扑克牌游戏项目设计说明

小组成员：

-
-
-
-

一、需求分析

我们希望实现一个直观的，易上手的扑克游戏，将“争上游”和“炸金花”作为参考对象，具体需求大致如下：

- 直观的 GUI 界面，用于与用户的交互，能显示4人的手牌与出牌情况，各玩家的状态（是否托管）
- 两种纸牌游戏的规则实现，能检查出牌是否合法、判断游戏胜利等，对于炸金花，另外判断下注
- 纸牌游戏的 AI 实现，用于托管或直接作为填充人数的方式
- 网络模块，用于局域网联机，应当保证传输质量，支持错误检测

二、总体设计

程序总体上分为4个模块：逻辑模块、网络模块、界面模块。逻辑模块负责游戏主逻辑实现及AI 策略设计及实现，网络模块负责网络层搭建，界面模块负责前端界面设计及实现。

具体任务为：

- 逻辑：作为后端完成游戏功能，设计合适的数据结构和类，
- 网络：提供信息的交互，实现局域网联机功能。
- 界面：通过 Qt 开发出一个友好的用户图形界面，最终结果应类似于 Win7 内置纸牌游戏的游戏界面。

在实际项目中，我们分配了两个人负责逻辑模块，分别完成“争上游”，“炸金花”的后端和对应的AI。一个人负责界面模块，一个人负责网络模块以及代码的整合。

三、系统模块说明

界面模块

基本介绍

借助qt creator进行界面设计，结合了后端接口实现交互功能，并保留了一定的网络接口

核心类说明：

主页面：

在main函数中调用，是整个游戏的入口，加入游戏是联机选项，选择创建游戏后会进入游戏的相关设置页面。

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    //connect(ui->pushButton,SIGNAL(clicked()),this,SLOT(on_pushButton_clicked()));
}
```

主页面的实现使用了qt creater的ui设计类，选择了栅格式的布局管理器，然后插入QButton、QLabel等组件，并通过修改样式表调整了组件的外观。使用spacer组件调整页面的布局结构，呈现出简单美观的开始页面。

选择页面

选择页面的设计基本和主页面类似。

1. 游戏选择

```
class GameChoose : public QMainWindow
{
    Q_OBJECT

public:
    explicit GameChoose(QWidget *parent = nullptr);
    ~GameChoose();

private slots:
    void on_pushButton_3_clicked();

    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

private:
    Ui::GameChoose *ui;
};
```

---

2. 模式选择

```
class ModeChoose : public QMainWindow
{
    Q_OBJECT

public:
    explicit ModeChoose(QWidget *parent = nullptr);
    ~ModeChoose();

private slots:
    void on_pushButton_3_clicked();

    void on_pushButton_2_clicked();

    void on_pushButton_clicked();

private:
    Ui::ModeChoose *ui;
};
```

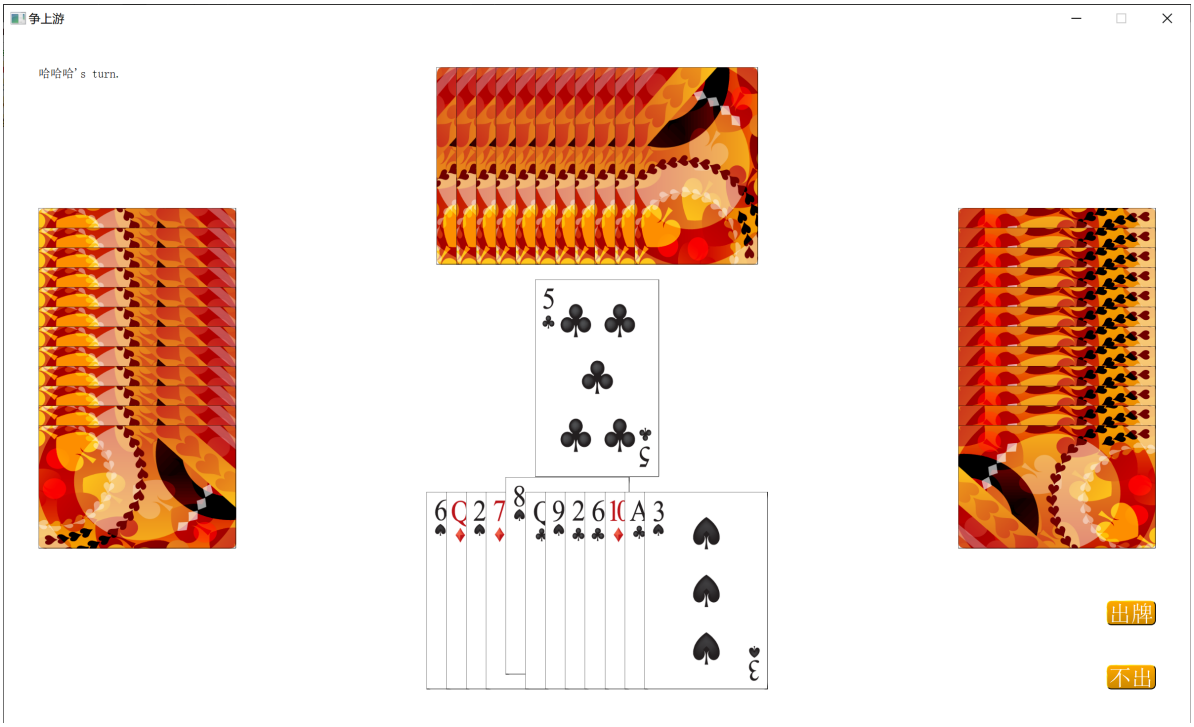
3. 人数选择

```
class NumChoose : public QMainWindow
{
    Q_OBJECT

public:
    explicit NumChoose(QWidget *parent = nullptr);
    ~NumChoose();

private slots:
    void on_pushButton_3_clicked();
    void on_pushButton_clicked();
    void on_pushButton_2_clicked();
    void on_pushButton_4_clicked();
    void on_pushButton_5_clicked();
    void on_pushButton_6_clicked();
private:
    Ui::NumChoose *ui;
};
```

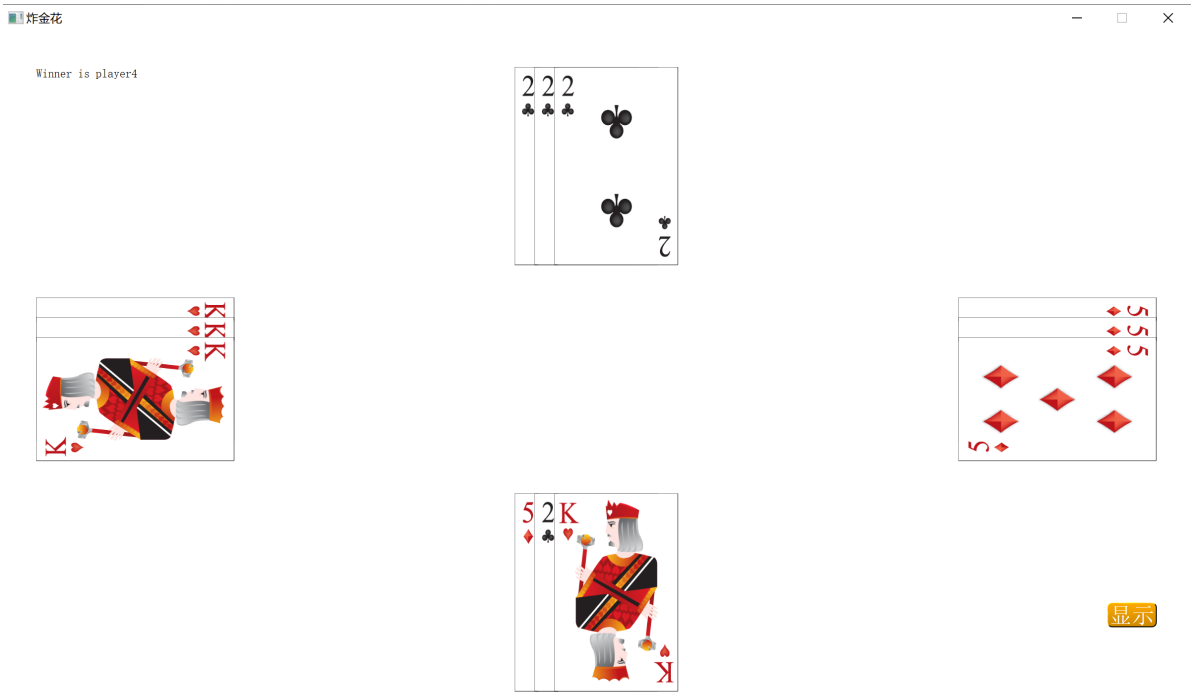
游戏页面类

游戏页面类有Game1和Game2两个，其中包括了界面的显示、游戏回合的控制、一个game类型的对象作为前后端的接口

```
class Game1 : public QMainWindow
{
    Q_OBJECT
public:
    void StartGame();//游戏开始
    explicit Game1(QWidget *parent = nullptr);
    void OneTurn();//一个回合
    void ShowCards();//所有卡牌显示
    void ShowOtherCards(int angle,int n);//其他玩家卡牌显示
    void ShowPreviousCards();//卡池显示
    void LoadPreviousCards();//将卡池的牌存入显示容器
    void LoadSelfCards();
    void LoadOtherCards();
private:
    vector<CardButton *> cards_in_hand;//手牌
    vector<vector<CardWidget *>> other_cards;//他人手牌
    vector<CardWidget *> previous_cards;//上一个人出的牌
    Game* newgame;//游戏主体，实现在后端
    QPushButton* sure;//出牌按钮
    QPushButton* pass;//过牌按钮
    QPushButton* gamestart;//游戏开始按钮
    QLabel* hintLabel;//提示标签

signals://信号
    void win(QString &str);
    void hint(QString &str);
public slots://槽
    void end(QString &str);
    void discard();
    void nondiscard();
    void changeHint(QString &str);
};
```




```
class Game2 : public QMainWindow
{
    Q_OBJECT
public:
    explicit Game2(QWidget *parent = nullptr);
    void StartGame();
    void ShowCards();//所有卡牌显示
    void ShowOtherCards(int angle,int n);//其他玩家卡牌显示
    void LoadSelfCards();
    void LoadOtherCards();
private:
    vector<CardWidget *> cards_in_hand;
    vector<vector<CardWidget *>> other_cards;
    FpfGame* newgame;
    QPushButton* sure;
    QPushButton* gamestart;
    QLabel* hintLabel;
signals:
    void win(QString &str);
    void hint(QString &str);
public slots:
    void end();
    void changeHint(QString &str);
};
```



交互说明

采用了信号和槽的方式实现交互，给需要交互的button绑定槽函数，包括自定义的cardbutton。

页面的切换、出牌、选牌、过牌、游戏开始等的交互全部都是通过信号和槽来实现。

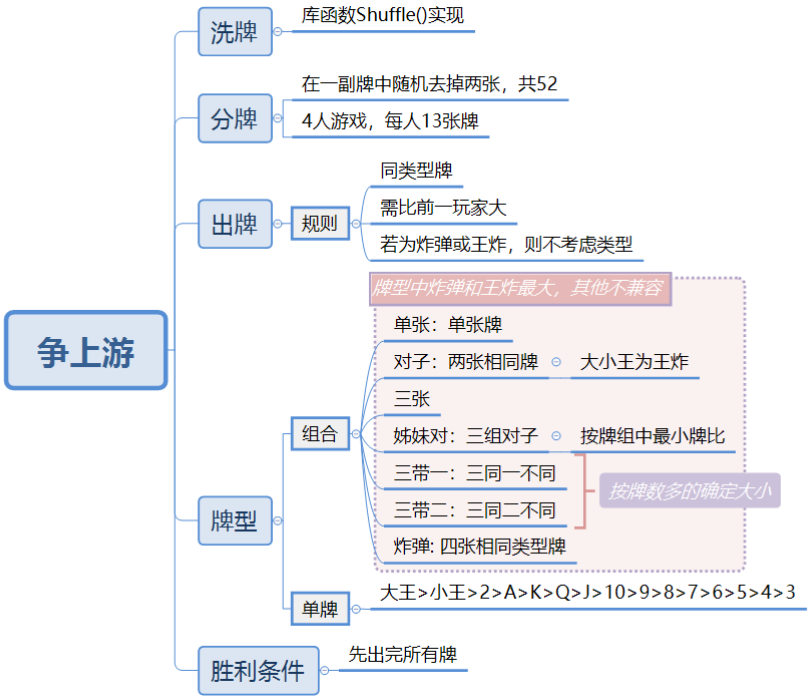
逻辑模块

基本介绍

逻辑模块实现了两种游戏，提供了后端接口，并设有具有一定智能的AI。

核心类说明

争上游



分别设计类来表示牌和牌组。

对于Card类，设计5种花色（Suit），分别为黑桃（SPADES），红心（HEARTS），方块（DIAMONDS），梅花（CLUBS），大小王（JOKER）。对于点数(Rank)，顺序为3，4，5，6，7，8，9，10，J，K，Q，A，2，小王，大王，同时设置虚拟点数FLAG最小以及MAXN最大作为哨兵，即用于后端逻辑判断。

对于Cards类，设计单张（SINGLE），对子（PAIRS），三张（TRIPS），姊妹对（THREEPAIRS），三带一（TREYS），三带二（FULLHOUSE），顺子（STRAIGNT），炸弹（BOMB）

```
class Card {
public:
    enum Suit { SPADES, HEARTS, DIAMONDS, CLUBS, JOKER };
    enum Rank { FLAG, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN,
KING, ACE, TWO, SJOKER, BJOKER, MAXN};
    Card() {};
    ~Card() {};
    Card(Suit suit, Rank rank) : suit(suit), rank(rank);
    void print() const;
    void set(char rank, char suit);
    void select(int status);
    const Suit getSuit();
    const Rank getRank();
    const int getStatus() const;
    bool operator>(const Card& other) const;
    bool operator<(const Card& other) const;
    bool operator==(const Card& other) const;
private:
    Suit suit;
    Rank rank;
    int selected;
};
```

```
class Cards: public Card {
public:
    enum Type { UNKOWN, SINGLE, PAIRS, TRIPS, THREEPAIRS, TREYS, FULLHOUSE, STRAIGNT,
BOMB };
    Cards (){};
    void print() const;
    Type getType();
    const vector<Card>& getSequence() const;
    void addCard(const Card& card);
    void setType(Type type);
    void removeCard(const Card& card);
    void removeCardsOfRank(Card::Rank rank);
    Cards selectMinCardsOfType(vector<Card> hand, Type type, Cards previous);
    bool operator!=(const Cards& other) const;
    bool operator>(const Cards& other) const;
    bool detector(Cards& cards);
private:
    Type type;
    vector<Card> sequence;
};
```

```
class Player {
public:
    void addCard(const Card& card);
    void removeCard(const Card& card);
    void printHand() const;
    const vector<Card>& getHand() const;
    void setName(string name);
    void setState(int state);
    string getName();
    int getState();
private:
    vector<Card> hand;
    string name;
    int state;
};
```

```
class Game {
public:
    void clearSelect(Cards & cards);
    //start here only a version for debugging, please modify it to satisfy frontend
design
    void start();
    //used to detect whether the card exist or selected before
    bool canPlaycard(const Player& player, Cards& cards);
    //use to detect whether the cards are legal sequence or satisfying rules
    bool isLegal(Cards& cards, Cards previousCard) const;
    void generateDeck();
    void shuffleDeck();
    void dealCards();
    bool isCardInHand(const Player& player, const Card& card);
    bool isCardGeneratedByGame(const Card& card);
    bool playerAI(Player& player, Cards& select, Cards previous);
```

```
public:
    vector<Card> deck;
    vector<Player> players;
    int noPlayableCount = 0;
    const int maxNoPlayableCount = 3;
    Card maxCard;
    int maxCardPlayer = 3;
};
```

逻辑模块通过Game类和Player类来实现主要接口，其中AI位于Game类中，可选择调用。是否添加bot由用户设置的Player类state决定，以下为游戏过程中会用到的函数：

```
void start();
bool canPlaycard(const Player& player, Cards& cards);
bool isLegal(Cards& cards, Cards previousCard) const;
void generateDeck();
void shuffleDeck();
void dealCards();
void clearSelect(Cards & cards);
bool isCardInHand(const Player& player, const Card& card);
bool isCardGeneratedByGame(const Card& card);
bool playerAI(Player& player, Cards& select, Cards previous);
void addCard(const Card& card);
void removeCard(const Card& card);
void printHand() const;
const vector<Card>& getHand() const;
void setName(string name);
void setState(int state);
string getName();
int getState();
```

说明：

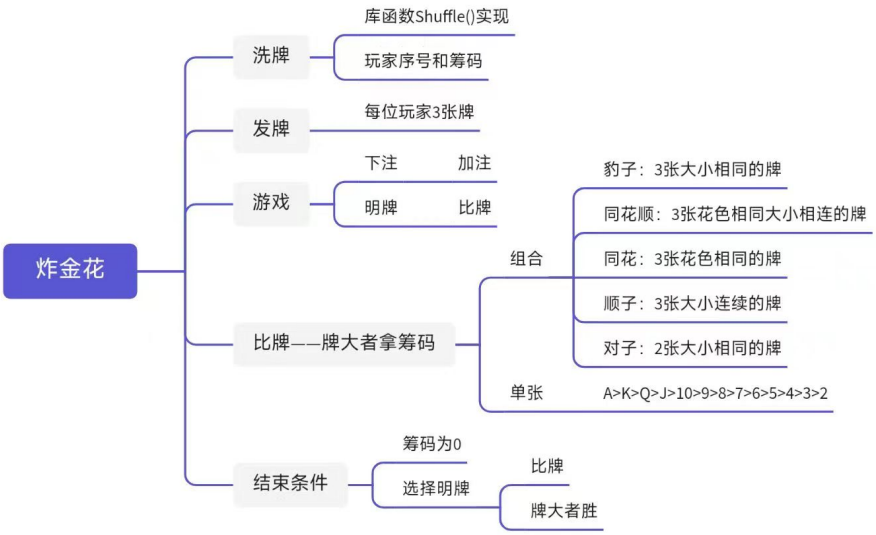
- `start`：开始新一轮游戏，设置玩家信息，牌组信息，直至游戏结束
- `canPlaycard`：用于判断对于指定玩家，牌组是否合法
- `isLegal`：用于判断对于指定牌组与上家出牌，该牌组是否符合游戏规则
- `generateDeck`：用于游戏开始时产生牌组，本游戏设定一副牌去掉一对大小王
- `shuffleDeck`：用于新局时洗牌
- `dealCards`：用于游戏开始时发牌
- `clearSelect`：撤销已选择的牌组
- `isCardInHand`：判断指定牌是否为玩家手牌
- `isCardGeneratedByGame`：判断指定牌是否为游戏生成
- `removeCardsOfRank`:将指定rank的手中所有牌去掉，便于炸弹和三张这类牌型输出
- `selectMinCardsOfType`:根据类型选择能出的最小牌，用于AI的实现
- `playerAI`：具有一定智能的自动出牌机器，能根据游戏条件选择牌组出牌，其基本逻辑框架为，根据前一名玩家所出牌的牌型，选择自己能出的合法牌中最小的牌，如果没有，则选择不出。以下是其在后端的框架：

```

    bool Game:: playerAI(Player& player, Cards& select, Cards previous){
        vector<Card> available = player.getHand();
        Card min(min.CLUBS,min.FLAG);

        map<Card::Rank, int> rankCounts;
        for (const auto& card : available) {
```

```
            rankCounts[card.getRank()]++;
        }
        vector<Card::Rank> sortedRanks;
        for (const auto& pair : rankCounts) {
            sortedRanks.push_back(pair.first);
        }
        sort(sortedRanks.begin(), sortedRanks.end());
        //0. if is the first
        Cards:: Type type = previous.getType();
        Cards least;
        if(previous.getType() != Cards:: UNKOWN){
        //1. find minimum cards that can beat previous cards, namely same type with
previo6us and larger than previous
            least = least.selectMinCardsOfType(player.getHand(), type, previous);
        }else{
            least = least.selectMinCardsOfType(player.getHand(), Cards::SINGLE,
previous);
        }
        if(least.getType() == least.UNKOWN){
            return false;
        }
        for(auto const& card: least.getSequence()){
            select.addCard(card);
        }
        if(select.detector(select)){
            return true;
        } else {
            for(auto const& card: least.getSequence())
                select.removeCard(card);
        }
        return false;
    }
```

- `addCard`：从牌组中添加牌
- `removeCard`：从牌组中移除牌
- `printHand`：显示玩家所有手牌
- `getHand`：获取玩家手牌
- `setState`：玩家设置托管或人工
- `setName`：玩家设置昵称
- `getName`：获取玩家昵称
- `getState`：获取用户状态，用于判断是否为托管状态

炸金花



Card类:

`color`、`number`：记录牌的花色、点数　`printcard()`：数据字符化输出　`operation`：重载运算符

```
class Card  {
private:
    int color;
    int number;

public:
    Card() {}
    Card(int color, int number);

    int getColor();
    void setColor(int color);

    int getNumber();
    void setNumber(int number);

    void printcard();
    bool operator==(const Card& other) const;
};
```

Player类:

- `handCard`：记录玩家手牌
- `computer`：判断玩家是否为电脑
- `see`：记录手牌是否可见
- `grade`：记录玩家牌型等级，用于牌的大小比较
- `choose`：记录玩家选择（下注/明牌）
- `setGrade()`：设置牌型等级
- `printplayer()`：展示手牌

```
class Player {
private:
    string name;
    int playernumber;
```

```
    vector<Card> handCard;
    string cardstype;

public:
    int grade;
    int base = 20;
    bool see = false;
    bool computer = true;
    int choose = 2;

    Player() {}

    string getName();
    void setName(string name);

    vector<Card> getHandCard();
    void setHandCard(int num, vector<Card> cardlist);

    int getGrade();
    void setGrade(int grade);
    int SingleJudge(vector<Card> handcard);

    void setNumber(int playernumber);
    int getNmuber();

    void printplayer();
};
```

Poker类：进行扑克牌处理

- `InitCard()`：创建52张牌组（去掉大小王）
- `Shuffle()`：打乱牌组
- `getCardType()`：判断牌型

```
class Poker {
public:
    vector<Card> list;
    void InitCard();
    void Shuffle();

    bool isTheSameNumber(vector<Card> playlist);        //豹子
    bool isTheSameColor(vector<Card> playlist);         //同花
    bool isStraight(vector<Card> playlist);             //顺子
    bool isPair(vector<Card> playlist);                 //对子

    string getCardType(Player p);
};
```

Game类：

- `Players`：玩家集合
- `Start()`：初始化游戏，创建玩家，洗牌、发牌，判断玩家牌型
- `playerAI()`：判断玩家为几人，人数不足4人时，由电脑补位，实现AI托管
- `gameplay()`：进行游戏

```
class Game {
private:
    vector<Player> Players;
public:
    void Start();
    int playerAI();
    bool gameplay(Player player);
};
```

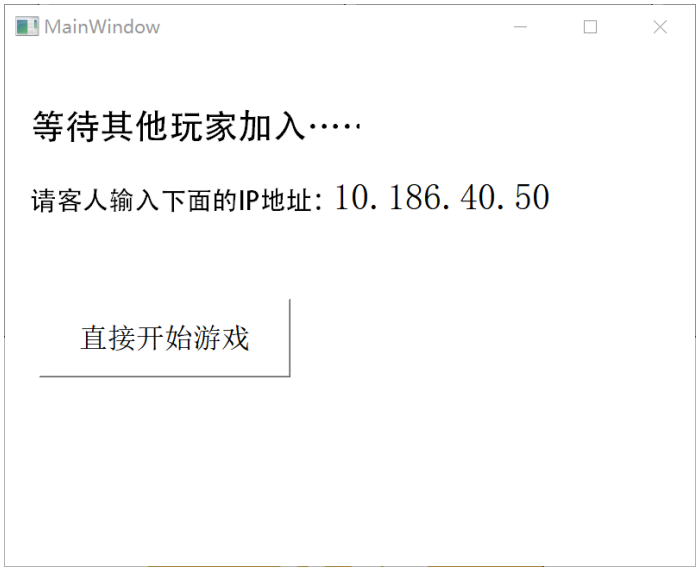
网络模块

基本介绍

网络的模块主要需要处理游戏开始前的认证以及游戏开始后的通信。关于通信有很多方式可以实现，比如著名的`winsocket`等，在这个项目中我使用了Qt框架内置的Socket来实现，因为这个项目主要基于Qt的，兼容性比较好。

房主开始游戏

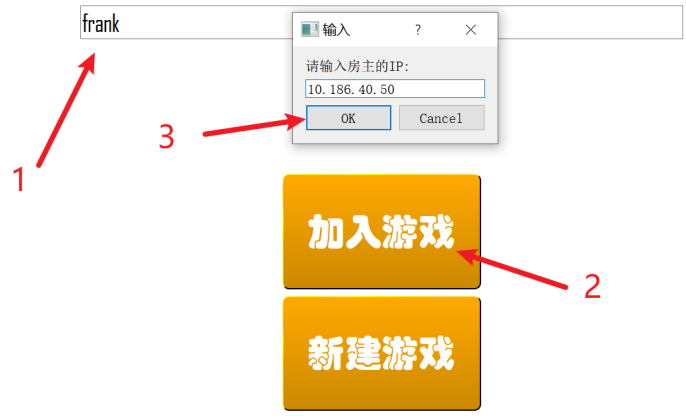
房主点击新建游戏，选择人数点击开始后开始UDP广播。自己进入`waitroom`界面等待客人加入，同时会把自己的IP显示在界面上，客人加入时通过输入绑定到这个IP上来加入游戏。；每加入一个人`now_people` +1,当`now_people`与房主预先选择的人数`playernum`相等时，房主停止广播，并进入游戏开始页面。



客人加入游戏

当房主开好房间（即进入上面的页面后，客人可以输入自己的name，再输入房主的IP加入到游戏中来。具体细节是客户端绑定到房主的IP后把自己的`name`发过去。当房主的`Socket`接受到消息时（即代表有客人加入房间），房主（服务端）会给他分配一个`player_id`作为这个客人的身份凭证，然后把服务端存储的`now_people-playernum-game-player_id`通过socket发送给它，客户端通过工厂类`factory`中的`splitQString`解析后，用新的数据更新自己的now_people，playernum,game.player_id 当客户端发现now_people=playernum时，也进入游戏界面。

Card Game



游戏后的网络设计

联机后的网络主要有`TcpSocket`来实现，首先的发牌需要不同玩家得到同一个发牌结果，而不能是各自调用`Game Init()`函数。具体的实现是服务端发完牌后，通过`hand2string`函数把各个玩家的手牌转化成字符串，再通过`Socket`把实现约定好的报文形式发送给各个客户端，客户端再通过工厂类`factory`中的`parseInitString`函数解析得到的报文，并用它来更新自己的手牌。

核心类说明：

收发类说明：

在实际的代码编写中，为了不让用于Socket的线程阻塞主线程造成程序崩溃，我采用了多线程的写法，即写了一个`cliThread`继承成Qt自带的`QThread`类，通过重载其自带的`run`函数来实现通信。另外我加上了`QTimer`来让`Socket`不断的收发消息，来实现通信的功能。

```
class cliThread : public QThread
{
private:
    QString host_ip;
public:
    void readname(QString n)
    {name = n;}
    void readIP(QString ip)
    {host_ip = ip;}
    void run() override {
        qDebug() << "正在加入游戏" ;
        QUdpSocket udpClient;
        QString localIP=getLocalIP();
        qDebug() << localIP ;
        udpClient.bind(QHostAddress(localIP), 9999); // 绑定到任意IP地址和随机端口号

        // 从输入框中获得玩家的name，发送给服务器(房主)
        QByteArray name1 = name.toUtf8();
        qDebug()<<name;
        udpClient.writeDatagram(name1.data(), name1.size(),
                                QHostAddress(host_ip), 8888);
        QTimer timer;
```

```
        timer.setInterval(1000); // 设置定时器间隔，单位为毫秒

       while(true){
       if (udpClient.hasPendingDatagrams()) {
            QByteArray datagram;
            datagram.resize(udpClient.pendingDatagramSize());
            QHostAddress senderAddress;
            quint16 senderPort;
            udpClient.readDatagram(datagram.data(), datagram.size(),
                                   &senderAddress, &senderPort);
            qDebug() << "Received datagram from" << senderAddress.toString()
                     << "on port" << senderPort;
            qDebug() << datagram;
            splitQString(datagram, now_people, playernum,game,player_id);
            qDebug() << now_people << playernum << game;
            // 处理收到的数据
            if (now_people==playernum) {
                 gamestart = 1;
                 qDebug() << "gamestart->1";
            }
       }
}
```

四、系统设计难点与解决

逻辑模块

争上游：

1. 代码复用

在判断出牌是否合法时，需要知道所出牌组的类型，再比较大小；设计AI托管时，AI需要在得知前一组牌的类型后，选择自己能出的牌；回到游戏，仍需判断是否合法。在游戏过程中，需要多次判断或设置指定牌组的类型，易造成冗余。

解决方案：

设计了 `detector` 函数，根据游戏规则，全面分析所有情况，对牌型进行判断和设定。

2. 调试

在争上游游戏中，由于游戏规则相对复杂，出牌是否合法需要很多判断条件。根据游戏逻辑，这些判断条件是有顺序关系的，应先判断牌是否存在，玩家手中有没有牌，然后再判断出牌类型及大小等。否则 `selectedCard` 将很难正确设置，并且不利于后续的调试，以及与其他模块的沟通。

解决方案：

设计了`IsCardInHand`，`IsCardGeneratedByGame`，`IsLegal` 等函数分层次进行出牌合法性判断。在避免冗余的同时提高了代码的正确性和易测试性。此后在调试过程中，很容易发现问题所在。

炸金花：

- 牌型判断：首先封装函数记录不同的牌型，相关函数牌型包括：豹子、顺子、同花、对子、单张，然后设计新函数，对传入的手牌集合进行牌型判断，并给出结果。这样只需调用新函数即可对玩家手牌进行登记判定。
- AI托管：在Player类中，设计参数`computer`对人/机玩家进行判断，若为电脑，则玩家选择默认为下注，直至筹码全部下完。

网络模块

1. 信息收发的实现

信息的收发有多种实现，我选用了Socket来完成。在实际调试中，在同一台电脑经常要开多个程序互相调试，十分费时和繁琐。

2. 广播的实现

在前期客人加入房间时需要服务端不断去广播，这时候如果如果按常规的写法不断的造成线程阻塞带来的“程序未响应”问题，通过查阅资料我了解到可以通过引入`QSocket`来实现。另外让一个函数不断的执行让我查询到可以采用一个定时器来实现。

3. 信息的交互

不同于在本地进行单机游戏，联机游戏需要不同服务端和客户端不断的进行信息的交互。我设计了一套交互规则来实现，这样不同机子按事先约定的形式来发出报文，接受的机器能调用工厂类之中的函数来解析到得到想要的信息，并用这些信息来更新自己的转态。

4. 异步和同步问题

扑克牌游戏需要不但的进行信息交互，而交互带来的时间损耗会带来异步和同步问题。我的解决思路是采用服务器-客户端模式：在服务器-客户端模式下，服务器负责处理所有玩家的操作，并将游戏状态同步给所有玩家。这样可以避免异步问题。

界面模块

1、不同页面的信息传递

初期实现了各个页面的设计，实现了不同页面的切换，但是在选择页面的选择信息还有开始页面的玩家姓名需要传递到游戏页面。

解决方案：

设置全局变量，将全局变量放在global.h中，然后在需要使用或者修改全局变量的类中包含global.h

2、卡牌的显示和选择

游戏卡牌有不同花色和序号，要将各种卡牌从card类转换成显示的容器，并且能够支持选中功能

解决方案：

选择自定义CardButton类和CardWidget类，分别是可以选择和不能选择的卡牌显示容器。将卡牌根据花色和序号进行编码，在后端的Card类中加入getType函数，调用以获得卡牌的类型名称对应字符串，然后在构造CardButton等容器时传入类型名称，同时将不同牌面的图片资源和类型名称设置为统一的名字，在容器实现时可以根据名称导入对应的图片资源。

3、组件摆放和大小问题

使用布局管理器时很难将组件放到自己需要的位置

解决方案：

使用自定义布局，根据主页面大小计算组件绝对位置和大小

五、总结

逻辑模块

争上游：

- 作为后端逻辑模块，我在编写时需要和前端团队成员进行沟通API，适应前端接口需求和相应逻辑，才能编写出正确的代码。在调试过程中，我需要判断bug在后端还是前端，这就需要与队友一起调试，锻炼了协作能力。而在前端队友建立的可视化分析下，很多问题被发现和解决。此外，我们团队还进行了几次集体调试，因而有效解决了很多问题，对相互间的代码有一定的了解。
- 中期时，由于规划时对具体实现没有深入分析，导致后端逻辑代码难以继续编写。重新分析了游戏规则后，我对各类的实现做了许多调整，最终确定为Game类，Player，Card类和Cards类，才逐渐建立起较为合理且易于维护的框架。同时由于前期设计考虑不够全面，留下了很多后来花费很多时间才找到，并感到难以理喻的bug，这使我明白项目初期框架的科学设计与稳扎稳打的重要性。
- **程序的不足**：在AI设计上还可以进一步改进，以后可继续实现不同难度等级的单机游戏，分别对应不同策略的AI。目前本程序的AI是基于在能出的牌中选择最小的牌这一策略设计的，具有初级牌手水平。如果考虑特殊情况下的拆对，拆三张等，或许可以提高AI的能力，以适应水平较高牌手的需求。

炸金花：

- 通过本次的程序设计，锻炼了使用C++进行编程的能力，将课程中所学的理论知识运用到实践，进一步加深了理解。作为后端逻辑模块，我在编写时需要对调用的函数进行一定的封装并提供接口，方便与负责前端的同学进行对接，并在过程中，根据实际情况对代码进行修改，以实现最终的图形化界面交互。
- **不足**：筹码下注的复杂度可以提高，比如玩家可选择下注多少，另明牌与弃牌的规则可以进一步细化；明牌判断中，对不同等级的牌大小进行再一步比较，可以量化比较结果，进行赌注的分配，加大玩家筹码后，增加游戏进行轮数。对于函数封装与继承的设计不够“优雅”存在冗余，在一些函数调用中参数传递不够简洁直接

网络模块

- 我原先从来没有接触过网络编程，作为数学专业的学生我也没有计网方面的理论知识。在前期我花了很多时间去学习相关的知识，包括`Socket`是什么，如何用其实现信息的交互，TCP/UDP有什么区别，什么是心跳机制，虽然一开始云里雾里，后来还是慢慢有了一个认识。
- 在真正把网络应用到我们这个项目中，我也遇到了很多问题，包括如何设计交互的信息格式，在哪里放socket，一个人调试网络很痛苦，不过在熬了许多个夜后，我终于把服务端和客户端的收发跑通了，十分有成就感。
- 第一次感受多个人合作一个项目，我的网络模块基本上是最后设计的，所以在接手同学的代码后我花了很多时间去理清各个接口以及他们的逻辑。在最后的整合中，因为炸金花的后端和网络模块是差不多同时开始的，所以当两边整合代码时遇到了很多问题，我也认识到版本控制系统的重要性，最后通过慢慢调试终于把两边的冲突解决了。

不足：到最后网络没有完全可用，基本只有前期的登录认证，加入房间时可以用的。实现过程碰到了一些个人难以解决的问题，还是感叹自己的能力微薄。另外由于时间比较紧，我的代码美观度也不足，为了实现功能写的比较乱。

界面模块

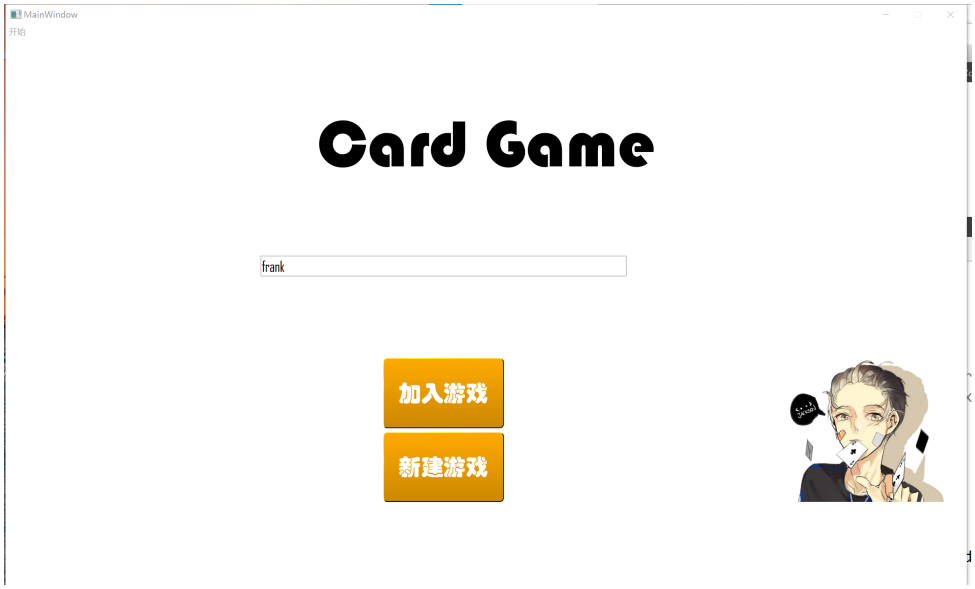
- 作为前端模块，我需要根据后端代码思考如何将控制台游戏转换成具有美观ui的窗口程序，并且设计交互功能。这过程中需要和后端团队的成员沟通API，构思实现整体实现逻辑，并对后端接口提出合理的接口需求。
- 在ui界面设计中，最麻烦和复杂的问题就是交互，在qt中也就是信号和槽，将按钮的按下和相应的函数进行连接。
- 将后端代码整合到项目中的时候，能感受到分文件、分函数、面向对象设计带来的可读性和可维护性的优秀表现，比如在和后端同学对接时，不同函数之间不会相互影响，在前端获得了接口后进行实现的同时后端的同学也在不断优化代码，而二者的同时更新不会带来很大的影响。
- **不足**：对于多线程的理解还略有所欠缺，实现信号和槽的时候没有充分考虑到多线程的问题，大量函数的互相调用和信号槽混用，可能会在更加负责的工程中引起问题。

附录一、程序使用说明

游戏操作

创建游戏：（多人联机或单机游戏）

输入名字，点新建游戏：

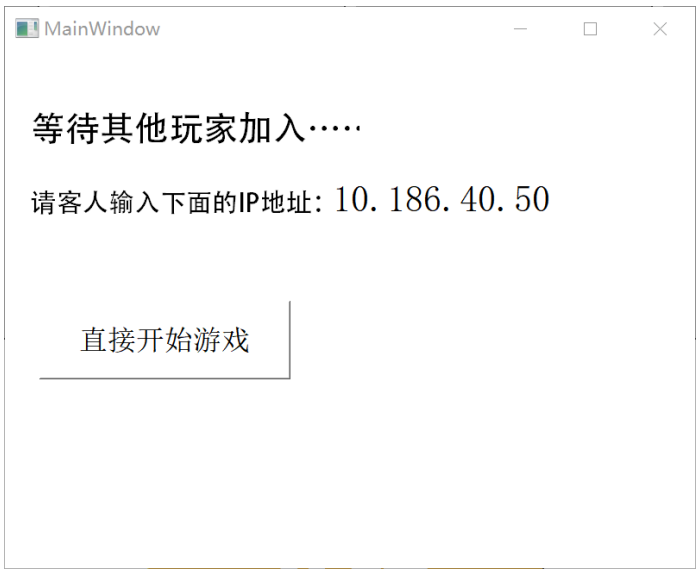


选择游戏界面：

选择联机游戏：

选择房间人数：

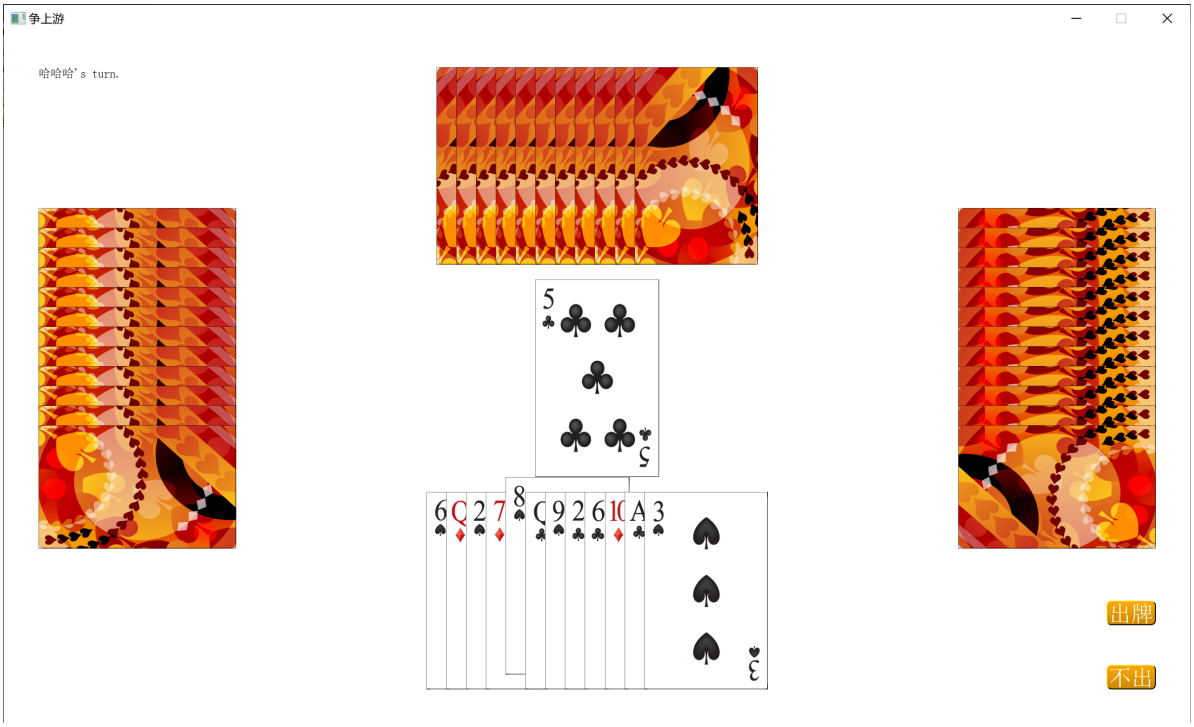
再点击开始后，会挑出一个框显示等待玩家进入。



客人加入游戏

单机游戏或联机模式开始后都会先进入“开始”界面：

争上游：



炸金花：

游戏规则

争上游

- 4个人 每人13张，人数不足由AI代替
- 牌型
  - 单张，对子，三张，姊妹对，要求三张连续对子，三带一，三带二，顺子，仅指5张连续点数的牌，炸弹，仅指4张相同的牌
- 不支持连对，及连续对子>3的牌组；支持王炸，即大王和小王组成的对子
- 大小比较：3<4<5<6<7<8<9<10<J<K<Q<A<2<小王<大王
  - 三带二、三带一的大小按牌数多的牌的比，如44433>33355
  - 姊妹对的大小按最小的牌比，如665544>554433

- 其他牌型遵循一般大小规则
- 当前玩家是本轮第一个出牌的玩家，可以任意出，否则，所出的牌必须大过上一个出牌的人，并且牌型与上一个人相同才可以出牌
- 如果玩家有王炸或炸弹，则无论上家牌型是什么都可以出牌，但如果上家也是炸弹，则必须比上家大

炸金花

- 4个人，每人3张牌，根据点数和花色比较大小；人数不足时，由AI代替
- 玩家可选择下注比牌，或放弃比牌
- 牌面比较：
  - 牌型比较：豹子（三张相同点数的牌） > 同花顺（相同花色的3张连牌） > 顺子 > 同花 > 对子 > 单张
  - 大小比较：A > K > Q > J > 10 > 9 > 8 > 7 > 6 > 5 > 4 >3 > 2

附录二、系统开发日志

5月7日

- 第一次开会：完成各个模块的分工

5月15日

- （顾格非）完成了前端的页面白板设计

5月19日

- （顾格非）完成了页面的跳转逻辑设计
- （江含韵）完成了争上游的初步尝试编写。
- （江含韵）根据前端需求修改了Player类

5月20日

- （王思雨）基本完成学习前端框架的学习
- （江含韵）增加争上游运算符重载比较大小和合法牌判断函数
- （高雪）完成后端的`player`,`game`,`card`类设计

5月25日

- （顾格非）基本完成网络模块的学习
- （江含韵）添加争上游初级AI

6月6日

- （王思雨）前端完成完成除游戏页面以外的页面设计
- （江含韵）修复争上游选择牌重复问题

6月7日

- （顾格非）在第一版的前端上加入Socket调试
- （江含韵）完成了争上游逻辑模块的初步编写。开始进行调试。

6月11日

- （王思雨）完成和争上游后端的交互，实现了带有bug的游戏页面

6月12日

- （顾格非）在前端上加入几个`waitroom`等界面，适配联机
- （王思雨）修复争上游前一张牌显示问题

6月13日

- （顾格非）完成服务端的Socket设计，可以新建房间开始广播
- （高雪）完成炸金花的逻辑模块设计

- （王思雨）重新修复之前修改后出牌和摸牌造成程序崩溃的遗留问题
- （江含韵）完成了争上游AI模块的编写，并在逻辑模块中加入了对AI模块的调用。

6月14日

- （顾格非）完成客户端的Socket设计
- （江含韵）拆分AI，增加牌型判断

6月15日

- （顾格非）完成客户端和服务端的调试，两边人数到时能同时进入游戏
- （王思雨）设计的炸金花的前端画面

6月16日

- （高雪）完成后端炸金花的交付
- （顾格非）设计游戏界面的通信，包括牌的报文通信
- （王思雨）完成的炸金花页面实现

6月18日

- （顾格非）整合所有代码并调试，打包成可执行程序
- （顾格非）整合实验报告

附录三、小组分工

前端设计，网络模块，代码整合：

前端实现、交互设计：

后端逻辑模块（争上游）、AI：

后端逻辑模块（炸金花）、AI：