技术设计文档

一、概述

本程序是一个卡牌匹配游戏，采用Cocos2d-x游戏引擎开发。游戏的核心玩法是玩家通过点击手牌区或桌面牌区的卡牌，将其移动到顶部牌区，匹配规则是点数差为1。游戏支持撤销操作，允许玩家回退上一步操作。

二、代码结构

2.1 核心模块

```
├── controllers/          # 控制器层
│   ├── GameController.h/.cpp    # 游戏主控制器
│
├── models/               # 模型层
│   ├── CardModel.h              # 卡牌数据模型
│   ├── GameModel.h              # 游戏状态模型
│   ├── UndoRecord.h             # 撤销记录模型
│
├── views/                # 视图层
│   ├── CardView.h/.cpp          # 单卡牌视图
│   ├── GameView.h/.cpp          # 游戏主视图
│
├── services/             # 服务层
│   ├── GameService.h            # 游戏规则服务
│   ├── GameConfigService.h      # 游戏配置服务
│
├── configs/              # 配置层
```

```
│   ├── models/
│   │   ├── LevelConfig.h       # 关卡配置
│   │   ├── CardResConfig.h     # 卡牌资源配置
│   ├── loaders/
│   │   ├── LevelConfigLoader.h # 关卡加载器
│
├── managers/             # 管理器
│   ├── UndoManager.h           # 撤销管理器
│
├── utils/                # 工具类
│   ├── JsonParser.h            # JSON解析器
```

2.2 关键类说明

类名	职责	主要方法
GameController	游戏主控制器	`handleCardClick()`, `handleUndoClick()`
GameModel	游戏状态模型	`getTopStackCard()`, `moveCardToPlayfield()`
CardModel	卡牌数据模型	`getValue()`
GameView	游戏主视图	`playCardMoveAnimation()`, `playUndoAnimation()`
CardView	单卡牌视图	`createCardFace()`, `moveToPosition()`
UndoManager	撤销管理器	`recordAction()`, `popUndoRecord()`
GameService	游戏规则服务	`isMatch()`

三、核心功能实现

3.1 卡牌匹配逻辑

```
bool GameService::isMatch(const CardModel& tableCard,
const CardModel& topCard) {
    int diff = abs(tableCard.getValue() -
topCard.getValue());
    return diff == 1;
}
```

3.2 撤销功能实现

```
void GameController::handleUndoClick() {
    if (!_undoManager->canUndo()) return;

    UndoRecord record = _undoManager-
>popUndoRecord();

    // 恢复卡牌状态
    switch (record.actionType) {
        case UndoActionType::HAND_CARD_CLICK:
            // 恢复到手牌区
            break;
        case UndoActionType::PLAYFIELD_CARD_CLICK:
            // 恢复到桌面牌区
            break;
    }

    // 播放撤销动画
    _gameView->playUndoAnimation(...);
}
```

四、 扩展新卡牌类型

4.1 步骤说明

4.1.1 **扩展卡牌枚举类型**

可以在LevelConfig.h中添加新的花色和点数：

```
enum CardSuitType {
    // ... 原有花色
    CST_JOKER,  // 新增花色：王
};

enum CardFaceType {
    // ... 原有点数
    CFT_JOKER, // 新增点数：王
};
```

4.1.2 **添加卡牌资源**

可以在CardResConfig中添加新卡牌的资源配置：

```
void CardResConfig::addJokerResources() {
    CardResItem jokerItem;
    jokerItem.imagePath = "res/cards/joker.png";

    // 添加大王和小王
    addCardRes(CST_JOKER, CFT_JOKER, jokerItem);
    addCardRes(CST_JOKER, static_cast<CardFaceType>
(CFT_JOKER+1), jokerItem);
}
```

4.1.3 **更新卡牌视图渲染**

在CardView::createCardFace中添加新卡牌的渲染逻辑：

```
void CardView::createCardFace(...) {
    if (suit == CST_JOKER) {
        // 特殊处理王牌的渲染
        std::string texture = (face == CFT_JOKER) ?
            "joker_red.png" : "joker_black.png";
        auto jokerSprite = Sprite::create(texture);
        // ... 添加精灵到卡牌
        return;
    }
    // ... 原有渲染逻辑
}
```

4.1.4 **更新匹配规则**

可以在GameService中添加新卡牌的匹配规则：

```
bool GameService::isMatch(...) {
    // 王可以匹配任何卡牌
    if (tableCard.suit == CST_JOKER || topCard.suit
== CST_JOKER) {
        return true;
    }
    // ... 原有匹配规则
}
```

五、扩展新类型回退功能

5.1 步骤说明

5.1.1 **定义新的回退操作类型**

在UndoRecord.h中添加新的操作类型：

```
enum class UndoActionType {
    // ... 原有类型
```

```
    MULTI_CARD_MOVE   // 新增：多卡牌移动操作
};
```

5.1.2 扩展撤销记录结构

修改UndoRecord结构以支持新操作：

```
struct UndoRecord {
    UndoActionType actionType;

    // 单卡牌操作字段
    int movedCardId;
    cocos2d::Vec2 originalPosition;
    int replacedCardId;

    // 多卡牌操作字段
    std::vector<int> movedCardIds;
    std::vector<cocos2d::Vec2> originalPositions;
};
```

5.1.3 实现新的回退逻辑

在GameController中添加对新操作的处理：

```
void GameController::handleUndoClick() {
    switch (record.actionType) {
        case UndoActionType::MULTI_CARD_MOVE:
            for (int i = 0; i <
record.movedCardIds.size(); i++) {
                int cardId = record.movedCardIds[i];
                Vec2 originalPos =
record.originalPositions[i];
                // 恢复卡牌位置
            }
            break;
```

```
        // ... 其他类型
    }
}
```

5.1.4 记录新的回退操作

在需要支持多卡牌回退的地方添加记录：

```
void GameController::handleSpecialMove() {
    // 执行多卡牌移动...

    // 创建撤销记录
    UndoRecord record;
    record.actionType =
UndoActionType::MULTI_CARD_MOVE;
    record.movedCardIds = {cardId1, cardId2};
    record.originalPositions = {pos1, pos2};
    record.replacedCardId = currentTopCard->id;

    _undoManager->recordAction(record);
}
```

5.1.5 更新撤销动画

在GameView中添加多卡牌撤销动画支持：

```
void GameView::playMultiUndoAnimation(const
vector<int>& cardIds,
                                      const
vector<Vec2>& positions,
                                      const
function<void()>& callback) {
    int completed = 0;
    for (int i = 0; i < cardIds.size(); i++) {
        auto cardView = getCardViewById(cardIds[i]);
```

```
            cardView->runAction(Sequence::create(
                MoveTo::create(0.4f, positions[i]),
                CallFunc::create([&] {
                    if (++completed == cardIds.size() &&
callback) callback();
                }),
                nullptr
            ));
        }
    }
```

六、 扩展性设计

6.1 卡牌系统扩展点

卡牌能力系统：

```
class CardModel {
public:
    // 添加能力字段
    CardAbility ability;
};

class GameService {
public:
    static bool isMatch(..., CardAbility ability) {
        // 根据能力调整匹配规则
    }
};
```

6.2 回退系统扩展点

复合回退操作

```
struct CompositeUndoRecord {
    vector<UndoRecord> records;
};

class UndoManager {
public:
    void recordCompositeAction(const
CompositeUndoRecord& composite) {
        _undoStack.push(composite);
    }
};
```