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。

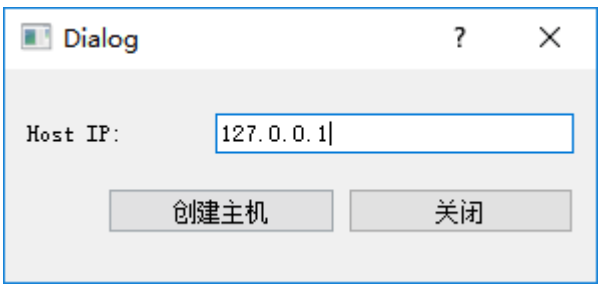
通信逻辑

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。

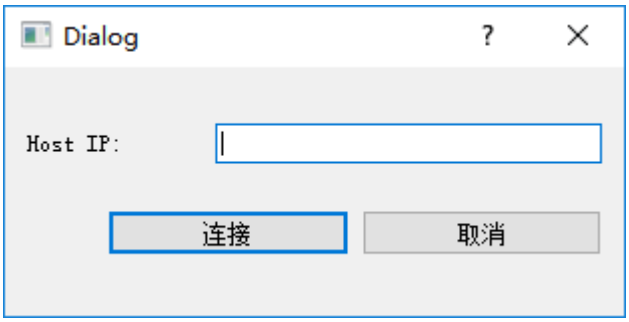
在监听到连接之后便回调主窗口类中的startOnlineGame函数，监听成功的代码如下：

```
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：

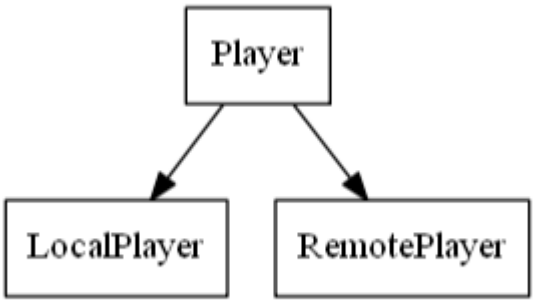
```
void MainWindow::startOnlineGame(QTcpSocket *tcpSocket, int color){
    communication = new Communication(this, tcpSocket, this) ;
    connect(communication, SIGNAL(readyReadPack()), this, SLOT(handleReadPack()));
    isPlayingOnline=true;
    player[color] = localPlayer[color] ;
    remotePlayer->setColor(color^1) ;
    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。

游戏规则