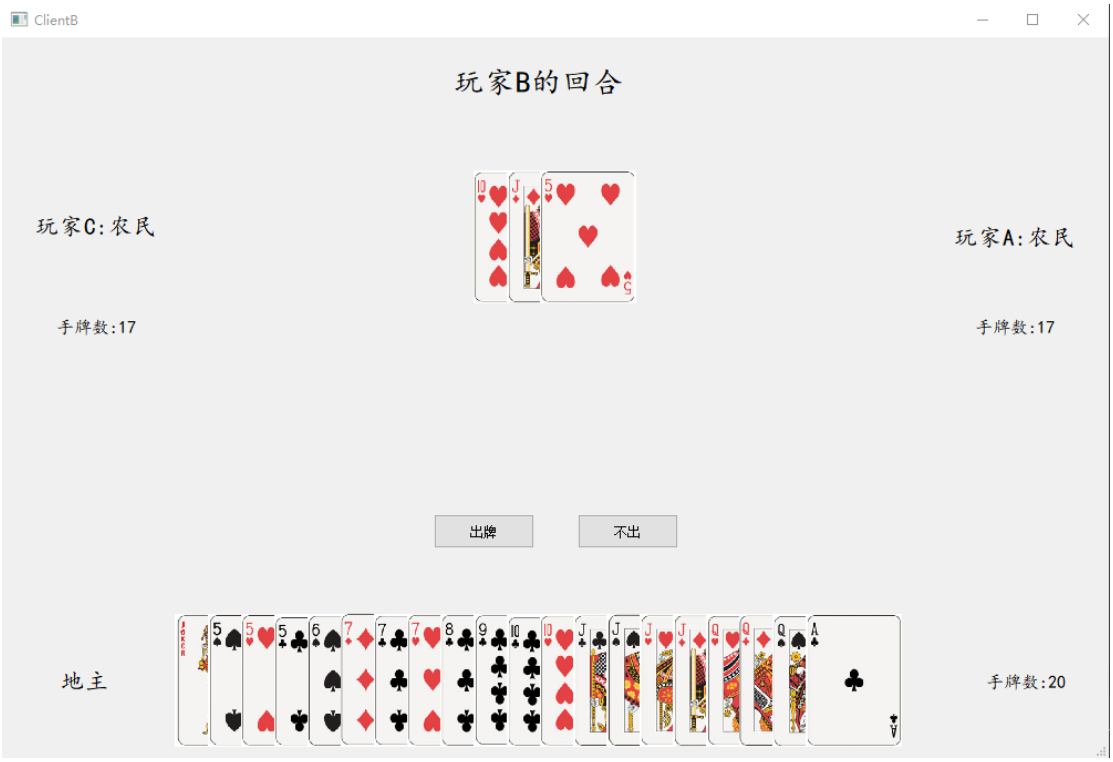
斗地主程序设计文档

1. 总体结构介绍

1.1 程序简介

本程序是实现斗地主卡牌对战游戏的程序。本程序基于 Qt 编写，基于 QMainWindow 类，实现了斗地主对战的基本功能。在程序中，三名玩家可以通过程序进行网络连接，并进行斗地主网络对战。玩家的抢地主、出牌、胜负结算等游戏流程都会在程序中进行结算和显示。本斗地主的规则采取 ppt 上介绍的游戏规则，玩家进入程序后，首先需要与其他两名玩家建立连接。建立连接后则开始选地主，地主产生后进行出牌对战，玩家可以选择出牌或者不出，最先出完手牌的阵营（地主或农民）胜利。游戏界面如下：



1.2 程序界面及功能介绍

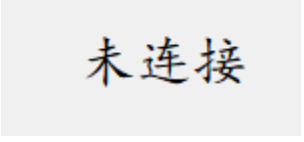
程序主界面如下：

程序界面由八个部分组成：消息提示栏、玩家身份栏、玩家手牌数栏、玩家出牌情况栏、地主牌、牌池、操作按钮区、手牌。

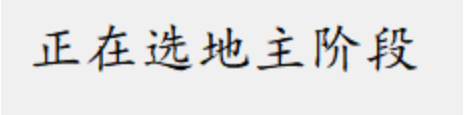
1.2.1 消息提示栏

消息提示栏用于提示玩家游戏当前进行到的阶段。

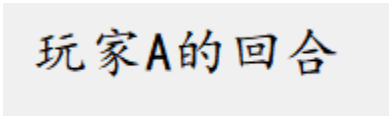
具体来说，当玩家间尚未建立连接时，消息提示栏会显示“未连接”。



当玩家建立连接后，进入发牌和叫地主阶段，则消息提示栏会显示“正在选地主阶段”。



当选出地主后，进入出牌阶段，消息提示栏则会提示当前出牌的是哪个玩家（A、B 或 C）。

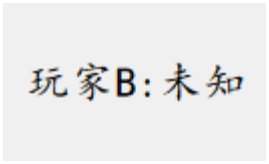


当游戏结束后，则会显示玩家的胜负情况（胜利或失败）。

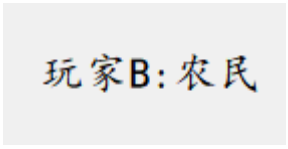
1.2.2 玩家身份栏

玩家身份栏提示了该玩家的身份。

当地主尚未选出时，此栏显示“未知”。



当地主选出后，此栏显示玩家的选出的身份（地主或农民）。

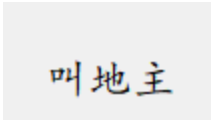


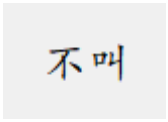
1.2.3 玩家手牌数栏

此栏显示玩家当前的手牌数，当玩家的手牌数变为 0 时，则宣告该玩家所在的阵营（农民或地主阵营，农民阵营有两位玩家，地主阵营有一位玩家）获胜。

1.2.4 玩家出牌情况栏

当游戏进行到选地主阶段时，此栏显示该玩家是否叫地主，“叫地主”代表该玩家叫了递增，“不叫”代表玩家选择跳过。





当游戏进行到出牌阶段时，此栏显示该玩家上次（也就是该玩家最近一次出牌阶段）的出牌情况，“出牌”代表该玩家出了牌，“不出”代表玩家选择跳过。

1.2.5 地主牌

地主牌显示发给地主的底牌，当地主尚未选出时，这三张牌出于背面朝上的状态。

当地主选出后，则处于正面朝上的状态。

1.2.6 牌池

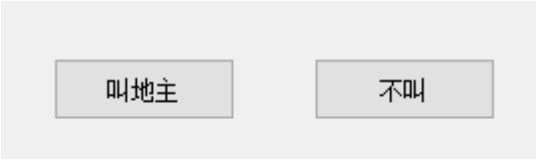
牌池展示了最近一次出牌的玩家的出牌情况，当有其他玩家（也有可能是同一个玩家）出牌后，则牌池的内容被最新的这次出牌覆盖。

1.2.7 操作按钮区

操作按钮区包含了玩家各个阶段的操作按钮。

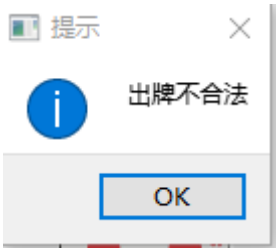
当玩家尚未连接游戏时，会出现开始连接的按钮，其他按钮出于隐藏状态。玩家点击开始连接程序会尝试与其他玩家建立连接。

当游戏处于叫地主阶段，并且进行到玩家操作的回合时，会出现叫地主/不叫的按钮。

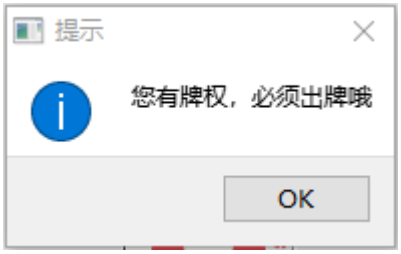


当游戏处于出牌阶段，并且进行到该玩家操作的回合时，会出现出牌/不出的按钮。

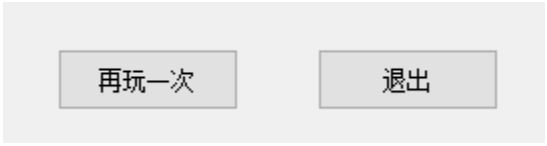
玩家选择出牌后则会将玩家选中的所有牌打出去，如果玩家选择的牌不合法，则会弹出提示框提示玩家，并且无法出牌。



玩家选择跳过后则会跳过玩家的回合，如果玩家手头有牌权，则会弹出提示框提示玩家，并且无法跳过。



当游戏出于结束阶段时，会显示再玩一次和退出按钮。如果所有玩家都选择再玩一次则重新开一局新的对局，否则断开连接退出程序。



1.2.8 手牌

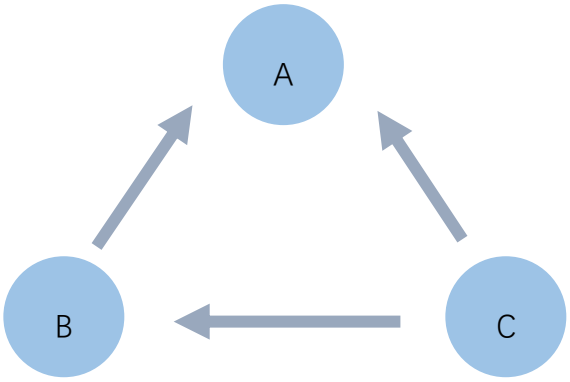
手牌展示了当前操作玩家所持有的手牌。当游戏进行到当前玩家的回合（出牌阶段）时，玩家可以在手牌中选择若干张牌，被选中的牌会比其他牌高度高一些。

手牌中所有的牌都自动进行了排序（从牌面小到大的顺序）。

1.3 程序整体结构及设计思路

本程序的主要通过三个进程互相进行网络连接实现网络对战。下面依次介绍本程序的网络结构和程序结构。

首先来看网络结构，本程序三个客户端之间的连接方式如下。



以上是 A、B、C 三个客户端间的连接方式。客户端 A 创建两个 QTcpServer 监听 18001 和 18002 端口，客户端 B 创建一个 QTcpServer 监听 18003 端口，通过 QTcpSocket 连接 18001 端口与客户端 A 进行连接，客户端 C 通过 QTcpSocket 连接 18002 和 18003 端口与客户端 A、B 进行连接。这样任何一个节点都可以向其他节点进行广播。

斗地主中的信息（身份、牌数、谁的回合）都是通过广播的方式进行同步的。那么诸如发牌、决定谁第一个选地主，谁最后当选地主这样的事情谁来完成呢？这写工作就主要由服务器来完成，在本斗地主中并没有单独的服务器进程，而是将服务器集成在客户端 A 中，并且开一个新的线程作为服务器线程，来处理这些事情。

综上所述，发牌、决定第一个选地主的人、决定谁当选地主等都有服务器线程来完成，服务器线程集成于客户端 A 中，并且通过 A 的套接字与 B、C 通信，通过信号与槽和客户端 A 的主线程通信。而同步牌池、谁的回合、牌数、叫地主情况、谁胜利等都是由 A、B、C 客户端（本游戏出牌顺序是 A->B->C->A）之间的通信实现。如 A 出牌后，就会向 B、C 广播它出了什么牌，剩下多少牌，而 B 接受到 A 出牌的信息，则更新界面信息，自动进入自己

的回合。至于胜负情况、叫地主情况（谁叫了谁没叫）等也是通过类似广播的方法实现，并且通过这样的方法推进回合（一个玩家结束回合，广播，下家接受到广播进入回合)。

接着我们来看一下程序结构。本程序可以分为用户交互和网络连接端两个部分。

用户交互主要负责图形的显示以及与用户的交互，主要由 MainWindow 类来负责。具体来说，用户的点击事件，程序内存中卡牌的显示，卡牌的交互效果，各种游戏信息的显示，按钮的显示以及按下按钮后的操作等都是由用户交互完成的。而卡牌的信息（花色、号码、选中状态、正面背面状态）等则是由 Card 类来进行实现。这部分的内容对于客户端 A、B、C 而言完全一样，代码一致。

网络连接部分主要负责网络信息的接收和发送，这部分不同客户端的稍有不同（与其连接方式有关系)，此外 A、B、C 的地位并不完全等价，比如 A 就是一个“弱中心”，B、C 选地主的结果需要集中传入 A 来处理，因此在网络连接部分不同客户端的实现略有区别。对于 A 而言，A 的服务器部分单独放到了一个线程里，主要通过将继承于 QObject 的 ServerWorker 类移动到另外的线程运行实现，线程间的通信主要通过信号与槽来实现。

除了上述介绍的内容，本程序在设计上仍然采用了状态管理的思路。MainWindow 类下有一个 QTimer 指针成员 updateSystemTimer 将会定时更新系统组件，根据系统当时的状态更新各个组件（如在准备阶段隐藏除了开始连接以外的所有按钮)。

状态介绍列举如下：

1、 Ready：表明游戏未开始，处于未连接状态

2、 Choosing：表明游戏处于选地主的状态。

3、 RoundA：表明游戏处于当前玩家的回合（注意：不是 A 玩家，而是当前玩家，例如对于客户端 B，RoundA 表示处于 B 玩家的回合)。

4、 RoundB：表明游戏处于当前玩家的下家的回合。

5、 RoundC：表明游戏处于当前玩家的上家的回合。

6、 End：表明游戏处于结束状态（已经有玩家出完牌)。

1.4 通信协议

下面将介绍本游戏的客户端的通信协议，首先是数据包的结构。

下面是数据包的结构示意图：

数据包主要由两个部分组成，包头存储了数据整体的长度，告诉对方这段数据的长度，包身就是数据的真正内容。数据内容主要是由字符串转换而来的字节流，传递的数据本质上是字符串。

这么设计的主要原因是为了解决粘包和分包的问题，每次先读取数据长度，如果剩下的部分大于数据长度，则把数据长度这么多的数据取出来读取，接着继续重复之前的操作，直到剩下的部分小于“数据长度”部分的长度或者“数据长度”之后的部分小于数据长度。

实现封包的函数位于 package.h 中（sendPackage()），而实现拆包的函数则在 mainwindow.cpp 中的信号读取槽函数（slotReadyReadB()和 slotReadyReadC()，分别对应当前客户端连上的另外两个客户端)。

接下来介绍传递信息的通信协议，载体为字符串。

1.4.1 "Call {1|2}"

当某个玩家按下“叫地主”或“不叫”按钮后，则会向其余玩家广播该信息，若广播的是"Call 0"表明该玩家选择不叫，"Call 1"表明该玩家选择叫地主。玩家接受到这条信息将会

根据它更新提示栏中的内容，如果发送这条信息的玩家是自己的上家，那么本家接下来就进入选地主阶段。

注：" Call {1|2}"中的{1|2}表示必选参数，即这条信息可以是"Call 1"或者是"Call 2"，下面的有关表示方法均类似地与命令行中的参数格式规范符合。

1.4.2 "HandsNum <Interger>"

当某个玩家收到手牌后或者出牌后（即手牌数改变后）会向其余玩家广播该信息，如广播"HandsNum 13"则表示自己当前的手牌数为 13。当玩家接受到该信息时则根据他更新手牌数提示栏的内容。

1.4.3 "Round"

当某个玩家接受到上家的出牌信息（下面会介绍）后，即将进入自己的回合时，会向所有玩家广播该信息，表明进入该玩家的回合。玩家接受到这条信息后，如果当前的转态不是End，则将状态设置为相应玩家的回合。

1.4.4 "Cards <CardString>…[End]"

当某个玩家出牌后则发出这条信息，告诉其他玩家他出了哪些牌。玩家收到该信息后，更新牌池为它出的牌，将牌权置为 false（其他玩家出了牌之后自己就不再有牌权了）。如果命令中不带有参数"End"则表明游戏此时没有结束，如果发信息的是自己的上家，那么接下来就进入本家的回合，如果参数中带有"End"则表明上家已经出完牌了，游戏结束，那么此时就不会进入本家的回合。

命令中的 CardString 表示卡牌的字符串代号（代号间以空格作为分隔符），其形式

为"<Joker>,<Number>,<Color>"，CardString 参数理论上可以任意多(当然实际上绝对不会超过 20 张牌)。下面是对各个参数的细节解释：

Joker 表示这张牌是不是王，N 表示不是王，R 表示大王，B 表示小王

Number 表示这张牌的号码，1-13 分别代表号码 3456…A2（注意与字面顺序不同，这是为了方便比较大小)，-1 表示大小王。

Color 表示这张牌的花色，N 表示无花色（大小王)，C 表示梅花，D 表示方块，H 表示红心，S 表示黑桃。

举个例子，"N,8,C"表示梅花 10，"R,-1,N"表示大王，"Cards N,8,C N,8,C End"表示发信息的玩家上个回合出了一对 8，并且他已经出完了牌（也就是他已经获胜了，这对 8 是他最后的牌)。

1.4.5 "Skip"

当某个玩家选择不出后向其余玩家广播该信息，告诉其余玩家自己的选择。当玩家收到这个信息后，更新提示栏，如果是上家给自己发的，那么就进入自己的回合。

1.4.6 "Success"

当某个玩家出完所有手牌后，向其他玩家广播该信息同时进入结束状态，告诉其余玩家自己胜利了。当玩家收到该信息后，进入结束状态，更新提示栏，若胜利的玩家与自己属于同一个阵营，则自己胜利，否则失败。

1.4.7 "CConnected"

当客户端 B 成功与客户端 C 连接上时，客户端 B 向客户端 A 发送此信息，告诉客户端

A 中的服务器线程客户端 C 已经成功连上客户端 B。

1.4.8 "Continued"

当游戏进入结束状态，客户端 B 与 C 选择再玩一次时向客户端 A 发送此信息，告诉客户端 A 中的服务器线程客户端 B 或客户端 C 选择重新开始。

1.4.9 "Hands <CardString>…"

客户端 A 中的服务器线程在生成完随机手牌后，将此信息发送给客户端 B 和客户端 C，告诉 B 和 C 他们的手牌。该信号的后半部分与 1.4.4 一致。也就是这里的 CardString 的格式（彼此之间也用空格相接）与 1.4.4"Cards <CardString>…[End]"中的 CardString 相符，此处不再赘述。客户端 B、C 在接受到这个信息之后更新手牌数、向其他玩家广播手牌数以及更新手牌区的显示。

1.4.10"LandLordBonus <CardString>…"

客户端 A 中的服务器线程在生成完随机手牌后，将此信息发送给客户端 B 和客户端 C，告诉 B 和 C 三张地主底牌。该信号的后半部分与 1.4.4 一致。也就是这里的 CardString 的格式（彼此之间也用空格相接）与 1.4.4"Cards <CardString>…[End]"中的 CardString 相符，此处不再赘述。客户端 B、C 在接受到这个信息之后更新地主底牌区，但同时把三张底牌设置为背面朝上的状态，因此在用户界面中这三张牌仍然是不可见的。

1.4.11"ChooseLandLord"

客户端 A 中的服务器线程随机决定出谁第一个叫地主后，即向该客户端发送此信息（若

这个人是玩家 A，则通过信号槽机制告诉客户端 A 中的主线程)。接受到该信息的客户端将叫地主/不叫的按钮设置为可见并开始叫地主。

1.4.12"LandLord {A|B|C}"

客户端 A 中的服务器线程根据三位玩家的叫地主/不叫反馈计算出谁是地主后，则向所有玩家广播该信息，告诉玩家谁是地主。接受到该信息的玩家更新界面中每个玩家的身份，初始化信息提示栏，曝光每张地主牌。如果自己是地主，那么就把地主牌加入自己的手牌。

1.4.13"Restart"

当三个客户端都选择再玩一次时，客户端 A 中的服务器线程会将此信息发送给每个客户端，接受该信息的客户端则将所有的变量和标签进行清空和初始化，将状态设置为 Choosing，重开一局，进入选地主阶段。

1.5 客户端流程介绍

1.5.1 准备阶段

准备阶段三个客户端初始化各自成员变量：包括 status（系统状态)，isAuthor（牌权)，各种 label、牌池、地主牌、手牌以及初始化系统更新定时器 updateSystemTimer 并且启动定时。

特别地，对于客户端 A，客户端 A 集成了服务器线程。服务器线程与客户端 C、B 之间的通信主要是通过客户端 A 的套接字实现的，而服务器线程与客户端 A 的通信则是通过信号与槽机制实现的。在准备阶段，首先建立服务器线程与客户端之间的信号与槽，接着启动服务器线程（由于此时还未建立连接，因此这个线程什么也不会做)。

1.5.2 连接及发牌阶段

当某个客户端的用户点击“开始连接”时，程序即刻建立套接字并开始尝试连接：客户端 A 监听 18001 和 18002 端口，客户端 B 监听 18003 端口，并尝试连接 18001 端口，客户端 C 尝试连接 18002 和 18003 端口。

当三个客户端互相连接成功并且完成接受信息的信号槽连接后，随机跳转至游戏界面，发牌，进入选地主阶段。

具体过程为，当客户端 B、C 连接上之后，客户端 B 会向客户端 A 发送"CConnected"，客户端 A 接受到该信号后随即记录成 flag。当客户端 A 的定时器 updateSystemTimer 检测（定时激发槽函数，在槽函数内进行检测）到客户端 B、C 之间连接成功（查看 flag），同时检测到客户端 B、C 已经连接上 A（可以通过 QTcpSocket 的 state()函数查看），这时则会将 status 设置为 Choosing（选择阶段），同时发送 connectedOk(QTcpSocket*, QTcpSocket*, QList<Card>* , QList<Card>*) 信号。当该信号发出后，对应的槽函数 choosingLandlord(QTcpSocket*, QTcpSocket*, QList<Card>* , QList<Card>*)执行。特别地，这个槽函数位于服务器线程中，因此这个槽函数将会在服务器线程中执行，这就进入到服务器处理的阶段。该槽函数首先会进行随机发牌，并且将各个玩家的手牌以及地主牌通过信号的方式发送给客户端 B、C，而直接将手牌和地主牌传给客户端 A（客户端 A 的主线程在发送信号以及调用服务器线程槽函数时传入了存储手牌、地主牌的 QList 指针，故服务器线程可以直接将对应的卡牌通过指针放入主线程的变量中）。客户端 B、C 在接受到手牌后则会自动进入 Choosing 状态。接着该槽函数会随机指定一个人开始选地主，并通过发送 "ChooseLandLord"信号告诉那个玩家由他开始选地主，如果是客户端 A 开始选地主，那么则通过发送 chosingDone(bool)信号激发 updateCardCoord(bool)（bool 值代表是否由客户端 A 开始选地主）进行处理，告诉客户端 A 的主线程由他/不由他开始选地主（如果是客户

端 A 开始选，那么 A 的主线程将会将叫地主/不叫的按钮设置为可见，否则不操作)。

1.5.3 选地主阶段

根据前文的叙述，当某个玩家接受到"ChooseLandLord"的信号后，则会将其叫地主/不叫的按钮设置为可见（如果是客户端 A 开始选地主，那么服务器线程将会通过信号槽的方式通知他将叫地主/不叫的按钮设置为可见)。因此这名玩家就可以与界面上的叫地主/不叫按钮交互，当玩家点击叫地主/不叫，客户端会更新状态，同时会将叫地主/不叫的决定广播给其余玩家（其余玩家在接受到广播后，会将发信息这名玩家的决定记录下来，并且更新界面)。当这名玩家的下家收到他的决定时，下家便会更新自己的状态同时进入自己的选地主阶段（将叫地主/不叫的按钮设置为可见)。

客户端 A 通过自己记录的叫地主决定的值的定时检测（通过定时器 updateSystemTimer 实现）三名玩家的决定，当检测到三名玩家全部做出决定后，随即发送 chooseOK(QTcpSocket*, QTcpSocket*,QString,QString,QString, int*, bool* , QList<Card>*, QList<Card>*)信号。当信号发出后，服务器线程中的 determineLandLord(QTcpSocket*, QTcpSocket*,QString,QString,QString, int*, bool* , QList<Card>*, QList<Card>*)槽函数随即执行。此槽函数将会根据三名玩家的选择决定最终的地主，并将谁是的地主的信息通过发送信息的方式告诉客户端 B、C，通过信号 determineDone(QString, QString, QString)告诉客户端 A 的主线程。客户端 B、C 在接受到信号会更新界面上玩家的身份、自己的手牌（地主可以获得额外的三张牌)、状态（切换到地主出牌的回合)、牌权（地主拥有牌权)、地主底牌（地主底牌在选出地主后会变为正面朝上的状态)。客户端 A 则通过信号 determineDone(QString, QString, QString) 对应的槽函数 dealLandLordResult(QString, QString, QString))来做对应的更新。

至此，服务器线程的功能就已经完成了。

1.5.4 出牌阶段

当每个客户端接受到谁是地主的信息后，会根据这个信息判断当前是谁的回合（地主先进入出牌回合）。若地主正是当前的客户端，则该客户端进入自己的回合，并将其出牌/不出的按钮设置为可见。因此这名玩家就可以与界面上的出牌/不出按钮交互（当然拥有牌权时选择不出是无效的），当玩家点击出牌/不出并且通过合法测试，客户端会更新状态，同时会将出牌/不出的决定（如果选择出牌，还会发送自己出了什么牌以及当前手牌数）广播给其余玩家。其余玩家在接受到广播后，会更新当前界面，更新牌池以及出牌玩家的手牌数。当这名玩家的下家收到他的决定时，下家便会更新自己的状态同时进入自己的出牌阶段并向其余玩家广播目前回合是自己的回合，当其他玩家接受到广播后则会将 status 更新为其回合的状态。下家接着会重复其上家做的事情，由此形成了一个闭环的循环。该循环保证顺序与叫地主一致，皆为 A->B->C->A，该循环将会持续直到一位玩家出完牌。

至于牌权，当一位玩家出牌后则自动获得牌权（地主一开始出牌前就有牌权），当到达该玩家的下一个回合前，有任意一个玩家出了牌，则该玩家失去牌权，如果一圈下来都没有玩家出牌，则该玩家仍然拥有牌权，则该玩家此时无法选择不出（有牌权选择不出是不合法的）。

1.5.5 结算阶段

出牌阶段按照 A->B->C->A 的形式不断循环，直到某位玩家打完手中的所有牌，在他最后一次出牌后，会向其他玩家发送"Success"信号，发送自己的出牌信息（附加上"End"），如"Cards N,8,C N,8,C End"，并且进入结束状态（即 status=End）。接受到信息的客户端随即进入结束状态（即 status=End），显示游戏结果。特别地，当下家接受到手牌信息并且检测到末尾有一个"End"时，虽然也会更新牌池而不会进入自己的回合。至此所有玩家已经进入

结束阶段。此时每个玩家界面上出现再来一次/退出的按钮，若玩家点击退出，则退出程序，若玩家点击再来一次，则向客户端 A 发送信息"Continue"（客户端 A 点击再来一次则直接记录下来即可）。当客户端的定时器 updateSystemTimer 检测到三名玩家都选择再来一次后，则向客户端 C、B 发送信号"Restart"，初始化各个变量到初始状态，将状态切换为 Choosing，发送 connectedOk(QTcpSocket*, QTcpSocket*, QList<Card>* , QList<Card>*)信号开始一局新的对局。其他客户端在收到"Restart"信号后则初始化各个变量到初始状态，将状态切换为 Choosing。至此，新的一局开始，客户端又从发牌阶段重新开始。

1.6 规则设计流程介绍

本斗地主采取的叫地主规则与 ppt 上一致，最后一个叫地主的人获得地主，如果没人叫就开始选的那个人成为地主。地主当选后获得三张地主牌，地主牌向所有玩家公示。本斗地主的采取的出牌规则与维基百科中介绍的出牌规则一致。在用户选择出牌后，程序会根据用户选择要出的牌以及牌池中的牌（上一个出牌玩家出的牌）进行判断用户的出牌是否合法。合法性判断分为两种情况，一种是用户有牌权的情况，另一种是用户没有牌权的情况。

1.6.1 有牌权的情况

有牌权的情况下调用 Card 类的静态函数 static bool isOK2Submit(QList<Card> my_cards) 进行判断，其中 my_cards 为用户出的牌的列表。有牌权时，只要用户出的牌符合标准牌型即可打出。下面是判断流程。

若 my_cards 中只有一张牌，那么用户的出牌是合法的，这是单张的情况。

若 my_cards 中有两张牌，且一张是大王，一张是小王，那么用户的出牌是合法的，这是王炸的情况。

若 my_cards 中有四张牌，并且这四张牌的数字相等，那么用户的出牌是合法的，这是炸弹的情况。

若 my_cards 中有两张牌，并且这两张牌的数字相等，那么用户的出牌是合法的，这是对子的情况。

若 isThree(my_cards)的返回值不为-1，那么用户的出牌是合法的，这是三带的情况。下面介绍 isThree 函数的接口。

int isThree(QList<Card> cards, int *num = NULL)

如果 cards 的大小不在[3,5]之间，直接返回-1，表示 cards 不构成三带。接着遍历 cards，记录其中重复出现次数为 3 的牌以及重复出现次数为 2 的牌，若不存在三张一样的牌，直接返回-1，存在的话就把这三张牌的数字存储在 num 指针中。若 cards 的大小为 3，说明是三带零，返回 0；若 cards 的大小为 4，说明是三带一（四张一样的牌算炸弹而非三带一）；若 cards 的大小为 5，那么看 cards 中是否有一对牌，若有一对牌，则为三带二，返回 2，否则不合法返回-1。

若 isFour(my_cards)的返回值不为-1，那么用户的出牌是合法的，这是四带的情况。下面介绍 isFour 函数的接口。

int isFour (QList<Card> cards, int *num = NULL)

如果 cards 的大小既不是 6 也不是 8，说明 cards 不构成四带二或者四代二对，返回-1。接着遍历 cards，记录其中重复出现次数为 4 的牌以及重复出现次数为 2 的牌，若不存在四张一样的牌，直接返回-1，存在的话就把这四张牌的数字存储在 num 指针中。若 cards 的大小为 6，说明是四带二，返回 2；若 cards 中存在一对牌，说明是四带二对，返回 4，其他情况返回-1（其他情况只有可能是四张一样的牌和两张不一样的牌）。

若 isShunZi(my_cards)的返回值不为 0，那么用户的出牌是合法的，这是顺子的情况。下面介绍 isShunZi 函数的接口。

int isShunZi(QList<Card> cards)

如果 cards 的大小不在[5,12]之间，直接返回 0，表示 cards 不构成顺子。接着遍历 cards，若发现有“2”（如“梅花 2”）则直接返回 0，表明不是顺子，接着查看 cards 排序后是否依次递增 1，如果是则构成顺子，返回 1，否则返回 0。

若 isShuangShun(my_cards)的返回值不为 0，那么用户的出牌是合法的，这是双顺的情况。下面介绍 isShuangShun 函数的接口。

int isShuangShun(QList<Card> cards)

如果 cards 的大小不在[6,20]之间或者不是双数，直接返回 0，表示 cards 不构成双顺。接着遍历 cards，若发现有“2”（如“梅花 2”）则直接返回 0，表明不是双顺。接着查看 cards 排序后是否下标为奇数的牌与前一张牌相同，下标为偶数的牌比前一张牌大一（下标从 0 开始），如果是则返回 1，表明是双顺，否则返回 0，表明不是双顺。

若 isPlane(my_cards)的返回值不为-1，那么用户的出牌是合法的，这是飞机的情况。下面介绍 isPlane 函数的接口。

int isPlane(QList<Card> cards, int *num = NULL)

飞机的判断略为复杂，首先对 cards 进行去重，把去重的结果另外储存。接着遍历去重的数组，寻找 cards 中相应元素的出现次数，把所有出现过三次且号码不为 2 的卡牌另外存起来，这是潜在的三条卡牌。如果这些三条的个数不为 2，直接返回-1，不构成飞机。接着从排序后三条中找出最长的依次递增 1 的子序列(若有多个等长的子序列,取靠后的一个)。若子序列的长度不足 2，返回-1，不构成飞机。否则从 cards 抽出剩下的部分（例如 cards 为 333444555678，子序列为 345，则从 cards 抽出 678)。对剩下的部分进行分析，若剩下的部分长度为 0，说明是无翼飞机，返回 0；若剩下的部分长度等于子序列的长度，说明全为三带一，小翼飞机，返回 1；若剩下的部分排序后两两成对，即下标为奇数的牌与前一张牌相同，且剩下部分长度正好是子序列长度的两倍，说明全为三带二，大翼飞机，返回 2；其他情况返回-1，不构成飞机。

只要命中上述任何一个“若”的情况则说明出牌合法，isOK2Submit 返回 1，否则返回 0，不合法。

1.6.2 无牌权的情况

无牌权的情况下调用 Card 类的静态函数 static bool isOK2Submit(QList<Card> my_cards, QList<Card> opponent_cards) 进行判断，其中 my_cards 为用户出的牌的列表，opponent_cards 为上一个出牌的玩家出的牌。没有牌权时，用户出的牌不仅要符合标准牌型，而且还要比 opponent_cards 大。下面是判断流程，返回 1 说明合法可以出，返回 0 说明不合法或大不过，不可以出。

首先考虑王炸的情况。

若 my_cards 的大小为 2 且一张大王一张小王，无论如何一定会比 opponent_cards 大（王炸只有一个），返回 1。

若 opponent_cards 为王炸，则返回 0，本家不可能大得过。

接着考虑炸弹的情况。

若 my_cards 大小为 4，四张牌数字一样，若 opponent_cards 不是炸弹，则直接返回 1（炸弹比非炸弹王炸大）。若 opponent_cards 也是炸弹，就看谁的单牌大，若己方单牌大，返回 1，否则返回 0。

若 opponent_cards 为炸弹，my_cards 不是炸弹也不是王炸（如果是的话肯定会命中前面的情况），返回 0。

最后考虑其他牌型的情况，即 my_cards 和 opponent_cards 中都没有炸弹和王炸。

若 opponent_cards 为单张，本家不是单张，返回 0；对面出大王，返回 0（没有比大王更大的单张），对面出小王，返回本家==大王（伪代码），对面没有王，本家有王，返回 1，

否则返回本家单张号码>对面单张号码（等于的情况也算大不过）。

若 opponent_cards 有两张牌，本家不是一对，返回 0，否则返回本家单张号码>对面单张号码（因为两张牌号码一样，所以比较任意一张牌都可以）。

若 isShunZi(opponent_cards)返回非 0（对面是顺子），则如果本家也是顺子，并且本家牌数与对面一致，本家的第一张牌大于对面的第一张牌则返回 1，否则返回 0。

若 isShuangShun(opponent_cards)返回非 0（对面是双顺），则如果本家也是双顺，并且本家牌数与对面一致，本家的第一张牌大于对面的第一张牌则返回 1，否则返回 0。

若 isThree(opponent_cards)返回值不是-1（对面是三带），则如果本家也是一样的三带（函数返回值相等，指都是三带一或都是三带二），并且本家的三带部分大于对面，则返回 1，否则返回 0。

若 isFour(opponent_cards)返回值不是-1（对面是四带），则如果本家也是一样的四带（函数返回值相等，指都是四带二或都是四带二对），并且本家的四带部分大于对面，则返回 1，否则返回 0。

若 isPlane(opponent_cards)返回值不是-1（对面是飞机），则如果本家也是一样的飞机（函数返回值相等，指都是无翼飞机或者小翼飞机等），并且本家的三带部分大于对面，则返回 1，否则返回 0。

以上的所有判断一定都能命中其中之一，若没有，则说明 opponent_cards 不属于标准牌型，也就是游戏出了 bug，此时在 debug 控制台输出"上家牌型不合法！"，返回 0。

2. 主要接口介绍

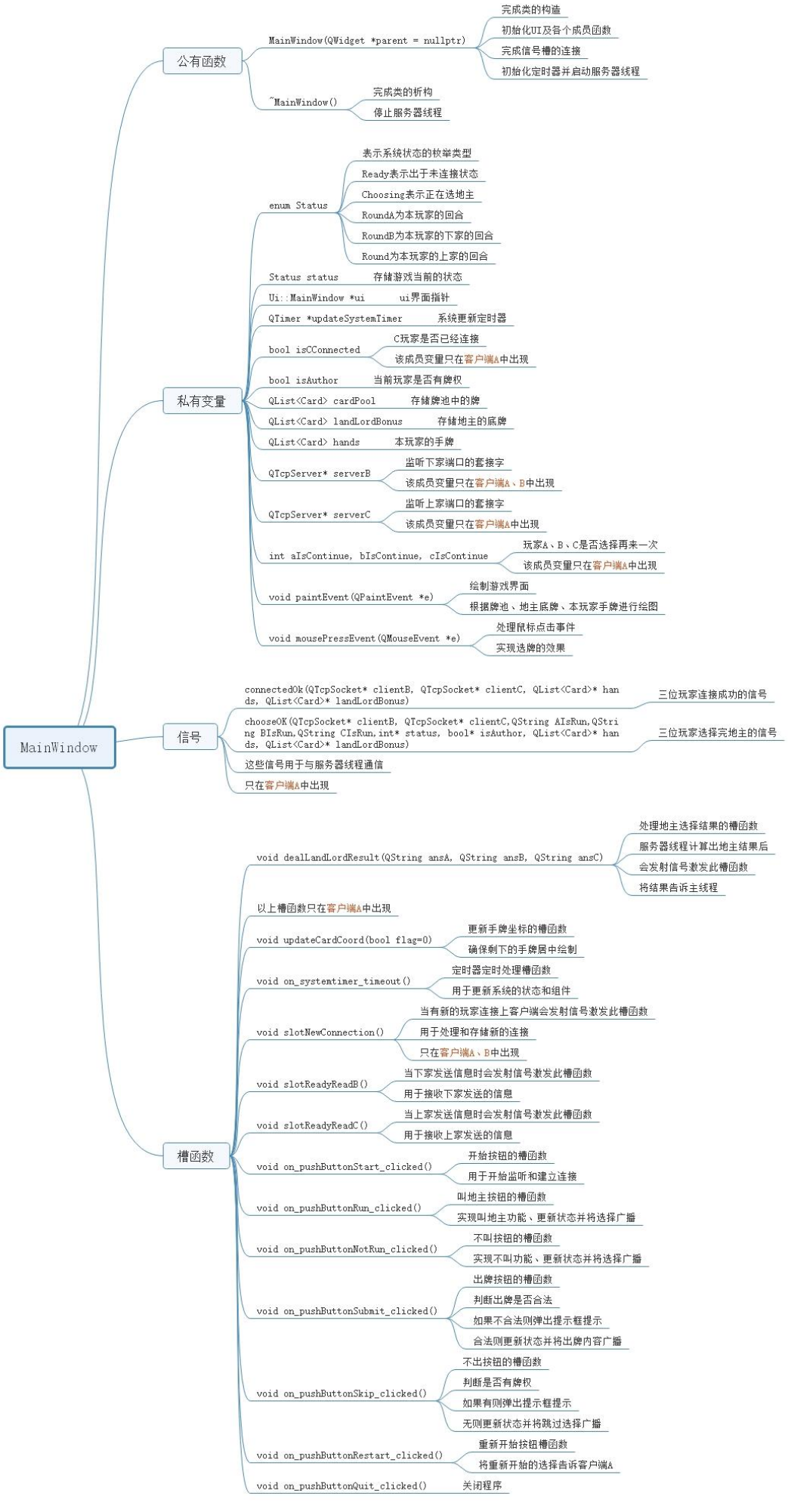
2.1 主要接口简介

本程序的主要接口为 MainWindow 类、Card 类、ServerWorker 类和全局函数

sendPackage。其中 ServerWorker 类只在客户端 A 中使用，是服务器线程的实现类。而其他的接口在三个客户端中都有使用。MainWindow 类是图形界面与交互的主要实现类，也是整个客户端的核心类，Card 类的实例变量、ServerWorker 类的实例变量均作为 MainWindow 类的成员变量发挥作用。Card 类实现了卡牌的数据结构、读写接口以及判断出牌合法性的接口。sendPackage 函数完成了对字节数组的封包和发送，在包头加入了数据长度，解决粘包和分包的问题。

2.2 MainWindow 类介绍

MainWindow 类是程序主界面的实现类，也是整个图形界面显示与交互的核心实现类。该类继承自 QMainWindow，具备 QMainWindow 的一切基本接口。在此基础上，MainWindow 还额外定义了一些成员变量与成员函数，便于更好地与用户进行交互，处理用户的输入以及保存游戏过程中产生的数据以及向其他客户端发送数据。以下是该类的整体结构图，结构图中只标明了 MainWindow 类相比 QMainWindow 新增加的定义与重载的函数。

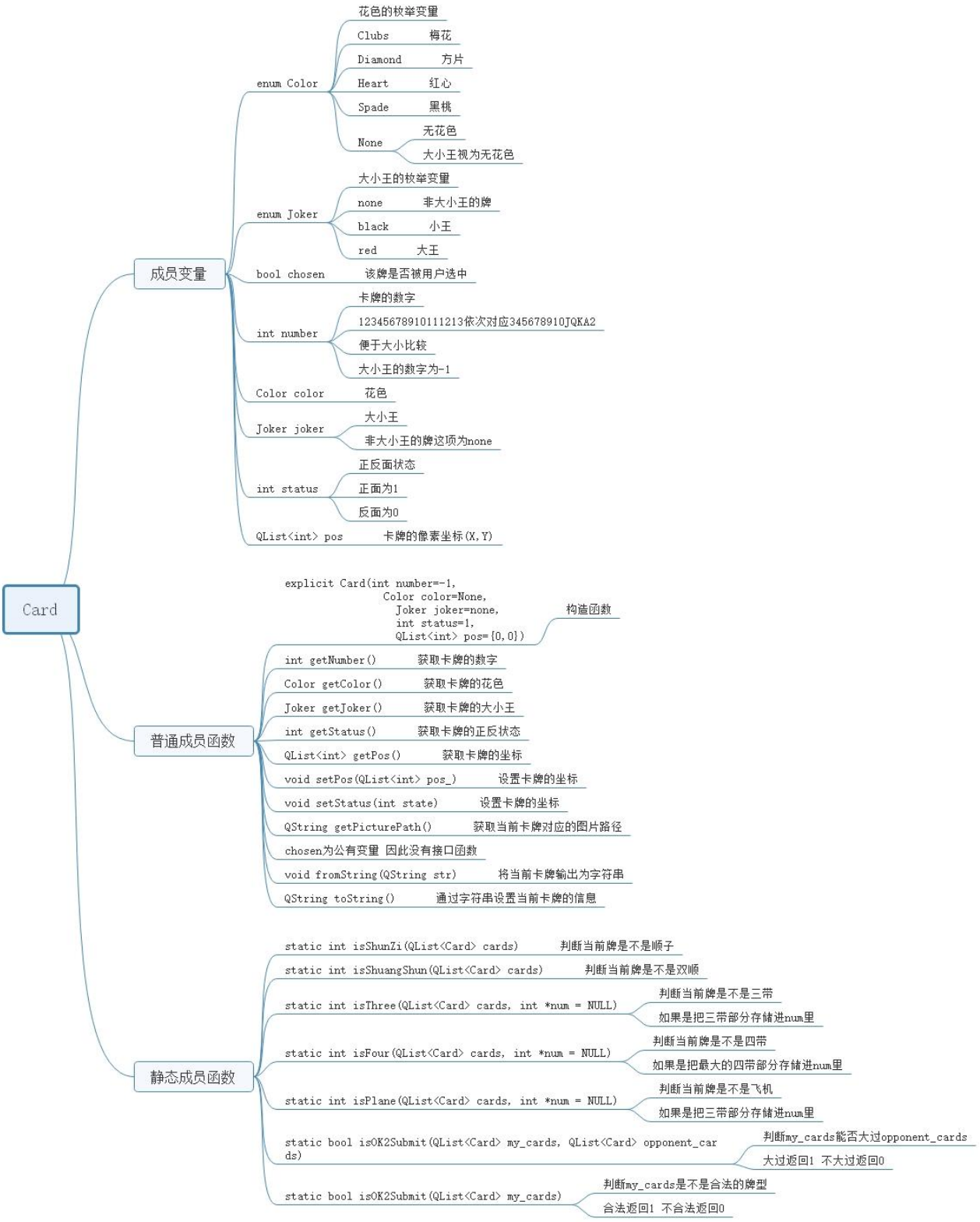

MainWindow
公有函数
MainWindow(QWidget *parent = nullptr)
完成类的构造
初始化UI及各个成员函数
完成信号槽的连接
初始化定时器并启动服务器线程
~MainWindow()
完成类的析构
停止服务器线程
私有变量
enum Status
表示系统状态的枚举类型
Ready表示出于未连接状态
Choosing表示正在选地主
RoundA为本玩家的回合
RoundB为本玩家的下家的回合
Round为本玩家的上家的回合
Status status 存储游戏当前的状态
Ui::MainWindow *ui ui界面指针
QTimer *updateSystemTimer 系统更新定时器
bool isCConnected
C玩家是否已经连接
该成员变量只在客户端A中出现
bool isAuthor 当前玩家是否有牌权
QList<Card> cardPool 存储牌池中的牌
QList<Card> landLordBonus 存储地主的底牌
QList<Card> hands 本玩家的手牌
QTcpServer* serverB
监听下家端口的套接字
该成员变量只在客户端A、B中出现
QTcpServer* serverC
监听上家端口的套接字
该成员变量只在客户端A中出现
int aIsContinue, bIsContinue, cIsContinue
玩家A、B、C是否选择再来一次
该成员变量只在客户端A中出现
void paintEvent(QPaintEvent *e)
绘制游戏界面
根据牌池、地主底牌、本玩家手牌进行绘图
void mousePressEvent(QMouseEvent *e)
处理鼠标点击事件
实现选牌的效果
信号
connectedOk(QTcpSocket* clientB, QTcpSocket* clientC, QList<Card>* hands, QList<Card>* landLordBonus)
三位玩家连接成功的信号
chooseOK(QTcpSocket* clientB, QTcpSocket* clientC,QString AIsRun,QString BIsRun,QString CIsRun,int* status, bool* isAuthor, QList<Card>* hands, QList<Card>* landLordBonus)
三位玩家选择完地主的信号
这些信号用于与服务器线程通信
只在客户端A中出现
槽函数
void dealLandLordResult(QString ansA, QString ansB, QString ansC)
处理地主选择结果的槽函数
服务器线程计算出地主结果后
会发射信号激发此槽函数
将结果告诉主线程
以上槽函数只在客户端A中出现
void updateCardCoord(bool flag=0)
更新手牌坐标的槽函数
确保剩下的手牌居中绘制
void on_systemtimer_timeout()
定时器定时处理槽函数
用于更新系统的状态和组件
void slotNewConnection()
当有新的玩家连接上客户端会发射信号激发此槽函数
用于处理和存储新的连接
只在客户端A、B中出现
void slotReadyReadB()
当下家发送信息时会发射信号激发此槽函数
用于接收下家发送的信息
void slotReadyReadC()
当上家发送信息时会发射信号激发此槽函数
用于接收上家发送的信息
void on_pushButtonStart_clicked()
开始按钮的槽函数
用于开始监听和建立连接
void on_pushButtonRun_clicked()
叫地主按钮的槽函数
实现叫地主功能、更新状态并将选择广播
void on_pushButtonNotRun_clicked()
不叫按钮的槽函数
实现不叫功能、更新状态并将选择广播
void on_pushButtonSubmit_clicked()
出牌按钮的槽函数
判断出牌是否合法
如果不合法则弹出提示框提示
合法则更新状态并将出牌内容广播
void on_pushButtonSkip_clicked()
不出按钮的槽函数
判断是否有牌权
如果有则弹出提示框提示
无则更新状态并将跳过选择广播
void on_pushButtonRestart_clicked()
重新开始按钮槽函数
将重新开始的选择告诉客户端A
void on_pushButtonQuit_clicked()
关闭程序

客户端 A、B、C 各自都有自己的 MainWindow 类，其 MainWindow 类除了有些成员变量和成员函数不完全一致（区别已经在结构图中给出）外，其 void on_pushButtonStart_clicked()、slotReadyReadB()和 slotReadyReadC()的实现也有些不同，因为每个客户端建立的连接、收到和发送的信息的种类和信息不完全等价，其他的代码三个客户端完全一致，最大程度复用了代码。特别地，服务器线程设在客户端 A，客户端 B、C 通过与客户端 A 的连接与服务器线程通信，而客户端 A 的主线程则通过信号与槽的机制与服务器线程连接。

本程序定义了表示状态的枚举变量，用于状态管理。当程序处于某个状态时会有一些固定的显示和操作，系统更新定时器就是定时根据系统当前的状态更新系统的组件。

2.3 Card 类介绍

在游戏中，我们需要正确的绘制卡牌需要知道卡牌的号码、花色、坐标、被选中的状态以及是否正面朝上，因此这里将卡牌的数据结构及相关结构独立成一个 Car 类进行管理，并且定义了相关方便的接口进行维护。该类继承自 QObject，且定义了 Q_Object 宏。以下是该类的整体结构图。



成员变量部分存储一个卡牌的基本信息。普通成员函数则定义了卡牌的一些接口，值得注意的是 void fromString(QString str)和 QString toString()函数。从前面的介绍可以看出，卡牌对象在通信时是以字符串的形式传输的，因此我们发信息时需要将卡牌对象转成字符串，

接受信息时需要根据接收到的字符串初始化卡牌。而这两个函数就为此提供了很好的接口。例如，将"N,8,C"传入 void fromString(QString str)，就能将当前的卡牌对象初始化为梅花 10，反之调用一个梅花 10 卡牌对象的 QString toString()会返回"N,8,C"。静态成员变量部分主要是一些判断牌型时需要用到的函数，因为判断牌型与 Card 类关系密切却不需要绑定到某个实例对象，因此将其设置为静态函数。

2.4 ServerWorker 类介绍

服务器独立运行与一个线程，而服务器与客户端 A 的通信又是由信号槽实现的。这里定理了一个 ServerWorker 类来实现服务器需要完成的功能，并且定义了相关与客户端 A 通信的信号槽。该类继承自 QObject，且定义了 Q_Object 宏。在实际使用时，会创建该类的实例对象，并将该类移动至其他线程运行，当客户端 A 收到信息需要进行服务器处理时，会通过信号槽告诉 ServerWorker，调用相应的槽函数进行处理。由于该类使用时在其他线程，因此可以确保所有的处理都不在主线程内。以下是该类的整体结构图。

2.5 其他全局函数

extern void sendPackage(QTcpSocket *sender, QByteArray sendByte)

接受一个 QTcpSocket 指针 sender 以及一个要传输的数据 sendByte。该函数会对 sendByte 进行封包，在其头部加入数据的整体长度，通过 sender 发送至指定的位置。

3. 总结

本程序充分利用 Qt 平台及其内置组件，借助网络编程思想，很好地实现了在线斗地主对战游戏。总的来说，界面简洁用户友好，代码编写有一定的结构性和可读性，各项功能实现得也比较完备。经过测试，本程序的功能基本符合预期，没有出现不符合预期的行为。