIN : STATUSBLACKWIN) ;
}

void MainWindow::on_actionCreateHost_triggered()
{
    DialogCreateHost *dialogCreateHost = new DialogCreateHost(this, this) ;
    dialogCreateHost -> show();
}

void MainWindow::on_actionConnectHost_triggered()
{
    DialogConnectToHost *dialogConnectToHost = new DialogConnectToHost(this, this) ;
    dialogConnectToHost->show() ;
}

void MainWindow::on_actionDebug_triggered()
{
    debugOn = ui->actionDebug->isChecked() ;
    if(debugOn){

        textBrowser->show();
```

```
    } else{
        textBrowser->hide();
    }
    update();
}

void MainWindow::on_actionPauseTimer_triggered()
{

}
```

communication.cpp

```
#include "communication.h"
#include "mainwindow.h"

Communication::Communication(QObject *parent, QTcpSocket *tcpSocket, MainWindow *mainWindow) :
QObject(parent)
{
    packHead = "CHESSPACK YYR";
    initTcpSocket(tcpSocket);
    this->mainWindow = mainWindow;
}

void Communication::initTcpSocket(QTcpSocket *tcpSocket){
    this->tcpSocket = tcpSocket ;
    if(tcpSocket != nullptr)
        connect(tcpSocket, SIGNAL(readyRead()), this, SLOT(handleRead())) ;
    readBuffer.clear();
    packages.clear();
}

void Communication::close()
{
    tcpSocket->close();
}

QString Communication::pack(QString s){
    return packHead + "\n" + QString::number(s.length()) + "\n" + s;
}

QString Communication::unpack(QString s, int &pos){
    //拆包成功则返回原信息，否则返回空串。pos用于返回字符串s中包结尾的下一位置
    QStringList strList = s.split('\n');
    pos=-1;
    if(strList.length()<3) return "" ;
    bool ok;
    int contentLen=strList.at(1).toInt(&ok),
headLen=strList.at(0).length()+1+strList.at(1).length()+1; //包头和包内容的长度
    if(strList.at(0)!=packHead || !ok) return ""; //不是以包头作为开头
    if(headLen+contentLen>s.length()) return ""; //非完整包
```

```
    pos = headLen+contentLen ;
    return s.mid(headLen, contentLen) ;
}

bool Communication::hasNextPack()
{
    return packages.length()>0;
}

QString Communication::nextPack()
{
    QString ret = packages.at(0);
    packages.pop_front();
    return ret;
}

void Communication::handleRead(){
    mainWindow->debug("READ MESSAGE");
    readBuffer += tcpSocket->readAll() ;
    int pos;
    QString tmp;
    if((tmp=unpack(readBuffer, pos))!=""){
        mainWindow->debug("THE MESSAGE IS A PACK");
        readBuffer.remove(0, pos) ;
        packages.append(tmp) ;
        emit(readyReadPack()) ;
    }
}

void Communication::sendMessage(QString s)
{
    if(tcpSocket != nullptr){
        QByteArray *byteArray = new QByteArray;
        byteArray->clear();
        s = pack(s);
        //mainWindow->debug(s) ;
        byteArray->append(s);
        tcpSocket->write(*byteArray);
    }
}
```

dialogcreatehost.cpp

```
#include "dialogcreatehost.h"
#include "ui_dialogcreatehost.h"
#include "mainwindow.h"
#include <QMessageBox>

DialogCreateHost::DialogCreateHost(QWidget *parent, MainWindow *mainWindow) :
    QDialog(parent),
    ui(new Ui::DialogCreateHost)
{
```

```
    ui->setupUi(this);
    isListening = false;
    this->mainWindow = mainWindow;
    tcpServer = new QTcpServer();
    connect(tcpServer, SIGNAL(newConnection()), this, SLOT(handleNewConnection())) ;
    ui->lineEdit->setText("127.0.0.1") ;
}

void DialogCreateHost::handleNewConnection(){
    tcpSocket = tcpServer->nextPendingConnection();
    tcpServer->close();
    mainWindow->startOnlineGame(tcpSocket, 0) ;
    this->close();
}

DialogCreateHost::~DialogCreateHost()
{
    delete ui;
}

bool DialogCreateHost::checkIP(QString s)
{
    QRegExp regExp("((\\d{1,2})|(1\\d{2})|(2[0-4]\\d)|(25[0-5])).){3}((\\d{1,2})|(1\\d{2})|
(2[0-4]\\d)|(25[0-5]))");
    return regExp.indexIn(s)!=-1 && regExp.matchedLength()==s.length();
}

void DialogCreateHost::on_pushButtonClose_clicked()
{
    tcpServer->close();
    this->close();
}

void DialogCreateHost::on_pushButtonStart_clicked()
{
    isListening ^= 1;
    if(isListening){
        QString s = ui->lineEdit->text();
        QHostAddress addr(s) ;
        if(!checkIP(s)){
            QMessageBox::critical(this, "错误", "请输入正确ip格式");
            isListening=0;
            return;
        }
        tcpServer->listen(addr, PORT) ;
        ui->pushButtonStart->setText("停止") ;
    } else{
        tcpServer->close();
        ui->pushButtonStart->setText("创建") ;
    }

}
```

dialogconnecttohost.cpp

```
#include "dialogconnecttohost.h"
#include "ui_dialogconnecttohost.h"
#include "dialogcreatehost.h"
#include "mainwindow.h"
#include <QMessageBox>

DialogConnectToHost::DialogConnectToHost(QWidget *parent, MainWindow *mainWindow) :
    QDialog(parent),
    ui(new Ui::DialogConnectToHost)
{
    ui->setupUi(this);
    tcpSocket = new QTcpSocket();
    connect(tcpSocket, SIGNAL(connected()), this, SLOT(handleConnected())) ;
    isConnecting=false;
    this->mainWindow = mainWindow;
}

DialogConnectToHost::~DialogConnectToHost()
{
    delete ui;
}

void DialogConnectToHost::handleConnected()
{
    mainWindow->startOnlineGame(tcpSocket, 1) ;
    this->close();
}

void DialogConnectToHost::on_pushButtonCancel_clicked()
{
    tcpSocket->close();
    this->close();
}

void DialogConnectToHost::on_pushButtonConnect_clicked()
{
    isConnecting^=1;
    if(isConnecting){
        QString s = ui->lineEdit->text();
        QHostAddress addr(s) ;
        if(!DialogCreateHost::checkIP(s)){
            QMessageBox::critical(this, "错误", "请输入正确ip格式");
            isConnecting=false;
            return;
        }
        tcpSocket->connectToHost(addr, DialogCreateHost::PORT);
        ui->pushButtonConnect->setText("取消连接") ;
    } else{
        tcpSocket->close();

        ui->pushButtonConnect->setText("连接") ;
```

```
    }
}
```