

设计文档

1. 项目概述

本项目是基于 Cocos2d-x 3.17 引擎开发的卡牌匹配游戏。采用了 MVC（Model-View-Controller）架构设计，通过模块分离，降低了系统复杂度，提升了后期扩展和维护的便利性。

游戏的主要功能包括：

- 卡牌的生成、显示与交互
- 卡牌匹配检测
- 手牌区与桌面区的管理
- 回退（Undo）操作支持
- 关卡配置动态加载

2. Classes 结构

```
Classes/  
├── AppDelegate.cpp / AppDelegate.h // 应用程序入口  
├── controllers/ // 控制器层  
│   ├── GameController  
│   ├── PlayFieldController  
│   └── StackController  
├── managers/ // 管理器（资源与配置）  
│   ├── LevelConfigLoader  
│   └── UndoManager  
├── models/ // 模型层  
│   ├── CardModel  
│   └── GameModel  
└── views/ // 视图层  
    ├── CardView  
    ├── GameView  
    ├── PlayFieldView  
    └── StackView
```

3. 核心模块设计

3.1 Model（数据模型层）

- CardModel：描述单张卡牌的属性
- GameModel：描述游戏整体数据状态，包括场上卡牌、堆区卡牌等集合

3.2 View（界面显示层）

- **CardView**: 单张卡牌的图形表现与动画
- **PlayFieldView**: 桌面区卡牌的布局与管理
- **StackView**: 备用堆区卡牌的布局与管理
- **GameView**: 总体布局管理，承载 **PlayFieldView** 和 **StackView**

3.3 Controller（控制逻辑层）

- **GameController**: 游戏启动、关卡加载、游戏重置等
- **PlayFieldController**: 管理桌面区卡牌逻辑
- **StackController**: 管理备用堆区逻辑

3.4 Manager（资源与工具）

- **LevelConfigLoader**: 解析关卡配置文件，生成初始数据
- **UndoManager**: 记录玩家操作历史，支持一步回退

4. 主要类说明

GameController: 管理游戏生命周期、协调各子模块

PlayFieldController: 管理桌面卡牌交互、检测匹配

StackController: 管理堆牌区，发牌与回收

GameModel: 保存全局卡牌数据

CardModel: 保存单张卡牌的属性

GameView: 游戏整体场景，布局 **PlayField** 和 **Stack**

CardView: 单张卡牌的视觉表现及动画

UndoManager: 记录操作以支持回退功能

LevelConfigLoader: 读取并解析关卡配置文件

5. 事件流与交互流程

1. 启动阶段:

- **GameController::startGame(levelId)**
- 调用 **LevelConfigLoader** 加载关卡数据
- 创建并初始化 **GameModel**
- 初始化各子控制器（**PlayFieldController**、**StackController**、**UndoManager**）
- 创建 **GameView**，添加到场景

2. 玩家交互:

- 点击卡牌 → **CardView** 触发点击回调
- 通知对应 **Controller**（如 **PlayFieldController**）处理逻辑
- 更新 **Model**
- **View** 观察 **Model** 变化并刷新界面

3. 回退操作:

- 点击回退按钮 → 通知 UndoManager
- 从 Undo 栈取出最近一次操作
- 逆向修改 Model 和 View 状态

6. 扩展与维护建议

- 新增关卡：只需在配置文件中增加描述，通过 LevelConfigLoader 自动加载。
- 新增卡牌类型：扩展 CardModel 类型枚举，同时增加对应资源。

7. 资源文件说明

项目的资源统一放置在 resources/ 文件夹下，包含以下内容：

```
resources/
├── card_bg.png      // 通用卡牌背面图片
├── configs/         // 配置文件目录
│   └── level_config.json // 关卡配置文件
├── numbers/         // 数字卡牌图像
├── suits/           // 花色图像
│   ├── heart.png
│   ├── spade.png
│   ├── diamond.png
│   └── club.png
```

8. 资源使用说明

- card_bg.png
 - 用作所有卡牌的背面或者花色数字背景图
- configs/level_config.json
 - 存储各关卡的卡牌布局、数量、花色等信息。
 - 游戏开始时由 LevelConfigLoader 加载，生成对应的 GameModel。
- numbers/
 - 包含大小两种数字图片，在 CardView 初始化时添加到 card_bg.png 组成卡牌
- suits/
 - 花色，包括红心（heart）、黑桃（spade）、方块（diamond）、梅花（club）。
 - 在生成卡牌时根据 CardModel 的花色属性动态绑定。

9. 关卡配置文件示例

level_config.json 示例结构:

```
{
  "Playfield": [
    {
      "CardFace": 12,
      "CardSuit": 0,
      "Position": {"x": 250, "y": 1000}
    },
    {
      "CardFace": 2,
      "CardSuit": 0,
      "Position": {"x": 300, "y": 800}
    },
    {
      "CardFace": 2,
      "CardSuit": 1,
      "Position": {"x": 350, "y": 600}
    },
    {
      "CardFace": 2,
      "CardSuit": 0,
      "Position": {"x": 850, "y": 1000}
    },
    {
      "CardFace": 2,
      "CardSuit": 0,
      "Position": {"x": 800, "y": 800}
    }
  ]
}
```

```

    },
    {
      "CardFace": 1,
      "CardSuit": 3,
      "Position": {"x": 750, "y": 600}
    }
  ],
  "Stack": [
    {
      "CardFace": 2,
      "CardSuit": 0,
      "Position": {"x": 0, "y": 0}
    },
    {
      "CardFace": 0,
      "CardSuit": 2,
      "Position": {"x": 0, "y": 0}
    },
    {
      "CardFace": 3,
      "CardSuit": 0,
      "Position": {"x": 0, "y": 0}
    }
  ]
}

```

说明：

- **playfield**: 初始在桌面区显示的卡牌列表。
- **stack**: 初始在备用堆区的卡牌列表。
- 每张卡牌由数值 (number) 和花色 (suit) 以及位置 (position) 定义。

10. 后续代码优化方向

1. 控制器（Controller）层限制对 View 和 Model 内部细节的直接访问。
2. 新增 ResourceManager 类统一管理资源路径。
3. 引入 AnimationHelper，统一管理卡牌翻转、移动等动画效果。
4. 支持本地存档功能，保存玩家进度。