“网络斗地主”程序设计文档

计95 刘玉河 2019011560

主要类

类	作用
Poker	记录单张扑克牌的花色与牌点，重载比较运算符
Combo	所有牌型的基类，记录牌型对应序号
lobby : public QMainWindow	准备界面UI，建立连接以及信息收发
game : public QWidget	游戏界面UI,卡牌显示
card : public QPushButton	卡牌显示、选中

客户端工作流程

准备界面(lobby)

建立连接

1. 客户端首先创建新的QTcpServer，并尝试监听指定IP地址的6001端口

- 若监听6001端口成功，则成为“玩家A”，开始接受剩下两个客户端的连接信号，玩家A的客户端在游戏开始前以及游戏结束后承担主要的流程控制。
- 若监听失败，则尝试监听指定IP地址的6002端口
    - 若监听6002端口成功，则成为“玩家B”，向正在监听6001端口的A建立连接，并开始接受“玩家C”客户端的连接信号
    - 若监听失败，则成为“玩家C”，向监听6001端口的A以及监听6002端口的B建立连接

2. ABC彼此的连接建立成功后，将对应的QTcpSocket对象指针存至私有数组成员connectABC[3]中，为后续信号收发作准备
3. A向BC发送“玩家就绪，准备开始游戏”信号（详见通信协议部分）

开始游戏

1. A点击Start按钮，发送GAMESTART信号给BC，各客户端进入**游戏界面**

游戏界面(game)

1. A依次完成**决定顺序**、**发牌**、**询问是否叫地主**操作，并通过对应信息发送给BC
    - 决定地主时，A客户端创建一个新的进程lordthread，按照决定的顺序发送询问信息，每次发送后进行等待直到收到“叫地主/不叫”的信息，再发送下一个询问信号，直到三个玩家被询问完后决定地主，正式进入**出牌流程**
2. A通知地主开始出牌

3. 当一个客户端出牌时，发送包含“出牌者ID”、”出牌情况“、“剩余牌数”的信息给其他客户端，其他客户端根据出牌者ID判断是否轮到自己的回合
4. 当一个客户端选择“不出”时，发送包含“自身ID”以及“最后一个出牌者ID”的“不出”信息给其他客户端，每个客户端自身维护一个保存了最后出牌者ID的变量
5. 如果最后一个出牌者ID与自身ID相同，说明该玩家获得了牌权，此时“不出”按钮无效，该玩家必须出牌
6. 每当收到出牌信息后，客户端判断剩余牌数是否为0，若为0则游戏结束，根据牌数为0者的身份判断胜负
7. 游戏结束时，游戏界面显示“重新开始”按钮，点击后BC将发送重玩信息给A，当重玩人数等于3时，A再次发送GAMESTART信号给BC，重新开始游戏
8. 点击游戏界面的“退出游戏”按钮，客户端将直接断开连接并退出游戏。

通信协议

传输数据格式：

约定每次收发信息长度固定为100字符，从开头进行读取，后续剩余空间用’/’字符填充

信息的格式为“信息头+信息内容”，不同部分之间用空格分隔

- 玩家的id按照叫地主顺序为 0、1、2
- 扑克牌的格式为“花色+牌点”

含义	标志
spade	S
heart	H
club	C
diamond	D
joker	J
牌点：3-…-JQK-1-2	0-1-…-11-12
大王	13
小王	14

例：红心五：H2、方块J：D8、小王J13、大王J14

信息表

信息头Header (对应常数)	发送方/接收方	内容格式	含义
WAITFORSTART (1)	A / BC	/	三方建立连接成功，等待开始
STARTUP (2)	A / BC	/	开启游戏界面
ARRANGE (3)	A / BC	a_id b_id c_id	决定叫地主顺序以及出牌顺序
SERVE (4)	A / BC	id (poker)*17	发牌给第id号玩家
ASKFORLORD (5)	A / BC	id	询问第id号玩家是否叫地主
WANTLORD (6)	BC / A	id (bool)want	表明第id号玩家是否叫地主(want真为“叫”，假为“不叫”)
LORDCARDS (7)	A / BC	id (poker)*3	告知第id号玩家为地主，并公布三张地主牌
GAMESTART (8)	A / BC	/	游戏开始，请地主开始出牌
REPLAY (10)	BC / A	/	请求重新开始新的一局游戏
COMBO (11)	出牌者 / 其他人	id left len (poker)*len	第id号玩家出牌，还剩left张牌，出了len张牌及其内容
PASS (12)	不出者 / 其他人	id lastplay_id	不出者id，最后一个出牌者lastplay_id

规则设计思路

在斗地主游戏规则的实现中，主要涉及两个方面：

1. 选中牌的牌型判断
2. 跟牌判断

本程序主要靠Poker与Combo两个类，以及check.cpp中的canBeNext()，whatCombo()函数来实现以上两个方面。

Poker

```
class Poker {
	card_rank _rank;	//typedef int card_rank;	// 牌点
	suit _suit;		//enum suit{single,pair...}; // 花色（加上joker类型）
public:
	Poker();
	Poker(card_rank,suit);		// 根据花色、牌点构造Poker对象
	Poker(QString);			// 根据预设的文本格式构造Poker对象
	card_rank getRank() const {return _rank;}	// 返回牌点
	suit getSuit() const {return _suit;}		// 返回花色
	bool operator<(const Poker&) const;		 // 比较运算符
	bool operator==(const Poker&) const;
	QString string() const;				// 返回对应文本（如"H2","D11","S0"）
};
```

Combo

- 公有函数：

```
Combo(comboCat cat);		//构造函数
Poker* cardList() const;	//返回内含牌
bool canBeCompared(const Combo&) const; //判断两个牌型是否相同
bool biggerSuit(const Combo&) const;	//判断一般牌型、炸弹、王炸之间的大小关系
comboCat getCat() const;		//返回牌型对应序号
virtual bool sameKind(const Combo&) const {return false;};  //虚函数，判断同一牌型的不同类型（如三带一/三带二）
virtual bool greaterThan(const Combo&) const {return false;};//虚函数，进行同一牌型之间的比较
```

- 派生类：

Single	Pair	Triple
Straight	DStraight	QuartWDoub
Plane	Bomb	Jokers

以及充当空牌型的NotACombo

每个派生类需要各自构建自身所需的Poker数据成员、重载sameKind()和greaterThan()函数

牌型判断：Combo& whatCombo(Poker* cards,int len);

whatCombo()函数接受Poker数组以及长度，返回对应牌型的引用（基类Combo引用）

判断牌型思路：

1. 统计牌中以**单张、对子、三张、四张**形式出现的个数以及对应牌

   分别记个数为sing_cnt,pair_cnt,trip_cnt,quart_cnt

   将对应牌存放在sings[],pairs[],trips[],quarts[]数组中

2. 根据不同牌型对应的sing_cnt ， pair_cnt ， trip_cnt ， quart_cnt的值进行**分类判断**

3. 对于情况复杂的飞机需要加入额外的判断

4. 如果判断成功，返回对应牌型派生类的新的对象的引用

5. 如果判断失败，返回一个NotACombo()类对象的引用

以“三带二“和”连对“为例

```
 //三带二 包含一个对子、一个三张
if ((!sing_cnt) && (pair_cnt==1) && (trip_cnt==1) && (!quart_cnt))
{
      return *(new Triple(trips[0], 2, pairs[0]));
}
//连对 只包含3到10个对子
if ((!sing_cnt) && (pair_cnt >=3 && pair_cnt <= 10) && (!trip_cnt) && (!quart_cnt))
{
      if(isContinuous(pairs,pair_cnt)) {  // isContinuous()函数判断pairs数组中的牌是否连续
            return *(new DStraight(pairs, pair_cnt));
     }
}
```

跟牌判断 bool canBeNext(const Combo&,const Combo&);

```
bool canBeNext(const Combo &a, const Combo &b)  // 返回牌型a能否跟牌型b
{
      if (a.canBeCompared(b) && a.sameKind(b))    // 相同牌型且带牌（三带、四带、飞机）类型和数目相同
     {
            return a.greaterThan(b);                // 同类型牌型之间的比较
     }
      else
            return a.biggerSuit(b);                 // 不同牌型时，判断 一般<炸弹<王炸 的关系
      return false;
}

bool Combo::canBeCompared(const Combo& b) const{
      return _catog == b._catog;                  // _catog：牌型对应序号
}

bool Combo::biggerSuit(const Combo& b) const{
      if(b._catog == notacombo) return true;
      return (_catog >= bomb && _catog > b._catog);
}
```

```cpp
// sameKind、greaterThan 分别以“四带二”和“顺子”为例
bool QuartWDoub::sameKind(const Combo &b) const
{
	const QuartWDoub &b1 = dynamic_cast<const QuartWDoub &>(b);
	return isPair==b1.isPair;	// 判断带的是否都是两个单张或对子
}

bool Straight::greaterThan(const Combo &b) const
{
	const Straight &b1 = dynamic_cast<const Straight &>(b);
	return cards[0] > b1.cards[0];  // 只需要判断两副顺子牌点最小的牌的大小
}
```