浙江大学

《面向对象程序设计》课程设计报告

作业名称：基于 Qt 的斗地主游戏

小组成员：

2022 年 6 月

1. 项目概述

1.1 总体设计

本课程设计基于 Qt 和 C++实现一个具有友好图形界面的斗地主游戏。游戏设计采用了前后端分离的模式——前端使用了跨平台的 C++ 开发库 Qt，实现了图形用户界面的设计与开发；后端负责游戏逻辑规则的设计和实现，以及对局 AI 的策略设计和开发。最终，前后端衔接，实现功能完整且用户体验良好的斗地主游戏。

本项目为单机游戏，斗地主中其余两个玩家由电脑 AI 代替。用户可以选择两种游戏模式：

1. 传统斗地主模式，即游戏中有一个地主与两个平民，用户和电脑 AI 都可以通过叫分来“抢地主”。

2. 大乱斗模式，即游戏中扑克牌平分（每个人 18 张牌），每个玩家都是“地主”身份，判定胜利的标准为自己手中牌是否最先为空。

支持的特殊牌型有：

炸弹：四张点数相同的牌或双王，如：7777。

对牌：任意两张点数相同的牌。

三张：任意三张点数相同的牌，如 888。

三带一：点数相同的三张牌+一张单牌或一对牌。如：333+6 或 444+99。

单顺：任意五张或五张以上点数相连的牌，如：45678 或 78910JQK 。

双顺：三对或更多的连续对牌，如：334455、7788991010JJ。

三顺：二个或更多的连续三张牌，如：333444 、555666777888。

飞机带翅膀：三顺＋同数量的单牌或同数量的对牌。如：444555+79 或 333444555+7799JJ

牌型比较：火箭最大，炸弹次之；再次是一般牌型（单牌、对牌、三张牌、三带一、单顺、双顺、三顺、飞机带翅膀、四带二）。一般牌型：只有牌型且张数相同的牌才可按牌点数比较大小。其中三带一、飞机带翅膀、四带二组合牌型，比较其相同张数最多的牌点数大小。

1.2 组员分工

wxy：负责前端设计与开发（即初界面与游戏界面的设计、界面跳转设计）、统筹规

划与编写设计报告；

zyc：参与设计后端的 Card，Player，PlayerAI，User，Stragety 类，负责完成后端出牌策略；

2. 代码分析

2.1 基类设计

自定义的基类主要有 3 个，MyPushButton, CardWidget, Player，具体功能如下：

构建基类 MyPushButton，公有继承 Qt 库中的 QPushButton，用于实现按键的图片加载和显示。用户有成员变量 QString inputImgPath，buttonModeInfo 用于存储图片和按钮类别信息，并且提供 zoomIn 和 ZoomOut 函数，用于按键的动画效果。

类初始化和动画 zoomIn 函数的实现如下：

```
MyPushButton::MyPushButton(const QString &inputpath, const QString &modeinfo) {
    inputImgPath = inputpath;
    buttonModeInfo = modeinfo;
    QPixmap pix;
    bool ret = pix.load(inputImgPath);
if (!ret) return;

    // set pic size
setFixedSize(pix.width() - 10, pix.height() - 10);

    // set Icon size
    setIcon(pix);
    setIconSize(QSize(pix.width(), pix.height()));
}

void MyPushButton::zoomIn() {
    QPropertyAnimation *animation = new QPropertyAnimation(this, "geometry");

    // set duration
    animation->setDuration(50);

    // set startvalue & endvalue
    animation->setStartValue(QRect(this->x(), this->y(), this->width(),
this->height()));
    animation->setEndValue(QRect(this->x(), this->y() + 10, this->width(),
this->height()));

    // set curves
    animation->setEasingCurve(QEasingCurve::InOutQuad);

    // set loops
    animation->setLoopCount(10);

    // start animation
    animation->start();
}
```

构建基类 CardWidget，公有继承 Qt 库的 QWidget。用于显示每张卡牌，设定卡牌对应的图片、所有者、是否被选中等信息。还需要为为控制模块提供函数 setOwner(), getOwner,

setFront(), setIsSelected 等，具体定义如下：

```
class CardWidget : public QWidget {
Q_OBJECT

public:
    explicit CardWidget(QWidget *parent = nullptr);
    void setPix(const QPixmap &pix);
    QPixmap getPix() const;
    void setOwner(Player *owner);
    Player *getOwner() const;
    void setCard(const Card &card);
    Card getCard() const;
    void setFront(bool isFront);
    bool getIsFront() const;
    void setIsSelected(bool isSelect);
    bool getIsSelected() const;

signals:
    void notifySelected(Qt::MouseButton mouseButton);

protected:
    void paintEvent(QPaintEvent *event);
    void mousePressEvent(QMouseEvent *event);

private:
    bool isSelected;
    bool isFront;
    Player *owner;
    QPixmap pix;
    QPixmap back;
    Card card;
};
```

构建基类 Player，用于记录玩家信息，派生出用户玩家 User 类和电脑玩家 Robot 类。包含成员变量 bool isLandLord, isPerson, betPoints 等，用于记录该玩家是用户还是电脑、是否为地主、叫分信息、拥有卡牌信息、上一次有效的卡牌等。具体类定义如下：

```
class Player : public QObject {
Q_OBJECT
public:
    explicit Player(QObject *parent = nullptr);
    void setPlayerID(int ID);
    int getPlayerID() const;
    void resetHandCards(const QVector<Card> &cards);
    void setHandCards(const QVector<Card> &cards);
    QVector<Card> getHandCards() const;
    void resetSelectCards(const QVector<Card> &cards);
    void setSelectCards(const QVector<Card> &cards);
    //QVector<Card> getSelectCards() const;
    CardGroups getSelectCards() const;
    void setBetPoints(int bet);
    int getBetPoints() const;
    void setIsLandLord(bool isLandLord);
    bool getIsLandLord();
    void setIsPerson(bool isPerson);
    bool getIsPerson();
    void setNextPlayer(Player *next);
    Player *getNextPlayer();
    Player *getPunchPlayer();
    CardGroups getPunchCards();
    int getCardsNumber() const;   // 得到当前牌的数量
    void addLandLordCards(const QVector<Card> &cards); // 添加地主牌
    bool isWin();  // 是否赢
    bool checkCardValid(CardGroups &currentCombo);

    void showCards();
    void clear();
    void callLord(int bet);
    void playHand(const QVector<Card> &cards);
    virtual void startCallLord();
```

```
    virtual void startPlayHand();
    virtual void thinkCallLord();
    virtual void thinkPlayHand();
    CardGroups lastCards; //pending

signals:
    void notifyCallLord(int);
    void notifyPlayHand(Player *);
public slots:
    void onPlayerPunch(Player *player, const CardGroups &cards);
protected:
    void sortHandCards();     // 默认升序，先按点数，点数相同则按花色排序
protected:
    bool isLandLord = false;  // 玩家是否为地主
    bool isPerson = false;    // 玩家是人还是机器人

    int betPoints = 0;        // 玩家叫的分
    int playerID;             // 玩家序号

    QVector<Card> handCards;  // 玩家手上的牌
    CardGroups selectCards;   // 选中的牌

    Player *punchPlayer;
    CardGroups punchCards;

    Player *nextPlayer;       // 下一个玩家
};
```

2.2 界面设计

斗地主游戏的前端主要负责显示界面，游戏中设计了两个界面：初界面和游戏界面。

构建 MainWindow 类，公有继承了 Qt 库中的 QMainWindow 类，实现游戏开始界面以及游戏模式的选择。MainWindow 具有成员对象 GameWindow（同样为 QMainWindow，实现之后的牌桌界面)。

1. 设置任务栏，setWindowTitle 显示“Landlord: Welcome!”以及 setWindowIcon 设置左上角图标。

2. 初始化用于游戏模式选择的按键（“人机模式”，“人人模式”即大乱斗模式，“退出游戏”），每个按键都是 MyPushButton 对象，链接对应的游戏模式的信号槽。点击两个模式选择按钮中的任意一个，所有按钮隐藏，调用控制类的 setgamemodel 函数，初始化游戏窗口，关闭当前窗口，显示游戏窗口界面。点击退出按钮，则直接关闭窗口，退出游戏。

3.定义主窗口的 paintEvent，显示背景图片。

构建 GameWindow 类，公有继承了 Qt 库中的 QMainWindow 类，实现游戏牌桌界面的显示，以及和控制模块的衔接。主要函数有：

1. GameWindow 初始化，内容包括：

设置窗口大小和任务栏信息；初始化 GameControl 对象，建立三个玩家对象；初始化卡牌，为 54 张卡牌分别建立 Card 对象，初始化图片地址和卡牌信息；初始化按键，显示“开

始游戏”按键，隐藏叫分按键（“1 分”、“2 分”、“3 分”、“不叫”）和出牌按键（“过”、“出牌”）；初始化玩家信息（结构体 PlayContext），设置每个玩家的卡牌排列方向、是否正面显示；初始化每个玩家的叫分情况、出牌情况，以及地主玩家编号（“left”, “right”, “me”, “everyone”），用 QLabel 进行显示；初始化信号槽。

在点击“开始游戏”按钮之后，链接信号槽 onStartBtnClicked()。如果在“人机模式”，会调用 call4LandLordCard()函数，进入“抢地主”环节；如果在“大乱斗模式”，直接开始游戏，调用 startGame。

2. 叫地主 call4LandLordCard()函数。显示叫分按键（“1 分”、“2 分”、“3 分”、“不叫”），每个按键链接不同信号槽，调用 gameControl 对象为玩家设置叫分信息，调用 updateBetPoints 函数，同时判断谁是地主。

3. 卡牌显示函数 otherPlayerShowCards 和 showOtherPlayerPlayCard。玩家每次出牌后，调用 otherPlayerShowCards 函数更新当前玩家和调用 showOtherPlayerPlayCard 函数其他两个玩家的卡牌显示情况

具体类定义如下：

```
class GameWindow : public QMainWindow {
Q_OBJECT
public:
    explicit GameWindow(QWidget *parent = nullptr);
//    void HidePlayerLastCards(Player* player);
    void UpdatePlayerCards(Player *player);
    GameControl *getgameControl();
    void init();

private:
    virtual void paintEvent(QPaintEvent *);
//    virtual void resizeEvent(QResizeEvent* event);
    void initCardWidgetMap();
    void initButtons();
    void initPlayerContext();
    void initLandLordCards();
    void initInfoLabel();
    void initSignalsAndSlots();
    void insertCardWidget(const Card &card, QString &path);
    void addLandLordCard(const Card &card);
    void showLandLordCard();
    void showMyCard(Player *myPlayer);
    void showOtherPlayerCard(Player *otherPlayer, const QString status);
    void showRemLandLordCard(QString status);
    void call4Landlord();
    void showMySelectedCard(Player *player);
    void startGame();
    void showPlayCard();
   void showOtherPlayerPlayCard(Player *otherPlayer, CardGroups cards, const QString
status);

signals:

public slots:
    void onStartBtnClicked();
    void onBetNoBtnClicked();
    void onBet1BtnClicked();
    void onBet2BtnClicked();
    void onBet3BtnClicked();
    void onBetPointsCall(Player *player);
    void cardSelected(Qt::MouseButton mouseButton);
```

```
    void playCards();
    void passCards();
    void otherPlayerShowCards(Player *player, CardGroups cards);
    void myPlayerShowButton(Player *player);
    void showEndStatus(Player *player);


protected:
    enum CardsAlign {
        Horizontal,
        Vertical
    };

    struct PlayerContext {
        QRect cardsRect;
        QRect playHandRect;
        CardsAlign cardsAlign;
        bool isFrontSide;
        QVector<Card> lastCards;
        QLabel *info;
        QLabel *rolePic;
    };

private:
    GameControl *gameControl;
    QPixmap cardBack;  // 背面图像
    QMap<Card, CardWidget *> cardWidgetMap;
    QVector<CardWidget *> restThreeCards;
    QMap<Player *, PlayerContext> playerContextMap;
    QPoint baseCardPos;
    QFrame *userTool;
    QSize cardSize;

    QLabel *myBetInfo;
    QLabel *leftBetInfo;
    QLabel *rightBetInfo;
    QLabel *passInfo;
    QLabel *playInfo;
    QLabel *rightPassInfo;
    QLabel *leftPassInfo;
    QLabel *myStatusInfo;
    QLabel *leftStatusInfo;
    QLabel *rightStatusInfo;
    QLabel *myLandLordInfo;

    MyPushButton *startBtn;
    MyPushButton *betNoBtn;
    MyPushButton *bet1Btn;
    MyPushButton *bet2Btn;
    MyPushButton *bet3Btn;
    MyPushButton *passBtn;
    MyPushButton *playBtn;

    static const QSize gameWindowSize;

    static const int cardWidthSpace;
    static const int cardHeightSpace;
    static const int cardRemSpace;

    static const int myCardWidthStartPos;
    static const int myCardHeightStartPos;
    static const int leftCardWidthStartPos;
    static const int leftCardHeightStartPos;
    static const int rightCardWidthStartPos;
    static const int rightCardHeightStartPos;
    static const int remCardWidthStartPos;
    static const int remCardHeightStartPos;

    // button param
    static const int betBtnWidthStartPos;
    static const int betBtnHeightStartPos;
    static const int betBtnWidthSpace;

    // bet info
    static const int fontSize;
    static const QPoint myBetInfoPos;
    static const QPoint leftPlayerBetInfoPos;
```

```
    static const QPoint rightPlayerBetInfoPos;

    static const int cardSelectedShift;

    // play info
    static const QPoint passBtnStartPos;
    static const QPoint playBtnStartPos;
    static const QPoint myCardZone;
    static const int myCardZoneWidthSpace;
    static const QPoint rightCardZone;
    static const int rightCardZoneHeightSpace;
    static const QPoint leftCardZone;
    static const int leftCardZoneHeightSpace;

    // status info
    static const QPoint myStatusPos;
    static const QPoint leftStatusPos;
    static const QPoint rightStatusPos;

    static const QPoint myLandLordPos;
};
```

2.3 规则与控制设计

GameControl 类负责游戏流程控制，包含成员对象当前玩家、用户玩家、AI 玩家、有效牌、地主牌、所有卡牌等等。这一部分完成了如下功能：

- 发牌的流程控制：发牌给各个玩家的顺序控制；
- 叫分的流程控制：玩家叫分的顺序控制；
- 出牌的流程控制：包括当前最大的牌（有效牌）以及当前玩家的指针；
- 游戏状态的控制：是否已经出现胜利的一方。

主要函数有：

1. updateBetPoints 函数，根据三个玩家的叫分情况判断地主。在“大乱斗模式”，三个玩家都设置为地主；在“人机模式”下，如果用户玩家叫 3 分，则直接判定为地主，否则三个玩家依次叫分，在都叫分后(即 betList.size()=3 时)，再开始判断地主。

2. initCards 和 getRandomCards 函数，实现随机发牌。先调用 initAllCards()函数初始化所有卡牌对象，然后根据不同的游戏模式，调用进行 getRandomCards 函数发牌。如果为“人机模式”，每个玩家随机 17 张牌，余下三张地主牌；若为“大乱斗模式”，每个玩家随机 18 张牌，不需要设置地主牌。

3. onPlayerHand 函数，每次玩家完成出牌之后，更新牌桌的卡牌显示，更新每个玩家下次能够出牌的限制信息。还需要判断当前玩家是否已经把牌出完，即是否胜利。最后将当前玩家设置为下一个玩家，若下一个轮到用户玩家出牌，则显示“过”和“出牌”按键。

具体的类定义如下：

```
class GameControl : public QObject {
Q_OBJECT
public:
    explicit GameControl(QObject *parent = 0);

public:
    Player *getCurrentPlayer();
    User *getPlayerA();
    Robot *getPlayerB();
    Robot *getPlayerC();
    CardGroups getCurrentCombo();
    Player *getEffectivePlayer();
    QVector<Card> getLandLordCards();
    void updateBetPoints(int bet); //处理叫地主分数
    void initCards(); //发牌
    void init(); //初始化玩家
    void setgamemodel(int gamemodel);
    int getgamemodel();

private:
    void initAllCards();
    QVector<Card> getRandomCards(int start, int cardnum);

signals:

    void callGamewindowShowCards();   //发牌结束后通知 gamewindow 显示卡牌
    void callGamewindowShowBets(Player *player);    //叫分结束后通知 gamewindow 显示叫
分
    void callGamewindowShowLandlord(); //处理叫分后通知 gamewindow
    void NotifyPlayerPlayHand(Player *player, CardGroups &cards);
    void NotifyPlayerbutton(Player *player);
    void NotifyPlayerStatus(Player *player);

public slots:

    void onPlayerHand(Player *player, CardGroups &cards); //处理叫地主分数
    void onPlayerHandRobot(Player *player); //处理叫地主分数
    // void updateBetPoints(int bet); //处理叫地主分数
    //void handout(int bet); //处理出牌

protected:
    struct BetRecord {
        Player *player;
        int bet;
    };

protected:
    Player *currentPlayer;   //当前玩家
    User *playerA;
    Robot *playerB;
    Robot *playerC;            //参与者 A、B、C
    CardGroups currentCombo; //有效牌
    Player *effectivePlayer; //有效玩家
    QVector<Card> landLordCards;//三张地主牌

    Player *punchPlayer;
    QVector<Card> punchCards;

    int betCalledNum;         //地主优势叫分
    QVector<BetRecord> betList;

    QVector<Card> allCards;

    int gamemodel;
};
```

2.4 对局 AI 设计

Robot 类、Strategy 类、User 类、CallLandLordThread 类实现了电脑 AI 的功能。其中，

Strategy 类包含了电脑 AI 的出牌规则，而 CallLandLordThread 类包含了电脑 AI 的叫分规则。

1. 构建 Robot 类，公有继承基类 User，增加了叫分和出牌函数，具体实现如下：

thinkCallLord 函数，根据拿牌情况判断叫分。初始化权重为 0，小王大王权重都为 6，调用 strategy 找出特殊牌型——“单顺”、炸弹的权重为 5，“三张”权重为 4，对牌权重为 1。所有权重累加之后，如果总权重大于等于 22，则叫 3 分；若总权重小于 22，大于等于 18，则交两分；若总权重小于 18，大于等于 10，则叫 1 分；否则不叫地主。

thinkPlayHand 函数，调用 strategy 类的 makeStrategy 函数。

2. 构建 Strategy 类，用于指定策略，完成出牌。主要函数如下：

makeStrategy 函数为出牌策略的主函数，根据情况依次调用其他策略函数：首先判断是否为第一个出牌，若为第一个出牌，调用 playFirst 函数。否则，调用 whetherToBeat 函数判断能否出牌，如果不能，返回空的 QVector<Card>；如果能出牌，返回卡牌。

playFirst 函数：首先判断剩下的牌能否一次性出完，如果不能，先找出第一组“单顺”的范围，在该范围中依次判断并排除“炸弹”和“三张”，如果该范围中还存在“单顺”，则出该“单顺”。若不存在，继续寻找“炸弹”、“飞机”、“三张”和对子。如果存在“飞机”，尝试找出两组三顺，在判断能否是否还存在两张单牌（可以出“飞机带翅膀”）...若不存在飞机，再判断“三张”、对子（要尽可能长）...具体代码如下：

```
QVector<Card> Strategy::playFirst() {
    CardGroups hand(cards);
    if (hand.getCardsType() != Group_Unknown)        // 只剩一手牌，直接出完
    {
        return cards;
    }

    QVector<Card> seqSingleRange = getFirstSeqSingleRange();
    if (!seqSingleRange.isEmpty()) {
        QVector<Card> left = seqSingleRange;

        //清除炸弹和三个
        QVector<QVector<Card> > cards4 = Strategy(player, left).findCardsByCount(4);
        for (int i = 0; i < cards4.size(); i++)
            for (int j = 0; j < cards4[i].size(); j++)
                left.removeOne(cards4[i][j]);

        QVector<QVector<Card> > cards3 = Strategy(player, left).findCardsByCount(3);
        for (int i = 0; i < cards3.size(); i++)
            for (int j = 0; j < cards3[i].size(); j++)
                left.removeOne(cards3[i][j]);

        QVector<QVector<Card> > optimalSeq =
                                Strategy(player, left).pickOptimalSeqSingles();
        if (!optimalSeq.isEmpty()) {
            int oriSingleCount = Strategy(player, left).findCardsByCount(1).size();

            for (int i = 0; i < optimalSeq.size(); i++)
                for (int j = 0; j < optimalSeq[i].size(); j++)
                    left.removeOne(optimalSeq[i][j]);

            int leftSingleCount = Strategy(player, left).findCardsByCount(1).size();

            if (leftSingleCount < oriSingleCount) {
                return optimalSeq[0];
```

```
            }
        }
    }

    bool hasPlane, hasTriple, hasSeqPair;
    hasPlane = hasTriple = hasSeqPair = false;
    QVector<Card> leftCards = cards;

    QVector<QVector<Card> > bombArray =
Strategy(player,leftCards).findHand(CardGroups(Group_Bomb, Card_Begin, 0),false);

    for (int i = 0; i < bombArray.size(); i++)
        for (int j = 0; j < bombArray[i].size(); j++)
            leftCards.removeOne(bombArray[i][j]);

    QVector<QVector<Card> > planeArray =
Strategy(player,leftCards).findHand(CardGroups(Group_Plane, Card_Begin, 0), false);
    if (!planeArray.isEmpty()) {
        hasPlane = true;
        for (int i = 0; i < planeArray.size(); i++)
            for (int j = 0; j < planeArray[i].size(); j++)
                leftCards.removeOne(planeArray[i][j]);
    }

    QVector<QVector<Card> > tripleArray =
Strategy(player,leftCards).findHand(CardGroups(Group_Triple, Card_Begin, 0), false);
    if (!tripleArray.isEmpty()) {
        hasTriple = true;
        for (int i = 0; i < tripleArray.size(); i++)
            for (int j = 0; j < tripleArray[i].size(); j++)
                leftCards.removeOne(tripleArray[i][j]);
    }
    QVector<QVector<Card> > seqPairArray = Strategy(player, leftCards).findHand(
            CardGroups(Group_Seq_Pair, Card_Begin, 0), false);
    if (!seqPairArray.isEmpty()) {
        hasSeqPair = true;
        for (int i = 0; i < seqPairArray.size(); i++)
            for (int j = 0; j < seqPairArray[i].size(); j++)
                leftCards.removeOne(seqPairArray[i][j]);
    }

    if (hasPlane) {
        bool twoPairFound = false;
        QVector<QVector<Card> > pairArray;
        for (CardPoint point = Card_3; point <= Card_A; point = CardPoint(point + 1))
{
        QVector<Card> pair = Strategy(player, leftCards).findSamePointCards(point, 2);
            if (!pair.isEmpty()) {
                pairArray << pair;
                if (pairArray.size() == 2) {
                    twoPairFound = true;
                    break;
                }
            }
        }

        if (twoPairFound) {
            QVector<Card> playCards = planeArray[0];
            for (int i = 0; i < pairArray.size(); i++)
                for (int j = 0; j < pairArray[i].size(); j++)
                    playCards.append(pairArray[i][j]);

            return playCards;
        } else {
            bool twoSingleFound = false;
            QVector<QVector<Card> > singleArray;
       for (CardPoint point = Card_3; point <= Card_A; point = CardPoint(point + 1)) {
          if (countOfPoint(leftCards, point) == 1) //(leftCards.PointCount(point) == 1)
                {
        QVector<Card>single=Strategy(player, leftCards).findSamePointCards(point, 1);
                if (!single.isEmpty()) {
                    singleArray << single;
                    if (singleArray.size() == 2) {
                        twoSingleFound = true;
                        break;
                    }
                }
```

```
                }
            }

            if (twoSingleFound) {
                QVector<Card> playCards = planeArray[0];
                for (int i = 0; i < singleArray.size(); i++)
                    for (int j = 0; j < singleArray[i].size(); j++)
                        playCards.append(singleArray[i][j]);、
                        return playCards;
            } else {return planeArray[0];}
        }
    }

    if (hasTriple) {
        if (CardGroups(tripleArray[0]).getBasePoint() < Card_A) {
            for (CardPoint point = Card_3; point <= Card_A; point = CardPoint(point + 1))
{
          int pointCount=countOfPoint(leftCards,point);//leftCards.PointCount(point);
          if (pointCount == 1) {
       QVector<Card> single = Strategy(player,leftCards).findSamePointCards(point, 1);
          if (!single.isEmpty()) {
       QVector<Card> playCards = tripleArray[0];
          for (int i = 0; i < single.size(); i++)
                playCards.append(single[i]);
          return playCards;
          }else if (pointCount == 2) {
        QVector<Card> pair = Strategy(player, leftCards).findSamePointCards(point, 2);
                    if (!pair.isEmpty()) {
                        QVector<Card> playCards = tripleArray[0];

                        for (int i = 0; i < pair.size(); i++)
                            playCards.append(pair[i]);
                        return playCards;
                    }}}
            return tripleArray[0];        // 找不到合适的带牌，直接出 3 个
        }}
    if (hasSeqPair)        // 打出最长的连对
    {
        QVector<Card> maxSeqPair;
        for (int i = 0; i < seqPairArray.size(); i++) {
            if (seqPairArray[i].size() > maxSeqPair.size()) {
                maxSeqPair = seqPairArray[i];
            }
        }
        return maxSeqPair;
    }
    Player *nextPlayer = player->getNextPlayer();

    if (player->getIsLandLord() != nextPlayer->getIsLandLord() &&
        nextPlayer->getHandCards().size() == 1) {
        for (CardPoint point = CardPoint(Card_End - 1); point >= Card_3; point =
CardPoint(point - 1)) {
            int pointCount =
countOfPoint(leftCards,point);//leftCards.PointCount(point);
            if (pointCount == 1) {
                QVector<Card> single =
Strategy(player,leftCards).findSamePointCards(point, 1);
                return single;
            } else if (pointCount == 2) {
                QVector<Card> pair =
Strategy(player,leftCards).findSamePointCards(point, 2);
                return pair;
            }
        }
    } else {
        for (CardPoint point = Card_3; point < Card_End; point = CardPoint(point + 1))
{
            int pointCount =
countOfPoint(leftCards,point);//leftCards.PointCount(point);
            if (pointCount == 1) {
                QVector<Card> single =
Strategy(player,leftCards).findSamePointCards(point, 1);
                return single;
            } else if (pointCount == 2) {
                QVector<Card> pair =
Strategy(player,leftCards).findSamePointCards(point, 2);
                return pair;
```

```
            }
        }
    }
    return QVector<Card>();
}
```

whetherToBeat 函数，判断能否接牌。首先判断有效牌是否是队友的牌，若是队友的牌，并且相对比较大，则不接；若相对不大，选择用小牌接牌，不用大牌。若是对家的牌，保证再接牌的同时保留“连对”、炸弹等特殊牌型。

```
bool Strategy::whetherToBeat(const QVector<Card> &myCards) {
    if (myCards.isEmpty()) return true;

    Player *hitPlayer = player->getPunchPlayer();

    if (player->getIsLandLord() == hitPlayer->getIsLandLord())        // punch 的是同家
    {
        QVector<Card> left = cards;
        for (int i = 0; i < myCards.size(); i++)
            left.removeOne(myCards[i]);
        if (CardGroups(left).getCardsType() != Group_Unknown) return true;
        CardPoint basePoint = CardPoint(CardGroups(myCards).getBasePoint());
        if (basePoint == Card_2 || basePoint == Card_SJ || basePoint == Card_BJ) {
            return false;
        }
    } else    // punch 的是对家
    {
        CardGroups myHand(myCards);

        if ((myHand.getCardsType() == Group_Triple_Single || myHand.getCardsType() ==
Group_Triple_Pair) &&
            (myHand.getBasePoint() == Card_2))    // 三个 2 就不打出去了
        {
            return false;
        }

        if (myHand.getCardsType() == Group_Pair && myHand.getBasePoint() == Card_2 &&
            hitPlayer->getHandCards().size() >= 10 && player->getHandCards().size() >=
5) {
            return false;
        }
    }

    return true;
}
```

3. 运行效果

图 3-1 主窗口

图 3-2 游戏窗口开始界面

图 3-3 抢地主

图 3-4 出牌——飞机带翅膀

图 3-5 出牌——三带二

图 3-6 出牌——连对

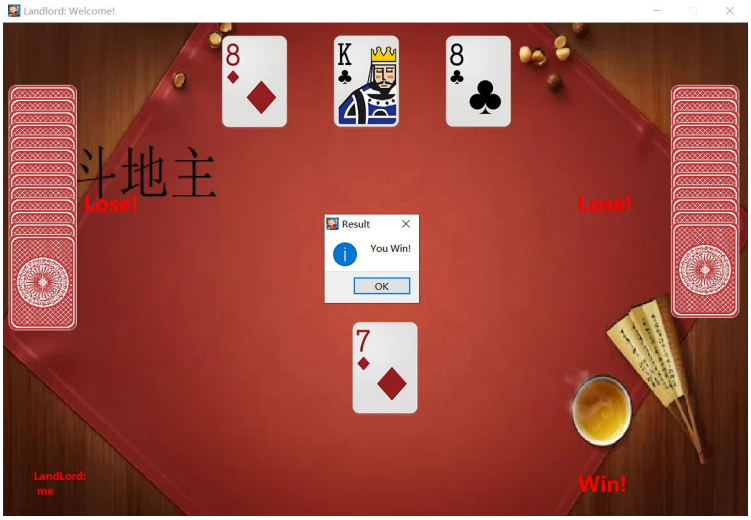

图 3-7 游戏结束

4. 总结

本次课程设计的工作量较大，在前期进行了合理规划的情况下，得以完整实现预期的大部分功能。在完成这次大作业的过程中，我们学习并能够熟练运用了 Qt 库，搭建出用户友好的交互界面；设计了斗地主游戏的总体框架结构，以及各个基类和派生类的功能和继承关系。在实际运用和艰难调试的过程中，我们对课程所学的知识和面向对象的思想有了更加深刻的理解与体会。

表 4-1 项目进度

日期	任务	备注
3.17-3.26	任务目标确定，初步规划	
3.26-4.15	实现出牌规则及输赢判定部分算法实现	
4.15-5.15	实现多用户同时交互	尽可能引进AI，实现单人人机模式
5.15-6.01	实现图形化界面，进行多模块合并操作	
6.01-6.10	撰写报告，并且准备展示	

5. 参考资料

[1] Socket 联机斗地主 (https://github.com/chenruijia120/Socket-Doudizhu)

[2] 联机斗地主 (https://github.com/340StarObserver/doudizhu)

[3] 单机斗地主 (https://github.com/windywater/LandlordCardGame)

[4] Landlord-THU (https://github.com/Cydiater/Landlord)