网络斗地主游戏设计文档

文档最后更新日期

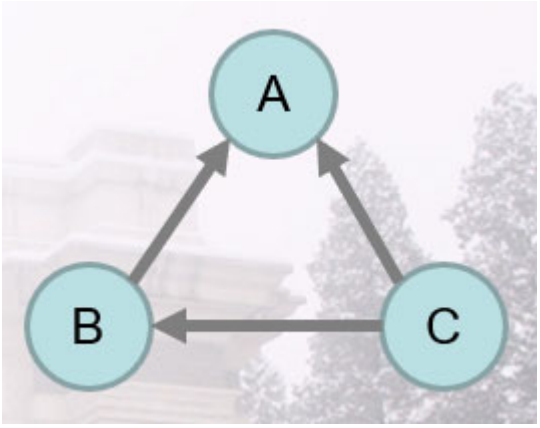
2020.09.06

概述

本次作业内容主要涉及网络编程，主要使用 Qt 所提供的各种界面类以及监听套接字 QTcpServer 和 QTcpSocket 来实现。

客户端工作流程

本次作业中共含有三个客户端 A,B,C，作为进行游戏的三个玩家。由于三个玩家之间互相需要通信以实现相关数据的同步，因此在程序中共设立了两个监听套接字和三个通信套接字，具体做法是在 A 客户端处设立一个监听套接字，并且作为整个游戏的主控制端，B,C 两处分别设立一个通信套接字与 A 连接；而对于 B,C 之间的数据传输，采取的办法是在 B 处另设一个监听套接字，用以和 C 中相应的通信套接字进行连接。整个连接状态如下图所示



之后，在游戏的过程中，作为主控制端，A 在开始的时候需要进行分牌，发牌以及随机选取一人开始叫地主的职能；在 A 中执行完相关的函数分发好数据之后，只需将这些数据通过网络传输给 B,C 客户端进行处理即可。

通信协议

本次程序主要使用的网络通信协议是 TCP/IP 协议，使用 qt 提供的 QTcpServer 和 QTcpSocket 两个套接字类及相关的函数来实现。在网络数据传输中，IP 协议是一个地址协议，只定义了相关的地址规则，实现了路由功能；而

TCP 协议则主要是保证数据通信的完整性和可靠性，防止传输过程中出现丢包。

粘包及其解决

由于此应用程序较为简单，客户端之间相互发送的数据常小于 TCP 发送缓冲区的大小，这造成 TCP 会将多次写入缓冲区的数据一次发送出去，接收端只会收到一次包含了多个不同包的数据，不同包的数据没办法进行拆分，从而发生粘包问题。

由于以上 TCP 的机制无法更改，因此，为解决粘包问题，本程序的解决思路是在每段要进行传输的数据前面加上此数据的长度，此长度信息所占内存为固定的 8 个字节，以方便接受端进行读入；而后依据此来读入指定长度的数据，从而达到一次只读入一个包的效果。具体实现时自定义了一个封装类，在类的相关函数中，为方便操作，要传输的信息都被定义成字符串的形式，相关的代码如下图所示：

```
void TcpDataWrap::mySend(QTcpSocket *sendSocket, QString sdata)
{
    QByteArray sendByte;
    QDataStream out(&sendByte, QIODevice::WriteOnly);
    //out.setVersion(QDataStream::Qt_5_3);
    //设置大端模式，C++、JAVA中都是使用的大端，一般只有linux的嵌入式使用的小端
    out.setByteOrder(QDataStream::BigEndian);
    //占位符,这里必须要先这样占位，然后后续读算出整体长度后在插入
    out << qint64(0);
    //回到文件开头，插入真实的数值
    out.device()->seek(0);
    qint64 len = (qint64)(sdata.size());
    out  << len;
    sendByte+=sdata.toUtf8();
    //qDebug()<<"di22hang"<<sendByte;
    sendSocket->write(sendByte);
}
```

```
QString TcpDataWrap::myRecv(QTcpSocket *recvSocket)
{
    qint64 datasize;
    datasize=recvSocket->read(buffer,8);
    datasize=qFromBigEndian<qint64>(buffer);
    QByteArray result;
    result=recvSocket->read(datasize);
    return result;
}
```

依照上述思路，可以做到触发一次通信套接字的 readyread 信号后就只读入

一个包，而其余的数据仍会留在接收端的缓冲区里，只能等着下次 readyread 信号，造成信息传递的不及时，程序出现 bug。为此，利用 qt 的信号与槽机制，自定义了信息剩余信号，在处理完当前从 socket 读入的数据后会对相应的 socket 缓冲区进行检测，如果仍存在有数据，则向主窗口对象发送信息剩余信号，有主窗口再次调用读入处理函数，以实现对数据的及时处理。先关的代码实现如下：

```
if(bSocket->bytesAvailable()) emit(dataLeftB());
return;
```

针对要传输每种的数据的标识，本程序定义了如下几种：

StartLord 从某位玩家开始抢地主，　Lord 广播地主身份

PreCards 上家所打出的牌 NextCard 下家所打出的牌 GameOver 游戏结束

每种数据的具体形式见程序代码所示。

规则设计实现要点

决定地主规则

由于地主的决定依赖于全部玩家的抉择，因此，对地主的决定，需要从第一个叫地主的玩家开始，依次记录每名玩家的抉择，转过一圈后才可判断出；之后依据 PPT 中给定的规则，从后向前寻找第一位确认叫的玩家，如果没有，就默认为是第一次选择的玩家，即当前玩家。相关的代码实现如下：

```
if(list.size()==4){
    //一轮结束
    //处理结果，找出谁应该是地主
    //向BC广播谁是lord
    int i;
    QString tolord="Lord#";
    for(i=3;i>0;--i){
        if(list.at(i).at(1)=="1"){
            switch(list.at(i).at(0).toLatin1()){
                case 'A':
                    tolord+="A";
                    setRole('A');
                    break;
                case 'C':
                    tolord+="C";
                    setRole('C');
                    break;
                case 'B':
                    tolord+="B";
                    setRole('B');
                    break;
            }
            break;
        }
    }
    if(i==0){
        tolord+="A";
        setRole('A');
    }
```

其中 setRole 函数为将玩家的身份信息—农民或地主展示出来

牌力比较规则

打出的手牌的类型确定

此部分参考了网络上的一些博客，将牌的类型分为如下几种：

```
WangZha 王炸      SingleCard 单牌
DoubleCard 对子      ThreeCard 三不带
BombCard 炸弹      ThreeSingleCard 三带单
ThreeDoubleCard 三带对      BombTwoCard 四带二
BombTwoDoubleCard 四带二双      ConnectCard 连子
CompanyCard 连队      PlaneCard 飞机
PlaneSingleCard 飞机带单牌      PlaneDoubelCard 飞机带对子
ErrorCard 错误
```

在判断时，将各种牌按照次数在打出的牌中的出现次数进行排序，出现较多

的，排在前面，是此牌的主牌；如果出现的一样多，在按照在游戏里面的大小进行从小到大的排序，这样可以较为方便的实现对牌型的判断。

先进行对王炸的判断，显然当二者一个是小王一个是大王时，才可以组成王炸；之后对一些简单的牌型进行判断，当所出的牌的数量小于 5 时，按上过程排序后很容易通过比较第一张和最后一张是否相等以及所打出的牌的总数量来判断出其属于哪个牌型，比如在相等的情况下 1 张的话只能是单牌，2 张的话是对子，三张的话是三不带，四张的话是炸弹。而如果四张的情况下只有第一张和倒数第二张相等的话，只能是三带一。

```
int t=transcards.size();
if(t==2&&transcards[0]==14&&transcards[1]==15){
    cardsType="WangZha";
    return;
}
if(t<5){
    if(transcards[0]==transcards[t-1]){
        switch (t) {
        case 1:
            cardsType="SingleCard";
            break;
        case 2:
            cardsType="DoubleCard";
            break;
        case 3:
            cardsType="ThreeCard";
            break;
        case 4:
            cardsType="BombCard";
            break;
        }
        return;
    }
    if(transcards[0]==transcards[t-2]&&t==4){
        cardsType="ThreeSingleCard";
        return;
    }
}
```

当所打出的牌的数量大于等于五张时，由于可以考虑的情况较多，可能是连对，可能是飞机、四带二等等，情况就比较复杂，因此不再详细叙述，具体请见源代码。