华南理工大学

South China University of Technology

《High-level Language Programming Project》Report

Project Name: Happy Landlord game

School : South China University of Technology

Major : Artificial Intelligence

Student Name : 刘兴琰 汪思佟 周奕 余幸桐

Teacher : 贾亚晖

Submission Date : 2024.1.13

Contents

1 Basic Function Realization

In this section we provide a summary of the completed project.

1.1 Functional realization

A stand-alone Happy Landlord game was implemented that focuses on users and bots playing against each other.

1.2 Technology realization

(1) Classes, objects, encapsulation, inheritance and polymorphism were applied

(2) We created 20 classes

(3) We have implemented at least two levels of inheritance hierarchy.

(4) We implemented a UI interface

(5) We implemented random file handling (write, read and update)

(6) Our code totaled over 2000 lines

2 Main Body of the Program

2.1 Environment configuration

Platform: Windows, Qt is cross-platform other platforms should be fine.

IDE：QtCreator

Qt Version Requirements: 5.10 and above

Qt version used to develop this project: 5.15.2, other versions may have problems with binary resource files (xxx.rcc) not loading.

Do not use Qt6 (the multimedia classes used no longer exist in Qt6)

Compilation package used: MinGW

About loading resource files:

Starting the program with QtCreator

Copy the resource file resource.rcc from the project directory to the generated build directory.Example of build directory name: build-Landlords-Desktop_Qt_5_15_2_MinGW_32_bit-Debug.

Double-click the .exe executable file directly to start the program.

Copy the resource file resource.rcc from the project directory to the same directory as the .exe file.

2.2 Rules of the Game

The rules of the game may vary slightly from region to region, and the game is realized according to the following rules.

2.2.1 Pattern

Pattern	Description
Joker Bomb	Big Joker+Small Joker， the biggest card
Bomb	Four cards of the same value (e.g. four 9s)
Single Card	Single card (e.g. 3 of hearts)
Pair Card	Two cards of the same value (e.g. 4 of clubs + 4 of diamonds)
Triple Card	Three cards of the same value (e.g. three Queens)
Three Bind One	Three cards of the same value + 1 single card, e.g. 333+6
Three Bind Pair	Three cards of the same value + 1 pair, e.g. 666+33
Sequence	Five or more consecutive singles, excluding 2's and double kings
Sequence Pair	Three or more consecutive pairs, excluding 2's and double kings
Plane	Two or more consecutive three-card tiles, excluding 2's and double kings
Plane_Two_Single/Plane_Two_Pair	Plane + same number of singles (or same number of pairs)
Four Bind Two/Four Bind Two Pair	Four equal cards + two hands (two singles or two doubles)
Bomb_Jokers_Pair/ Bomb_Jokers_Two_Single	Joker Bomb + two hands (two singles or two doubles)

2.2.2 Size of the Pattern

Joker Bomb is the largest, and can be pressed against any other hand.

Bombs are smaller than Joker Bomb and larger than other decks. It is all about the size of the cards according to the number of points they have when it comes to bombs.

All cards except Rocket and Bomb must be of the same suit and have the same total number of tiles in order to compare sizes.

Individual cards are compared to size by points (regardless of suit), in that order: Big Joker > Little Joker > 2 > A > K > Q > J > 10 > 9 > 8 > 7 > 6 > 5 > 4 > 3

Pairs, where all three cards with the same number of points are compared in size according to the number of points.

A Sequence is sized according to the number of points on the largest card.

Plane_Two_Single/Plane_Two_Pair and fours with twos are compared according to the three-of-a-kind and four-of-a-kind portions of them, and the brought-in side cards are not involved in the deck size comparison.

2.2.3 Game Characters

A total of three players are needed to participate in the game, and these three have two roles, which are:

Landlord: 1 person, three people bet to grab the landlord, the highest score can be called the landlord, the landlord's own gang.

Peasants: 2. The player who did not grab the landlord is a peasant, and these two are in a group.

2.2.4 Rules of the Game

(1) Dispatch

A deck of 54 cards, 17 for each player, and 3 base cards, which cannot be seen until the landlord is determined.

(2) Calling the Landlord / Robbing the Landlord

Calling the landlord takes turns in the order in which the cards are dealt, and each person can only call once.

When calling the landlord, you can call "1 point", "2 points", "3 points", "no call".

The player who steals the landlord after can only call for a higher score than the player in front of him or not.

The player who bets the most points at the end of the land grab is the landlord; if a player calls "3 points " then the call for the landlord is ended immediately and that player is the landlord; if none of them call, then the cards are re-distributed and the landlord is called again.

(3) The player who called the landlord in the first round

Since it is a stand-alone version of Landlord, the user player is directly designated as the first player to call the landlord

It can also be randomly selected by the system.

(4) Play Card

Hand the landowner the three base cards and show them for all to see.

The landlord plays his cards first, then in counterclockwise order, and when it's the user's turn to follow, the user has the option of " not playing" or playing a card that is larger than the previous player.

The game ends when a player runs out of cards.

2.2.5 The Outcome of the Game

The game ends when either player finishes playing, and the character represented by the player who finishes first wins:

Landowner wins if the landowner finishes first.

Farmer wins if either farmer finishes first.

2.2.6 Scoring Rules for Games

Bottom score: the betting score when calling the landlord, the available betting scores are: 1 point, 2 points, 3 points.

Multiplier: Initially 1, doubled for each Bomb or King Bomb (not counted if left in hand).

When a game is over, individual player scores are calculated based on low scores and multipliers:

Landlord wins:

Landlord: 2 * bottom score * multiplier

Peasant: - base points * multiplier

Farmer wins:

Landlord: -2 * base score * multiplier

Peasants: bottom score * multiplier

2.3 Main program introduction

This stand-alone game requires 7 types of classes, in order: Cards, Players, Windows, Game Controls, Threads, Game Strategy, and Audio.

Figure 1 Main structure of the project

Meanwhile, the following three major components need to be developed to complete the development of this game: cards, players, and windows.

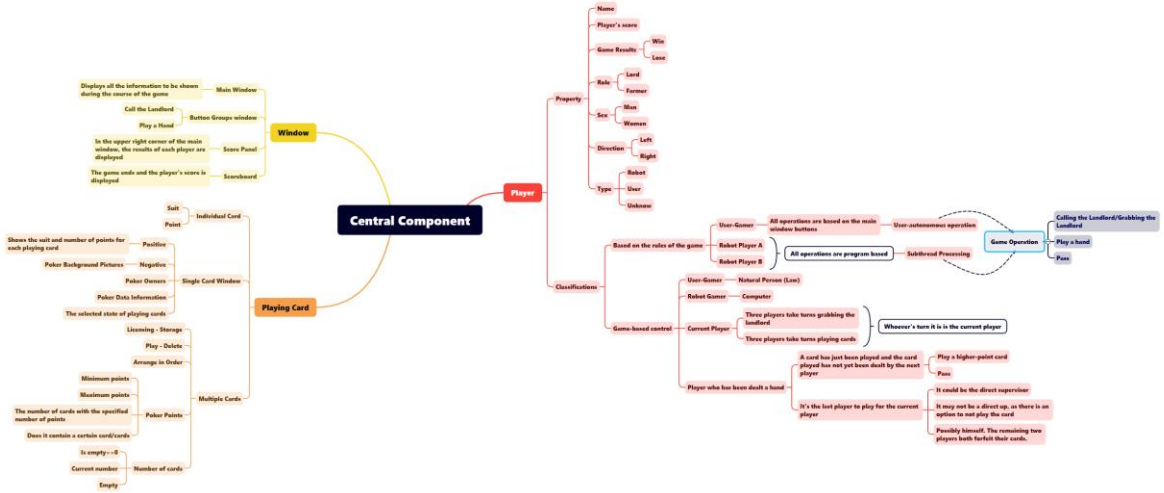


Figure 2 Three major components

2.3.1 Cards Class

There are two card classes: single card classes and multiple cards class.

✓ Card Class

Each playing card in the game has its own data attributes: suit and points, through this

class we can store and read the data of each playing card.

```
#ifndef CARD_H
#define CARD_H
#include <QVector>

class card
{
public:
    //花色
    enum CardSuit
    {
        Suit_begin,//为了方便后面的操作
        Diamond,
        Club,
        Heart,
        Spade,
        Suit_End

    };
```

```
//点数
enum CardPoint
{
    Card_begin,
    Card_3,
    Card_4,
    Card_5,
    Card_6,
    Card_7,
    Card_8,
    Card_9,
    Card_10,
    Card_J,
    Card_Q,
    Card_K,
    Card_A,
    Card_2,
    Card_SJ,//small joker
    Card_BJ,//big joker
    Card_end
};
```

```
    card(); //无参构造
    card(CardPoint point, CardSuit suit); //有参构造 接受点数和花色两个变量
    void setPoint(CardPoint point);//用来设置和获取点数和花色
    void setSuit(CardSuit suit);
    CardPoint point() const;
    CardSuit suit() const;

private:
    CardPoint m_point;
    CardSuit m_suit;
};
//对象比较
bool lessSort (const card& c1, const card& c2); //升序排序
bool greaterSort(const card& c1, const card& c2);  //降序排序
bool operator < (const card& c1,const card& c2); //用于两个card对象之间的比较
bool operator ==(const card& left, const card& right); //比较两个card是否相等
uint qHash(const card& card);//全局函数qHash
using CardList = QVector<card>;  //定义了一个名称为cardlist的类别名，表示一个QVector容器，用于存储多个Card对象
```

Figure 3 Card.h

lessSort function: used to compare the size of two playing cards, first compare the number of points, if the number of points is the same and then compare the suit, for sorting operations.

greaterSort function: used to compare the size of two playing cards, first compare the number of points, if the number of points is the same and then compare the suit, for sorting operations.

operator== function: overloaded the equality operator "==", used to compare two playing cards are equal.

qHash function: used to calculate the hash value of the playing cards object, in order to find and compare in the container.

operator< function: overloaded with the less-than operator "<", used to compare the size of two playing cards, which actually calls the lessSort function.

✓ Cards Class

During the course of the game, each player has multiple cards in their hands, and this category manages the cards in the hands of individual players during the course of the game.

```
#ifndef CARDS_H
#define CARDS_H
#include"card.h"
#include<QSet>
class Cards
{
public:
    enum SortType{Asc,Desc, NoSort};
    Cards();
    Cards(const card&card);
    //添加扑克牌
    void add(const card& card);//一开始这里没有加上const，所以一直报错
    void add(const Cards& cards);

    //一次性插入多个数据（操作符重载<<）
    Cards& operator <<(const card& card);
    Cards& operator <<(const Cards& cards);

    //删除扑克牌
    void remove(card& card);
    void remove(Cards &cards);

    //扑克牌的数量
    int cardCount();

    //是否为空
    bool isEmpty();

    //清空扑克牌
    void clear();
```

```
    //最大点数
    card::CardPoint maxPoint();

    //最小点数
    card::CardPoint minPoint();
    //指定点数的牌的数量
    int pointCount(card::CardPoint point);

    //某张牌是否在集合中
    bool contains(const card& card);
    bool contains(const Cards& cards);

    //随机取出一张扑克牌
    card takeRandCard();

    //QVector<Card> 把Qset变成这个QVector
    CardList toCardList(SortType tpye=Desc);

private:
    QSet<card> m_cards;//多张扑克牌添加


};
```

Figure 3 Cards.h

add function: this function is used to add a single playing card or a group of playing cards to a group, there are several overloaded versions, you can add a single playing card, a group of

playing cards or multiple groups of playing cards. This function is implemented by adding playing cards to a QSet container, ensuring that there are no duplicate playing cards in the group.

operator<<: This function overloads the bitwise operator "<<", which allows a single playing card or a group of playing cards to be added to a set. This function actually calls the add function, which returns a reference to the current object for chaining purposes.

max(min)Point function: This function returns the maximum (minimum) number of points of the playing cards in the group. It is achieved by iterating through all the playing cards in the QSet container and comparing their points.

pointCount function: This function returns the number of playing cards in the group with the specified number of points. It is realized by traversing all playing cards in the QSet container and counting the number of playing cards with the specified number of points.

contains function: this function is used to check whether the group contains the specified single playing card or a group of playing cards. It is realized by calling the contains function of the QSet container.

takeRandomCard function: this function takes a random playing card from the group and removes it from the group. It uses the QRandomGenerator class to generate a random number and then accesses the playing cards in the QSet container through an iterator to find the card corresponding to the random number and remove it from the container.

toCardList function: This function converts the playing cards in the group into a CardList (QVector) object, sorted by the specified sort type. It does this by iterating through all the playing cards in the QSet container, adding them to a CardList object, and then using the std::sort function to sort them according to the specified sort type.

2.3.2 Player Class

There are three player classes in the game: player class, robot class, and user player class. The robot class and the user player class are subclasses of the player class.

✓ Player Class

This class defines some properties and methods common to all players, such as:

Attributes: player's role, player type, player's gender, position of player's avatar, etc.

Methods: setting/getting player's name, gender, score, role; player switching during card playing; player grabbing landlord; player playing cards, etc.

```
#ifndef PLAYER_H
#define PLAYER_H

#include <QObject>
#include "cards.h"

class Player : public QObject
{
    Q_OBJECT
public:
    enum Role{Lord, Farmer};    // 角色
    enum Sex{Man, Woman};       // 性别
    enum Direction{Left, Right};    // 头像的显示方位
    enum Type{Robot, User, UnKnow}; // 玩家的类型
    explicit Player(QObject *parent = nullptr);
    explicit Player(QString name, QObject *parent = nullptr);

    void setName(QString name);
    QString getName();

    void setRole(Role role);
    Role getRole();

    void setSex(Sex sex);
    Sex getSex();

    void setDirection(Direction direction);
    Direction getDirection();

    void setType(Type type);
    Type getType();

    void setScore(int score);
    int getScore();

    void setWin(bool flag);
    bool isWin();

    void setPrevPlayer(Player* player);
    void setNextPlayer(Player* player);
    Player* getPrevPlayer();
    Player* getNextPlayer();

    void grabLordBet(int point);
```

Figure 4 Player.h

PrepareCallLord Function: before grabbing the landlord, the player can do some preparations in this function, such as displaying relevant information, calculating the strategy of grabbing the landlord and so on.

PreparePlayHand Function: before playing cards, players can do some preparations in this function, such as displaying the current hand, calculating the strategy of playing cards, etc.

ThinkCallLord Function: when robbing the landlord, players need to think and make decisions in this function, choosing whether to rob the landlord and the score of robbing the landlord.

ThinkPlayHand Function: when playing cards, players need to think and make decisions in this function to choose the cards to play.

The above four functions are all virtual functions. These four dummy functions provide a framework for players to implement specific land grabbing and card playing operations according to their own needs and strategies. Different types of players can override these functions to implement different logic, thus realizing different game strategies and player behaviors.

The class also defines three signals: notifyGrabLordBet, notifyPlayHand, and notifyPickCards, which are used to notify the relevant actions.

✓ Robot Player Class:

Inherits the parent class properties and methods and overrides the parent class virtual functions for grabbing the landlord and playing cards.

In the Robot class, the specific logic of robbing the landlord and playing cards of the robot player is realized by overriding the virtual function in the Player class. In the stage of robbing the landlord, the robot player selects the appropriate landlord multiplier to rob the landlord according to certain weight calculation. In the card playing phase, the robot player calculates the appropriate combination of cards according to a certain strategy, and plays the cards through the playHand function.

```
#ifndef ROBOT_H
#define ROBOT_H

#include "player.h"
#include <QObject>

class Robot : public Player
{
    Q_OBJECT
public:
    using Player::Player;
    explicit Robot(QObject *parent = nullptr);

    void prepareCallLord() override;
    void preparePlayHand() override;

    void thinkCallLord() override;
    void thinkPlayHand() override;
};
```

Figure 5 Robot.h

✓ UserPlayer Class：

Inherits the parent class properties and methods and overrides the parent class virtual functions for grabbing the landlord and playing cards.

This class inherits the Player class, which has the basic attributes and behaviors of a player. prepareCallLord() and preparePlayHand() functions in the UserPlayer class can be implemented according to the needs of the game, which can be used to perform relevant operations or logic in the landlord grabbing and card playing phases of the game for the user player. By defining this class, the game can realize the participation and interaction of the user player, enabling the user to grab the landlord and play cards and interact with other players in the game.

```
#ifndef USERPLAYER_H
#define USERPLAYER_H

#include "player.h"
#include <QObject>

class UserPlayer : public Player
{
    Q_OBJECT
public:
    using Player::Player;
    explicit UserPlayer(QObject *parent = nullptr);

    void prepareCallLord() override;
    void preparePlayHand() override;

signals:
    void startCountDown();
};

#endif // USERPLAYER_H
```

Figure 6 UserPlayer.h

2.3.3 Window Class

✓ Loading animation window at the start of the game: Loading

By inheriting the QWidget class and overriding the paintEvent function, you can implement a customized Loading window widget that can draw customized images on the interface.

```
#ifndef LOADING_H
#define LOADING_H

#include <QWidget>

class Loading : public QWidget
{
    Q_OBJECT
public:
    explicit Loading(QWidget *parent = nullptr);

signals:
protected:
    void paintEvent(QPaintEvent* event);

private:
    QPixmap m_bk;
    QPixmap m_progress;
    int m_dist = 15;
};

#endif // LOADING_H
```

Figure 7 Loading.h

✓ GamePanel Class

The GamePanel class is an implementation file for a game interface, which contains a constructor, destructor, and several functional functions. The constructor is mainly used to initialize the interface and game-related objects, while the destructor is used to release the interface objects. Other functional functions include initializing the cards, player interface, game scene and other elements, handling the changes of the game state, and realizing the functions such as card movement, player playing, countdown, and so on. The display and interaction logic of the interface is realized by connecting the signal and slot functions.

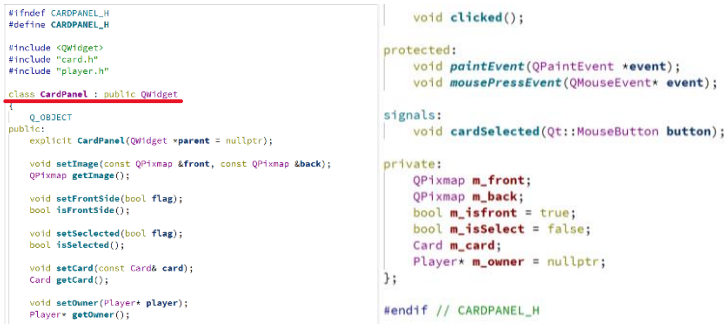
```
#ifndef GAMEPANEL_H
#define GAMEPANEL_H

#include <QLabel>
#include <QMainWindow>
#include <QMap>
#include "cardpanel.h"
#include "gamecontrol.h"
#include "animationwindow.h"
#include "player.h"
#include "countdown.h"
#include "bgmcontrol.h"

QT_BEGIN_NAMESPACE
namespace Ui { class GamePanel; }
QT_END_NAMESPACE

class GamePanel : public QMainWindow
{
    Q_OBJECT

public:
    GamePanel(QWidget *parent = nullptr);
    ~GamePanel();
    enum AnimationType{ShunZi, LianDui, Plane, JokerBomb, Bomb, Bet};
    void gameControlInit();
    void updatePlayerScore();
    void initCardMap();
    void cropImage(QPixmap& pix, int x, int y, Card& c);
    void initButtonsGroup();
    void initPlayerContext();
    void initGameScene();
    void gameStatusPrecess(GameControl::GameStatus status);
    void startDispatchCard();
    void cardMoveStep(Player* player, int curPos);
    void disposeCard(Player* player, const Cards& cards);
    void updatePlayerCards(Player* player);
    QPixmap loadRoleImage(Player::Sex sex, Player::Direction direct, Player::Role role);

    void onDispatchCard();
    void onPlayerStatusChanged(Player* player, GameControl::PlayerStatus status);
    void onGrabLordBet(Player* player, int bet, bool flag);
    void onDisposePlayHand(Player* player, const Cards& cards);
    void onCardSelected(Qt::MouseButton button);
    void onUserPlayHand();
    void onUserPass();
    void showAnimation(AnimationType type, int bet = 0);
    void hidePlayerDropCards(Player* player);
    void showEndingScorePanel();
    void initCountDown();

protected:
    void paintEvent(QPaintEvent* ev);
    void mouseMoveEvent(QMouseEvent* ev);


private:
    enum CardAlign{Horizontal, Vertical};
    struct PlayerContext
    {
        QRect cardRect;
        QRect playHandRect;
        CardAlign align;
        bool isFrontSide;
        QLabel* info;
        QLabel* roleImg;
        Cards lastCards;
    };

    Ui::GamePanel *ui;
    QPixmap m_bkImage;
    GameControl* m_gameCtl;
    QVector<Player*> m_playerList;
    QMap<Card, CardPanel*> m_cardMap;
    QSize m_cardSize;
    QPixmap m_cardBackImg;
    QMap<Player*, PlayerContext> m_contextMap;
    CardPanel* m_baseCard;
    CardPanel* m_moveCard;
    QVector<CardPanel*> m_last3Card;
    QPoint m_baseCardPos;
    GameControl::GameStatus m_gameStatus;
    QTimer* m_timer;
    AnimationWindow* m_animation;
    CardPanel* m_curSelCard;
    QSet<CardPanel*> m_selectCards;
    QRect m_cardsRect;
    QHash<CardPanel*, QRect> m_userCards;
    CountDown* m_countDown;
    BGMControl* m_bgm;
};
#endif // GAMEPANEL_H
```

Figure 8 GamePanel.h

✓ Single playing card window: CardPanel

Each playing card corresponds to one of these window objects.

The CardPanel class implements a panel for displaying playing cards by inheriting from the QWidget class. It is possible to set the front and back images of the panel and control the displayed image by setting the front and back properties of the panel. It is also possible to set the playing card and the owner of the panel, and to handle the panel's interactive actions through click events.



```
#ifndef CARDPANEL_H
#define CARDPANEL_H

#include <QWidget>
#include "card.h"
#include "player.h"

class CardPanel : public QWidget
{
    Q_OBJECT
public:
    explicit CardPanel(QWidget *parent = nullptr);

    void setImage(const QPixmap &front, const QPixmap &back);
    QPixmap getImage();

    void setFrontSide(bool flag);
    bool isFrontSide();

    void setSeclected(bool flag);
    bool isSelected();

    void setCard(const Card& card);
    Card getCard();

    void setOwner(Player* player);
    Player* getOwner();
    void clicked();

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

signals:
    void cardSelected(Qt::MouseButton button);

private:
    QPixmap m_front;
    QPixmap m_back;
    bool m_isfront = true;
    bool m_isSelect = false;
    Card m_card;
    Player* m_owner = nullptr;
};

#endif // CARDPANEL_H
```

Figure 9 CardPanel.h

✓ Customized buttons: MyButton

This class performs button beautification. The class inherits from QPushButton and overrides the mouse event and draw event functions. By overriding these event functions, it realizes the picture switching and drawing of the button in different states.

```cpp
#ifndef MYBUTTON_H
#define MYBUTTON_H

#include <QPushButton>

class MyButton : public QPushButton
{
    Q_OBJECT
public:
    explicit MyButton(QWidget *parent = nullptr);

    void setImage(QString normal, QString hover, QString pressed);

signals:

protected:
    void mousePressEvent(QMouseEvent* ev);
    void mouseReleaseEvent(QMouseEvent* ev);
    void enterEvent(QEvent* ev);
    void leaveEvent(QEvent* ev);
    void paintEvent(QPaintEvent* ev);

private:
    QString m_normal;
    QString m_hover;
    QString m_pressed;
    QPixmap m_pixmap;

};
```

Figure 10 MyButton.h

✓ Button Group window in the main game window: ButtonGroup

For user players to grab the landlord and play cards.

ButtonGroup is a class that inherits from QWidget to manage the button elements on the game interface. The display and interaction logic of the buttons is implemented by initializing their images and connecting signal and slot functions, and controlling the visibility of the buttons based on the panel type and betting score.

```cpp
#ifndef BUTTONGROUP_H
#define BUTTONGROUP_H

#include <QWidget>

namespace Ui {
class ButtonGroup;
}

class ButtonGroup : public QWidget
{
    Q_OBJECT

public:
    enum Panel{Start, PlayCard, PassOrPlay, CallLord, Empty};
    explicit ButtonGroup(QWidget *parent = nullptr);
    ~ButtonGroup();

    void initButtons();
    void selectPanel(Panel type, int bet = 0);

signals:
    void startGame();
    void playHand();
    void pass();
    void betPoint(int bet);

private:
    Ui::ButtonGroup *ui;
};

#endif // BUTTONGROUP_H
```

Figure 11 ButtonGroup.h

✓ Game Score Panel Window: ScorePanel

Displays a player's score in the upper right corner of the window.

This class inherits from QWidget and is a class for displaying information about the game score. The display and styling of the score panel is achieved by setting the score and font properties.

```cpp
#ifndef SCOREPANEL_H
#define SCOREPANEL_H

#include <QLabel>
#include <QWidget>

namespace Ui {
class ScorePanel;
}

class ScorePanel : public QWidget
{
    Q_OBJECT

public:
    enum FontColor{Black, White, Red, Blue, Green};
    explicit ScorePanel(QWidget *parent = nullptr);
    ~ScorePanel();

    void setScores(int left, int right, int user);
    void setMyFontSize(int point);
    void setMyFontColor(FontColor color);

private:
    Ui::ScorePanel *ui;
    QVector<QLabel*> m_list;
};
```

Figure 12 ScorePanel.h

✓ Game over player's score window: EndingPanel

Game end popup to show each player's score.

Inherited from QWidget, this class is a class for displaying the end of game result and score. By setting the style of the title image, score and buttons, it realizes the display and interaction logic of the end of game panel.

```
#ifndef ENDINGPANEL_H
#define ENDINGPANEL_H

#include "scorepanel.h"

#include <QLabel>
#include <QPushButton>
#include <QWidget>

class EndingPanel : public QWidget
{
    Q_OBJECT
public:
    explicit EndingPanel(bool isLord, bool isWin, QWidget *parent = nullptr);

    void setPlayerScore(int left, int right, int me);

signals:
    void continueGame();

protected:
    void paintEvent(QPaintEvent* ev);

private:
    QPixmap m_bk;
    QLabel* m_title;
    ScorePanel *m_score;
    QPushButton* m_continue;

};
```

Figure 13 EndingPanel.h

✓ Countdown window: Countdown

This class inherits from QWidget, it is a control class for displaying countdown timer. By setting the time of countdown and loading the corresponding picture, it realizes the display and logic of countdown control.

The countdown time for the user player to play the card, if more than 20 cards are not played, the system defaults not to play and skips directly.

```
#ifndef COUNTDOWN_H
#define COUNTDOWN_H

#include <QTimer>
#include <QWidget>

class CountDown : public QWidget
{
    Q_OBJECT
public:
    explicit CountDown(QWidget *parent = nullptr);

    void showCountDown();
    void stopCountDown();

signals:
    void notMuchTime();
    void timeout();

protected:
    void paintEvent(QPaintEvent* ev);

private:
    QPixmap m_clock;
    QPixmap m_number;
    QTimer* m_timer;
    int m_count;

};

#endif // COUNTDOWN_H
```

Figure 14 CountDown.h

✓ Special effects animation window: AnimationWindow

AnimationWindow's custom QWidget class for displaying in-game animation effects: King Bomb, Bombs, Airplanes, Jokers, Pairs, and so on.

```h
#ifndef ANIMATIONWINDOW_H
#define ANIMATIONWINDOW_H

#include <QWidget>

class AnimationWindow : public QWidget
{
    Q_OBJECT
public:
    enum Type{Sequence, Pair};
    explicit AnimationWindow(QWidget *parent = nullptr);

    // 显示下注分数
    void showBetScore(int bet);
    // 显示数字炸弹
    void showSequence(Type type);
    // 显示火箭
    void showJokerBomb();
    // 显示炸弹
    void showBomb();
    // 显示飞机
    void showPlane();

signals:

protected:
    void paintEvent(QPaintEvent* ev);

private:
    QPixmap m_image;
    int m_index = 0;
    int m_x = 0;

};
```

Figure 15 AnimationWindow.h

2.3.4 Game Control Class

GameControl is one of the more important classes in the game, it manages and controls a lot of core data in the game:

- ✓ Initialization of the player object.
- ✓ Initialization of all playing card data.
- ✓ Game state: dealing, calling landlord, playing cards.
- ✓ Player states: considering calling the landlord, considering playing cards, some player wins.
- ✓ Game data: player card data, player betting and score doubling data, player scoring.
- ✓ Game reset and cards dealt.

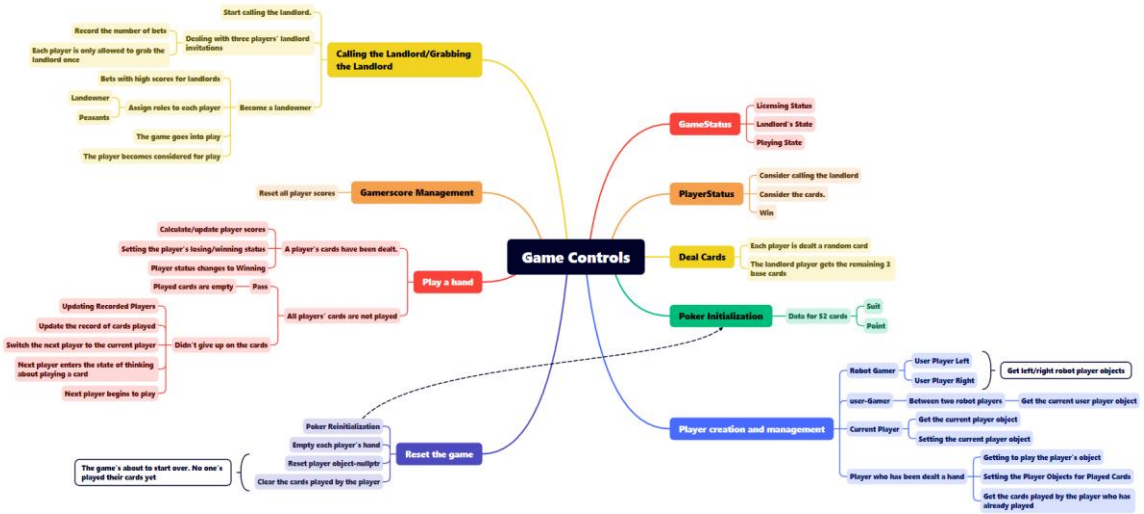


Figure 16 Game Controls

The GameControl class implements a simple landlord game logic by managing player objects and controlling the game flow. It can initialize player objects, deal cards, handle player's landlord calling and card playing operations, and judge the winners and losers according to the game rules and update the player's score. At the same time, the class also provides some auxiliary functions for obtaining information about the player and the playing cards.

Figure 17 GameControl.h

2.3.5 Game Strategy Class

There are two game strategy categories in the game: the card playing category and the card strategy category.

✓ PlayHand Class

In accordance with the rules of the game of "Fight the Landlord", it can obtain information about the card types and points of the cards in the player's hand, and can compare the sizes of the sorting based on the rules of the game of "Fight the Landlord".

Figure 18 PlayHand.h

✓ Strategy Class

This class is used to formulate the card strategy, is the most complex logic in the entire project, the largest amount of code in a class, through a series of algorithms implemented in this class can be realized according to the actual situation of the robot player to call the landlord / grab the landlord function, the robot player's card function.

In this class. According to the specific rules and conditions, the selection and judgment of different types of decks are realized. Through these functions, it can realize the automatic selection of the optimal deck to play cards, and decide whether to suppress the hanging cards according to the current situation.

Figure 19 Strategy.h

2.3.6 Thread Class

There are two thread classes in the game, the robber thread class and the thread class that plays the cards.

✓ RobotGrapLord Thread Class

Robot player grab the landlord to create a sub-thread, the end of the landlord to destroy this sub-thread.

This class is inherited from QThread and is used to execute the robot land grabbing logic in a new thread by calling the land grabbing function of the Player object after a period of dormancy in the run() function, which realizes the function of the robot land grabbing.

Figure 20 RobotGrapLord.h

✓ RobotPlayHand Thread Class

Robot player out of cards to create a sub-thread, out of cards to destroy this sub-thread.

This class inherits from QThread and is used to execute the logic of the robot playing cards in a new thread, which is realized by calling the Player object's card playing function after a period of dormancy in the run() function.

```cpp
#ifndef ROBOTPLAYHAND_H
#define ROBOTPLAYHAND_H

#include "player.h"

#include <QThread>

class RobotPlayHand : public QThread
{
    Q_OBJECT
public:
    explicit RobotPlayHand(Player* player, QObject *parent = nullptr);

signals:
protected:
    void run() override;

private:
    Player* m_player;

};
```

Figure 21 RobotPlayHand.h

In this case, the run() function is a rewritten thread execution function that represents the logic to be executed in a new thread. In the function, the thread is made to sleep for 2000 milliseconds (2 seconds) by calling the msleep() function, and then m_player's thinkPlayHand() function is called to carry out the robot's play logic.

2.3.7 Game Audio Class

Game Audio Class BGMControl is mainly used to control the playback of various music in the game:

- ✓ Background Music
- ✓ Landlord Grabbing / Landlord Calling Cue Music
- ✓ Dealer music, Card Selection Music
- ✓ Cue music for card suit and number of points
- ✓ Remaining card count (when only one or two cards are left) hints

This class inherits from the QObject class and is a QObject object.

The BGMControl class implements a customized Loading window widget by inheriting the QWidget class and overriding the paintEvent function, which can draw customized images on the interface. And by using multiple `QMediaPlayer` objects and conditional statements, it can play different music according to different conditions.

Figure 22 BGMControl.h

2.4 Write, Read and Update Random Files

2.4.1 Write and Update File

We create a file output stream object file and specify the file name as "scores.txt".

```
// 创建并打开一个本地.txt文件
    std::ofstream file("scores.txt");
```

Then we write the calculated values of left, right and me to the file.

```
// 将left、right、me的值分别写入文件的三行
    file << "分数结算：" << std::endl;
    file << "左侧机器人：" << left << std::endl;
    file << "右侧机器人：" << right << std::endl;
    file << "我：" << me << std::endl;
    file.close();// 关闭文件
```

Figure 23 File Write and Update

2.4.2 Read File

We call the std::ifstream class to read the local text file "scores.txt" that we created earlier. It first creates a file input stream object called readFile and opens the "scores.txt" file for reading.

```
// 读取文件
  std::ifstream readFile("scores.txt");
```

Next, we read the first header line of the file using std::getline(readFile, line) and skip it. Then, the second line of the file is read using std::getline(readFile, line) and stored in the string variable line. Then, use line.find(":") to find the position of the colon in the second line and use line.substr(line.find(":") + 2) to intercept the substring after the colon. This substring is then converted to an integer using std::stoi and the result is stored in the variable readLeft. Similarly, the third and fourth lines of the file are read using the same logic and the result is stored in the variables readRight and readMe, respectively.

Finally, we call m_score->setScores(readLeft, readRight, readMe) to pass the read values to the m_score object's setScores function for output to the ui interface.

```
    if (readFile.is_open()) {
        std::string line;
        // 读取文件内容并解析值// 第一行为标题，可以跳过
        std::getline(readFile, line);
        // 读取左侧机器人的值
        std::getline(readFile, line);
        int readLeft = std::stoi(line.substr(line.find(":") + 2));
        // 读取右侧机器人的值
        std::getline(readFile, line);
        int readRight = std::stoi(line.substr(line.find(":") + 2));
        // 读取我的值
        std::getline(readFile, line);
        int readMe = std::stoi(line.substr(line.find(":") + 2));
        // 关闭读取文件
        readFile.close();
// 现在，readLeft、readRight、readMe 分别包含了读取的值}
    else {// 文件打开失败的处理std::cerr << "无法打开文件 scores.txt" <<
std::endl;}
    m_score->setScores(readLeft, readRight, readMe);}
```

3 Code Test and Results

The Happy Landlord game is able to run successfully. We have implemented all kinds of basic operations of the game, including card dealing, card playing, time reminder, scoring, random generation of backgrounds and characters, and generation of various special effects.

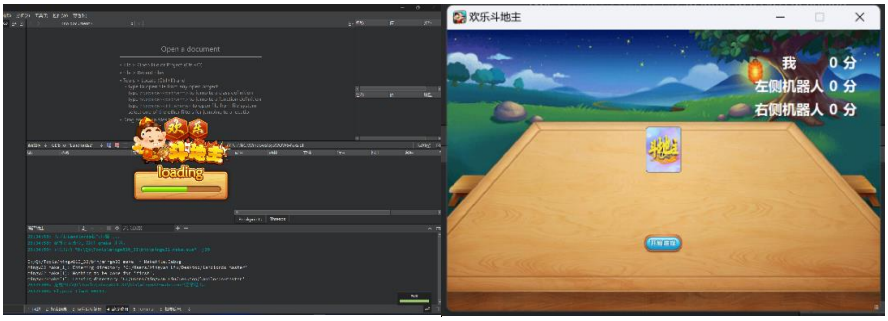


Figure 24 Enter the game interface

Figure 25 Course of games

Figure 26 Game Over Window Display

4 Summary

In this group collaborative project, we have implemented the basic functions of the Happy Landlord game, but due to the technical level and time constraints, our project still has a lot of room for improvement.

In the course of this project, we learned how to configure the Qt environment and how to use Qt for the project. In the process of constructing the program framework, we have gained a deep understanding of inheritance and derivation, class encapsulation, and object orientation that we have learned in this semester, and we have applied what we have learned to the development of the program, which has further improved our programming level. In terms of group cooperation, although we have carried out a lot of activities that require cooperation, this kind of cooperation is closer to the cooperation in the future work, where we have a clear division of labor and higher efficiency

There are still many areas where our project can be further improved in the future to enhance user experience and functionality expansion. On the one hand, we can consider developing the project into an online game to increase the interactivity and competition among users. Through network connection, users can play against other players and enjoy a more exciting and interesting gaming experience.

On the other hand, we can introduce a user account management system that allows users to create their own accounts and personalize their settings. This allows users to save their progress and scores, compare their rankings with other players, and even unlock some special props or rewards.

In addition to the above features, we can also consider integrating many different types of landlord games. By adding different rules and playing styles, we can provide more diversified game choices to meet different users' needs and preferences.

5 Reference

[1] https://subingwen.cn/project/landlord/index.html