南开大学 计算机大类

李子熹

2410818

3205

2025年5月13日

高级语言程序设计

实验报告

目录

[一. 作业题目 2](#_Toc27048)

[二. 开发软件 2](#_Toc11416)

[三. 课题要求 3](#_Toc2555)

[四. 主要流程 3](#_Toc29611)

[1．系统架构与类设计 3](#_Toc5056)

[（1）核心类图 3](#_Toc13314)

[（2）抽象基类Player 4](#_Toc18922)

[（3）人类玩家HumanPlayer（用户交互） 4](#_Toc31636)

[（4） AI 玩家AIPlayer（策略实现） 5](#_Toc17951)

[2. 卡牌系统与布局实现 5](#_Toc23256)

[（1）Card类（数据模型） 5](#_Toc18132)

[（2）自定义布局CardLayout（卡牌重叠排列） 6](#_Toc11993)

[（3）卡牌控件CardWidget（UI 显示） 6](#_Toc13201)

[3. 牌型判断与比较（CardTypeJudger） 6](#_Toc27907)

[（1）王炸判断 6](#_Toc12299)

[（2）炸弹判断 7](#_Toc1613)

[（3）顺子判断 7](#_Toc7463)

[4. 游戏流程控制（MainWindow） 7](#_Toc25766)

[（1）发牌逻辑 7](#_Toc7991)

[（2）出牌处理 8](#_Toc23756)

[五.单元测试 8](#_Toc15077)

[1. 测试目标 8](#_Toc1)

[2. 核心测试用例 8](#_Toc26280)

[（1）牌型判断测试 8](#_Toc21556)

[（2）AI 策略测试 9](#_Toc7355)

[（3）UI 交互测试 9](#_Toc28309)

[六、收获与反思 9](#_Toc6634)

[1. 技术实现亮点 9](#_Toc2993)

[2. 代码优化空间 9](#_Toc6071)

[3. 调试经验 10](#_Toc22920)

[七、总结 10](#_Toc1178)

高级语言程序设计大作业实验报告

**一. 作业题目**

基于Qt 的landlord game

1. **开发软件**

|  |  |  |
| --- | --- | --- |
| Qt Creator | Qt 6.x | 集成开发环境，用于 UI 设计与代码编写 |
| C++ | C++17 | 编程语言 |
| Qt 框架 | 6.5.2 | 提供 GUI 组件、信号槽机制等功能 |

**三. 课题要求**

1. ****面向对象设计****：通过抽象类、继承和多态实现玩家、卡牌等核心模块，分离数据模型与业务逻辑。
2. ****图形界面****：使用 Qt 的 UI 设计工具和自定义布局，实现手牌显示、出牌交互、游戏状态提示等功能。
3. ****游戏逻辑****：
   1. 卡牌生成与洗牌、发牌逻辑；
   2. 牌型判断（单牌、对子、炸弹、顺子等）与比较；
   3. 人类玩家交互与 AI 玩家策略。
4. ****代码规范****：模块化设计，代码结构清晰，注释完整，便于扩展（如新增牌型或玩家类型）。

**四. 主要流程**

**1．系统架构与类设计**

**（1）核心类图**

**classDiagram**

**Player <|-- HumanPlayer**

**Player <|-- AIPlayer**

**CardWidget --> Card**

**CardLayout --> CardWidget**

**CardTypeJudger --> Card**

**Deck --> Card**

**MainWindow --> HumanPlayer**

**MainWindow --> AIPlayer**

**MainWindow --> Deck**

**MainWindow --> CardTypeJudger**

**MainWindow --> CardLayout**

**class Player {**

**+playCards(lastCards: QList<Card\*>): QList<Card\*> [pure virtual]**

**+addCard(card: Card\*): void**

**+removeCard(card: Card\*): void**

**+cards(): QList<Card\*>**

**}**

**class HumanPlayer {**

**+playCards(lastCards: QList<Card\*>): QList<Card\*>**

**-m\_selectedCards: QList<Card\*>**

**-m\_playConfirmed: bool**

**+requestPlay(lastCards: QList<Card\*>): signal**

**}**

**class AIPlayer {**

**+playCards(lastCards: QList<Card\*>): QList<Card\*>**

**-findBestCardsToPlay(lastCards: QList<Card\*>): QList<Card\*>**

**-findSmallestSingle(): QList<Card\*>**

**-findBomb(): QList<Card\*>**

**}**

**class Card {**

**-Suit suit**

**-Rank rank**

**+suitString(): QString**

**+rankString(): QString**

**}**

**class CardTypeJudger {**

**+judgeCardType(cards: QList<Card\*>): CardType**

**+isBetter(cardsA: QList<Card\*>, cardsB: QList<Card\*>): bool**

（2）抽象基类Player

// player.hclass Player : public QObject {

Q\_OBJECTpublic:

virtual QList<Card\*> playCards(QList<Card\*> lastCards) = 0; // 纯虚函数，子类必须实现

void addCard(Card \*card);

void removeCard(Card \*card);

QList<Card\*> cards() const { return m\_cards; }protected:

QList<Card\*> m\_cards; // 玩家手牌列表

bool m\_isLandlord; // 是否为地主};

### **（3）人类玩家HumanPlayer（用户交互）**

// humanplayer.cpp

QList<Card\*> HumanPlayer::playCards(QList<Card\*> lastCards) {

m\_lastCards = lastCards;

m\_selectedCards.clear();

m\_playConfirmed = false;

emit requestPlay(lastCards); // 触发UI更新，提示用户选牌

while (!m\_playConfirmed) {

qApp->processEvents(); // 处理UI事件，保持界面响应

}

return m\_selectedCards; // 返回用户选择的手牌

}

### **AI 玩家AIPlayer（策略实现）**

优先出炸弹 / 王炸，其次根据牌型选择最小或更大的手牌：

// aiplayer.cpp

QList<Card\*> AIPlayer::findBestCardsToPlay(QList<Card\*> lastCards) {

if (lastCards.isEmpty()) { // 首轮出牌，出最小单牌

return findSmallestSingle();

}

// 检查炸弹或王炸

QList<Card\*> bomb = findBomb();

if (!bomb.isEmpty() && CardTypeJudger::isBetter(bomb, lastCards)) {

return bomb;

}

QList<Card\*> rocket = findRocket();

if (!rocket.isEmpty()) {

return rocket; // 王炸最大，直接出牌

}

// 处理单牌/对子

CardTypeJudger::CardType lastType = CardTypeJudger::judgeCardType(lastCards);

switch (lastType) {

case CardTypeJudger::Single:

return findLargerSingle(lastCards).isEmpty() ? findSmallestSingle() : findLargerSingle(lastCards);

case CardTypeJudger::Pair:

return findLargerPair(lastCards).isEmpty() ? findSmallestPair() : findLargerPair(lastCards);

default:

return findSmallestSingle(); // 暂不处理复杂牌型，出最小单牌

}}

### **2. 卡牌系统与布局实现**

### **（1）Card类（数据模型）**

// card.hclass Card : public QObject {

Q\_OBJECTpublic:

enum Suit { Spade, Heart, Club, Diamond, Joker }; // 花色枚举

enum Rank {

Three=3, Four, Five, Six, Seven, Eight, Nine, Ten,

Jack, Queen, King, Ace, Two, SmallJoker=16, BigJoker=17 // 点数枚举（3到王）

};

QString suitString() const; // 返回花色名称（如"Spade"）

QString rankString() const; // 返回点数名称（如"3", "A"）};

### **（2）自定义布局CardLayout（卡牌重叠排列）**

// cardlayout.cppvoid CardLayout::setGeometry(const QRect &rect) {

QLayout::setGeometry(rect);

int x = rect.x();

for (int i = 0; i < m\_items.size(); ++i) {

QLayoutItem \*item = m\_items[i];

QSize size = item->sizeHint();

// 每张卡牌向右重叠5像素（m\_spacing=5）

item->setGeometry(QRect(x, rect.y(), size.width(), size.height()));

x += size.width() - m\_spacing; // 计算下一张卡牌的x坐标

}}

### **（3）卡牌控件CardWidget（UI 显示）**

// cardwidget.cppvoid CardWidget::paintEvent(QPaintEvent \*event) {

QPainter painter(this);

painter.setRenderHint(QPainter::Antialiasing);

// 绘制卡牌边框和背景

painter.fillRect(rect(), Qt::white);

painter.drawRect(rect());

// 绘制选中状态（浅蓝色背景）

if (m\_selected) {

painter.fillRect(rect().adjusted(2, 2, -2, -2), QColor(220, 230, 255));

}

// 绘制点数（居中）和花色（左上角）

painter.drawText(rect(), Qt::AlignCenter, m\_card->rankString());

QColor suitColor = (m\_card->suit() == Card::Joker) ? Qt::blue :

(m\_card->suit() == Card::Spade || m\_card->suit() == Card::Club) ? Qt::black : Qt::red;

painter.setPen(suitColor);

painter.drawText(rect().adjusted(5, 5, -5, -5), Qt::AlignTop | Qt::AlignLeft, m\_card->suitString());}

### **3. 牌型判断与比较（CardTypeJudger）**

### **（1）王炸判断**

// cardtypejudger.cppif (size == 2) { // 王炸必为2张牌

bool hasSmall = false, hasBig = false;

for (Card \*card : cards) {

if (card->rank() == Card::SmallJoker) hasSmall = true;

if (card->rank() == Card::BigJoker) hasBig = true;

}

if (hasSmall && hasBig) return Rocket; // 同时存在大小王}

### **（2）炸弹判断**

if (size == 4) { // 炸弹必为4张牌

int rank = cards.first()->rank();

bool allSame = std::all\_of(cards.begin(), cards.end(),

[rank](Card\* c) { return c->rank() == rank; });

if (allSame) return Bomb; // 4张点数相同}

### **（3）顺子判断**

if (size >= 5 && size <= 12) { // 顺子长度5-12

QList<int> ranks;

for (Card \*card : cards) ranks.append(card->rank());

std::sort(ranks.begin(), ranks.end());

bool isContinuous = true;

for (int i = 1; i < ranks.size(); ++i) {

if (ranks[i] != ranks[i-1] + 1) { // 检查点数连续

isContinuous = false;

break;

}

}

if (isContinuous &&

!ranks.contains(Card::SmallJoker) &&

!ranks.contains(Card::BigJoker)) { // 不含大小王

return Straight;

}}

### **4. 游戏流程控制（MainWindow）**

### **（1）发牌逻辑**

// mainwindow.cppvoid MainWindow::dealCards() {

m\_deck->initialize(); // 初始化牌堆（54张牌）

m\_deck->shuffle(); // 洗牌

for (int i = 0; i < 17; ++i) { // 每人发17张

for (Player \*player : m\_players) {

player->addCard(m\_deck->drawCard());

}

}

// 3张底牌给地主

int landlordIndex = QRandomGenerator::global()->bounded(3);

for (int i = 0; i < 3; ++i) {

m\_players[landlordIndex]->addCard(m\_deck->drawCard());

}

updatePlayerCardsDisplay(); // 刷新手牌显示}

### **（2）出牌处理**

void MainWindow::on\_playButton\_clicked() {

QList<Card\*> selected = getSelectedCards(); // 获取用户选中的卡牌

CardTypeJudger::CardType type = CardTypeJudger::judgeCardType(selected);

if (type == CardTypeJudger::Invalid) {

QMessageBox::warning(this, "错误", "非法牌型！");

return;

}

if (!m\_lastPlayedCards.isEmpty() &&

!CardTypeJudger::isBetter(selected, m\_lastPlayedCards)) {

QMessageBox::warning(this, "错误", "无法大过上一手牌！");

return;

}

onPlayerPlayedCards(0, selected); // 人类玩家出牌}

**五.单元测试**

### **1. 测试目标**

验证牌型判断准确性、AI 策略合理性、UI 交互正确性。

### **2. 核心测试用例**

### **（1）牌型判断测试**

| **输入卡牌** | **预期牌型** | **测试代码片段** |
| --- | --- | --- |
| 黑桃 3、红桃 3、梅花 3、方块 3 | 炸弹（Bomb） | judgeCardType(cards) 应返回 Bomb |
| 大王、小王 | 王炸（Rocket） | judgeCardType(cards) 应返回 Rocket |
| 黑桃 5、黑桃 6、黑桃 7 | 单牌（Single） | judgeCardType(cards) 应返回 Single（每张单独判断） |

### **（2）AI 策略测试**

cpp

// 测试AI首轮出牌是否为最小单牌

AIPlayer ai;

ai.addCard(new Card(Card::Spade, Card::Five));

ai.addCard(new Card(Card::Heart, Card::Three));

QList<Card\*> firstPlay = ai.playCards(QList<Card\*>());ASSERT(firstPlay.first()->rank() == Card::Three); // 应返回3

### **（3）UI 交互测试**

* ****手动测试****：点击卡牌应改变选中状态（浅蓝色高亮），出牌按钮在非法牌型时禁用。
* ****边界测试****：选中 0 张 / 超过 12 张牌时，提示 “非法牌型”。

## **六、收获与反思**

### **1. 技术实现亮点**

* ****多态的灵活运用****：通过Player基类的playCards虚函数，轻松扩展玩家类型（如后续添加联网玩家）。
* ****自定义布局的挑战****：通过重写QLayout实现卡牌重叠，理解了 Qt 布局系统的尺寸计算逻辑（sizeHint、minimumSize）。
* ****牌型判断的边界处理****：顺子排除大小王、炸弹必须 4 张同点，通过枚举和算法确保逻辑严谨。

### **2. 代码优化空间**

* ****AI 策略扩展****：当前 AI 仅支持单牌、对子、炸弹，可添加连对、三带一 / 二、顺子等复杂牌型的判断与出牌逻辑。
* ****内存管理****：使用 Qt 的QList<Card\*>存储手牌，需确保delete已出牌的卡牌（当前通过Player::removeCard实现）。
* ****国际化支持****：卡牌花色点数显示为英文，可添加中文翻译（如 “黑桃 3”），通过QString本地化处理。

### **3. 调试经验**

* ****信号槽连接错误****：曾因CardWidget::clicked信号参数不匹配导致交互失效，通过 Qt Creator 的信号槽编辑器快速定位。
* ****牌型比较失败****：发现CardTypeJudger::isBetter中顺子比较未排序手牌，导致首张点数判断错误，添加std::sort后修复。

## **七、总结**

本次实验通过 Qt 框架实现了斗地主游戏的核心功能，从面向对象设计到 UI 交互，再到复杂游戏逻辑的实现，完整实践了软件开发的全流程。通过自定义布局和多态策略，代码具备良好的扩展性；牌型判断和 AI 策略的实现，提升了对算法边界条件的处理能力。未来可进一步优化用户体验（如卡牌图片替换文字）、完善 AI 智能程度，或添加网络对战功能，将单机游戏扩展为联机模式。实验过程中，Qt 的模块化设计和 C++ 的多态特性显著提高了开发效率，为后续 GUI 项目积累了宝贵经验。