|  |  |
| --- | --- |
| 基于  C  o  c  o  s  2  D  -  x  的卡牌类游戏框架的设计与实现  **张岩林**  吉林大学 | **分 类 号：TP311 单位代码：1 0 1 8 3**  **研究生学号：2014544088 密 级：公 开**    吉 林 大 学  硕士学位论文  **基于cocos2d-x的卡牌类游戏框架的设计与实现**  **Design and Implementation of Card Game Framework Based on Cocos2d - x**  **作者姓名：张岩林**  **专 业：软件工程**  **研究方向：图形图像与虚拟现实**  **指导教师：李文辉 教授**  **培养单位：软件学院**  **2017年4月** |

**基于cocos2d-x的卡牌类游戏框架的设计与实现**

**Design and Implementation of Card Game Framework Based on Cocos2d - x**

作者姓名：张岩林

专业名称：软件工程

指导教师：李文辉 教授

学位类别：工学硕士

答辩日期：2017年 月 日

未经本论文作者的书面授权，依法收存和保管本论文书面版本、电子版本的任何单位和个人，均不得对本论文的全部或部分内容进行任何形式的复制、修改、发行、出租、改编等有碍作者著作权的商业性使用（但纯学术性使用不在此限）。否则，应承担侵权的法律责任。

吉林大学硕士学位论文原创性声明

本人郑重声明：所呈交的硕士学位论文，是本人在指导教师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：2017 年 月 日

《中国优秀博硕士学位论文全文数据库》投稿声明

研究生院：

本人同意《中国优秀博硕士学位论文全文数据库》出版章程的内容，愿意将本人的学位论文委托研究生院向中国学术期刊（光盘版）电子杂志社的《中国优秀博硕士学位论文全文数据库》投稿，希望《中国优秀博硕士学位论文全文数据库》给予出版，并同意在《中国博硕士学位论文评价数据库》和CNKI系列数据库中使用，同意按章程规定享受相关权益。   
论文级别：□硕士 □博士   
学科专业： 软件工程  
论文题目： 基于cocos2d-x的卡牌类游戏框架的设计与实现  
作者签名： 　　　　　　　　　　指导教师签名：   
  
　　　　　　　　　　　　　　　　　　　 2017年 月 日

作者联系地址（邮编）： 吉林大学软件学院130012   
作者联系电话：14743102929

**摘 要**

**基于cocos2d-x的卡牌类游戏框架的设计与实现**

随着时代节奏变得越来越快，在人们追求更快捷便利的服务形式的大趋势下，移动游戏广泛的出现在了人们的生活中。其中卡牌类游戏已经成为国内手机游戏市场的主流，在苹果App Store中国区畅销榜Top50游戏中，卡牌类占到了13款。卡牌类游戏的开发一直以短周期、低成本、高风险、高利润为标杆，这样的特征吸引了众多冒险者前来开拓这片崭新的领域。

本论文根据当今智能手机游戏市场现状，以及所要开发的卡牌类手游特点，首先对主流开源手机游戏引擎进行总结，但作者发现主流游戏引擎并不能很好的满足作者的需求。所以为了贴合所开发的2D卡牌类游戏，作者对Coscos2D-x引擎及相关技术进行了详细的研究，并且在该引擎基础，改写了引擎的渲染模块，实现了游戏线程和渲染线程的分离。重写了场景控制模块，利用双层状态机的思想去管理UI模块的加载和释放。并且在此基础上完成了卡牌类手游的框架，该框架包含了一个完整的卡牌类游戏所必须的模块。该游戏框架相比原引擎，无论在游戏CPU利用率以及功耗比上，都有着明显的优化。

本文使用这个框架，利用国外著名页游的游戏UI，重新制作成手机端卡牌游戏。本文最后对游戏整体架构及重要模块进行总结，并对在制作过程中遇到的问题以及针对问题的解决方案进行了说明。

**关键词：**

cocos2d-x，图形学，openGL ES，多线程编程，手游框架

**Abstract**

**Design and Implementation of Card Game Framework Based on Cocos2d - x**

With the rhythm of the life getting faster and faster, under the general trend of people in the pursuit of more convenient and convenient form of service,mobile games widely appear in people's life.Which card game has become the mainstream of the domestic mobile gaming market. In the Apple App Store China Top sellout Top50 game,Card game occupy 13. The development of card games has been a short development cycle, low cost, high risk and high profit as a benchmark. this feature attracts many adventurers to try to open up this area.

In this paper, based on the current situation of smart phone game market, as well as the characteristics of the card to be developed. Summarize the characteristics of the mainstream engine. But the mainstream game engine is not very good to meet the needs of the author, so in order to fit the characteristics of the 2D card game, the author further study the knowledge of Coscos2d-x engine and its related technology.Especially the engine foundation, The engine's rendering module. To achieve the separation of game threads and rendering threads,athor Rewrite the scene control module and the rendering module.Author also build a two-tier state machine to manage UI loading and release. And on this basis,athor complete the card game framework.The framework contains the function that a complete card game must has.Compared to the original engine, both in the game smooth level and CPU share,the new framework have a significant optimization.

The author uses this framework and famous game’s UI,rebuild into a card games in mobile.Then author describes and summaries the various modules of the game and the problems encountered in The process of development.

**Keywords:**

cocos2D-x,Computer graphic,openGL ES,multithread programming, Mobile game framework

**目 录**

[第1章 绪 论 1](#_Toc231)

[1.1 选题背景及意义 1](#_Toc13494)

[1.2 本课题的发展及研究现状 2](#_Toc30678)

[1.2.1 智能手机发展现状 2](#_Toc18492)

[1.2.2 卡牌类游戏发展及研究现状 3](#_Toc10605)

[1.3 本文组织结构 4](#_Toc28574)

[第2章 相关技术概述 6](#_Toc31980)

[2.1 主流手游引擎简介和比较 6](#_Toc7635)

[2.2 Coscos2d-x游戏引擎 6](#_Toc26456)

[2.2.1 Coscos2d-x游戏引擎介绍 6](#_Toc12810)

[2.2.2 Coscos2d-x的结构 7](#_Toc6242)

[2.2.3 Cocos2d-x工作流 8](#_Toc7555)

[2.2.4 Cocos2d-x的渲染模块 9](#_Toc845)

[2.2.5 Cocos2d-x的场景更新模块 10](#_Toc22873)

[2.2.6 Cocos2d-x的事件管理模块 10](#_Toc19499)

[2.3 寻路算法 11](#_Toc13126)

[2.4 静态数据与动态数据存储 12](#_Toc9200)

[第3章 引擎的优化与实现 15](#_Toc29168)

[3.1 相关需求分析和优化目标 15](#_Toc12840)

[3.2 逻辑线程和渲染线程的分离 15](#_Toc5188)

[3.2.1 游戏循环 16](#_Toc24918)

[3.2.2 现有引擎的多线程渲染 17](#_Toc19678)

[3.2.3 多线程渲染模块 20](#_Toc19264)

[3.3 场景更新模块 21](#_Toc8601)

[3.3.1 分层状态机介绍 23](#_Toc11452)

[3.3.2 新版场景更新模块框架 24](#_Toc6332)

[3.4 事件管理模块 25](#_Toc17587)

[第4章 游戏设计与实现 26](#_Toc26712)

[4.1 声音模块 27](#_Toc24697)

[4.1.1 需求分析 28](#_Toc471)

[4.1.2 功能构建 28](#_Toc10155)

[4.2 时间管理模块 29](#_Toc25296)

[4.2.1 需求分析 29](#_Toc3121)

[4.2.2 功能构建 29](#_Toc7570)

[4.3 菜单模块 29](#_Toc10005)

[4.3.1 需求分析 30](#_Toc24334)

[4.3.2 功能构建 31](#_Toc20812)

[4.4 主面板模块 31](#_Toc9477)

[4.4.1 需求分析 33](#_Toc32686)

[4.4.2 功能构建 33](#_Toc11939)

[4.5 补给面板模块 34](#_Toc25510)

[4.5.1 需求分析 35](#_Toc17130)

[4.5.2 功能构建 35](#_Toc7777)

[4.6 编队面板模块 35](#_Toc935)

[4.6.1 需求分析 36](#_Toc648)

[4.6.2 功能构建 37](#_Toc30906)

[4.7 建造面板模块 37](#_Toc208)

[4.7.1 需求分析 37](#_Toc7933)

[4.7.2 功能构建 38](#_Toc32216)

[4.8 修理面板模块 39](#_Toc24339)

[4.8.1 需求分析 39](#_Toc1073)

[4.8.2 功能构建 40](#_Toc13273)

[4.9 面板跳转实体 40](#_Toc7052)

[4.9.1 需求分析 41](#_Toc17757)

[4.9.2 功能构建 41](#_Toc5723)

[4.10 大地图场景 41](#_Toc9344)

[4.10.1 需求分析 42](#_Toc29028)

[4.10.2 功能构建 43](#_Toc29869)

[4.11 战斗场景 44](#_Toc32032)

[4.11.1 需求分析 44](#_Toc31823)

[4.11.2 功能构建 45](#_Toc1753)

[第5章 资源优化 48](#_Toc6633)

[5.1 纹理优化 48](#_Toc10206)

[5.1.1 压缩纹理的特点 48](#_Toc19199)

[5.1.2 在cocos2d-x中使用压缩纹理 48](#_Toc21329)

[5.2 加载纹理数据 49](#_Toc6958)

[第6章 游戏测试 51](#_Toc16209)

[6.1 改良后的引擎测试 51](#_Toc17703)

[6.2 游戏单元测试 52](#_Toc27796)

[6.3 游戏性能测试 52](#_Toc23662)

[结 论 53](#_Toc26539)

[参考文献 54](#_Toc19970)

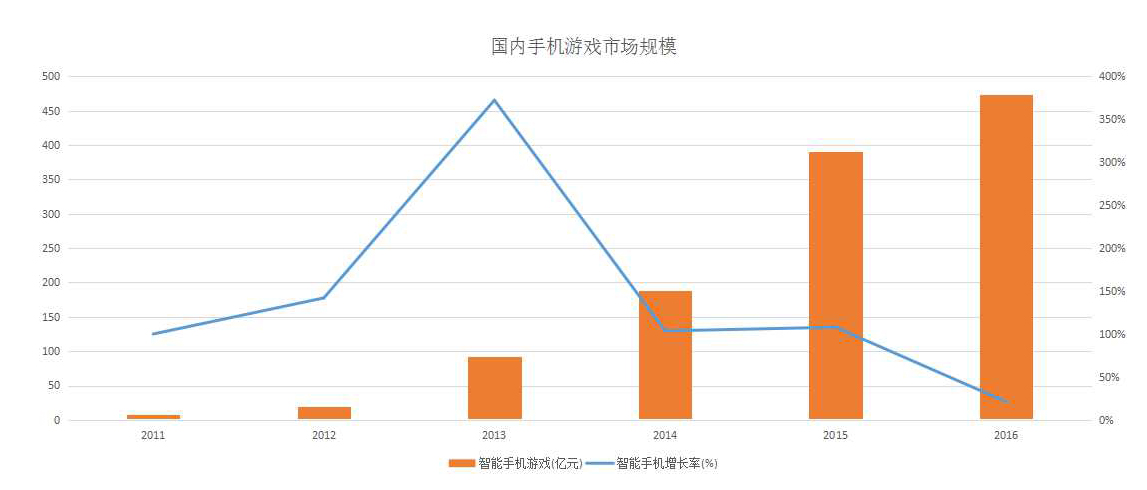
[作者简介 56](#_Toc31107)

[致 谢 57](#_Toc14174)

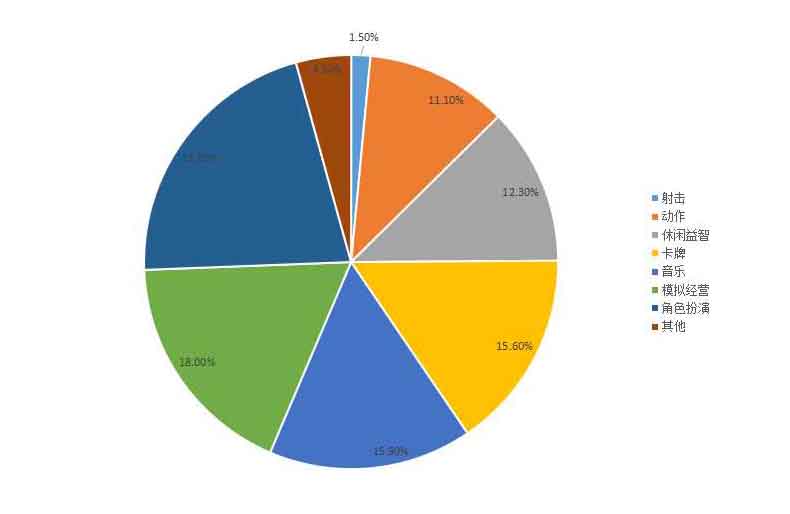
**第1章 绪 论**

**1.1 选题背景及意义**

近些年，随着智能终端的普及，以及硬件水平的提升。手机游戏得到了空前的发展，我们手机里边安装的游戏，远远不是印象中类似俄罗斯方块，踩地雷，连连看之类画面简陋，规则简单的游戏。现在的手机游戏拥有可以和客户端游戏相媲美的画质及交互性能，并且手机游戏具备客户端游戏所不具备的便携性和移动性，使得玩家可以随时随地的掏出手机，方便快捷的使用app store或者google play安装所需要软件，使用简单的支付方式购买游戏道具。手机游戏的出现给人们带来一种集沟通，娱乐为一体的全新的生活方式。在满足人们娱乐方面的方面，手机游戏发挥举足轻重的作用。

图1.1 中国历年手游市场规模以及增长率

2016年全球手机游戏市场的收入规模已经超过300亿美元。如图1-1所示，2016年中国国内手游全年总收入达到450亿人民币，中国手游市场增速放缓但仍然保持增长[1]。同时随着手机硬件水平的进一步提升，手游的总体趋势向重度化，以及端游化发展。大量的RPG类游戏上线，其中不乏诛仙，剑侠情缘，倩女幽魂等一系列原本在端游时代就重量级的RPG游戏IP。图1-2总结了2016年国内市场所有上线游戏游戏类型所占市场比例，综合2016年来看，我们很难说哪一种玩法潮流是居于统治地位，每一类手游在突破过去的旧模式，每一个突破都在今年掀起波澜。比如说有人预言卡牌类游戏在逐渐的消亡[2]，但不得不说2016最赚钱的游戏应属于阴阳师。阴阳师属于典型的日式卡牌类游戏，但巧妙是在游戏前期，制作团队避开了粗暴的卡牌强度和战斗力数值问题，将玩家的目光定格在游戏的阵容搭配，这种横向技巧的探索上，产生了一种“我有智商靠搭配，血统爆发胜土豪”的错觉。同时在同类的游戏中，画风，操作细节，声优，抽卡粘性上属于上乘，中日文化同根同源，日本的神道教传说大量参考了我国的山海经，所以对于和风文化，中国人也不会感到陌生。综上所述一系列优点，成为了阴阳师长期保持热度的基石。

图1.2 2016年国内市场所有上线游戏游戏类型所占市场比例

随着中国社会的发展，人们的消费水平逐渐提高，我们越来越注重日常中精神世界的消费。游戏的开发向科学化，规范化，规模化的方向发。

**1.2 本课题的****发展及研究现状**

**1.2.1 智能手机发展现状**

2016年全球智能手机出货量为13.6亿部，年成长为4.7%，三星占据世界第一出货量。苹果2016年出货仅为2.09亿部，成长率为-11.5%，其中国产品牌华为，OPPO，vivo排名仅次于三星，苹果，均进入了前五。华为手机成长最快，自华为MATE7系列取得成功后，P8,META8相继受到市场追捧，取得巨大的成功。凭借华为自身强大的研发能力以及持有的众多自主知识产权，相信国产手机在未来的发展会更进一步。目前中国手机用户已超13亿，智能手机市场趋向于饱和，新的手机用户增长越来越加缓慢，换机成为了市场主流。

在智能手机发展的过程中，同时带来了手机App以及手机游戏的快速发展。苹果的App Store，谷歌公司的Google Play市场，国内的百度，360市场应用分发数量快速增长，也从侧面带动了手机APP和手机游戏的快速增长。

苹果的开发审核制度较为严格，所以IOS的应用数量不如Google Play上多，但应用质量较高。国内未开通Google Play服务，安卓应用安装主要依靠各大互联网公司自己的应用商店，其中包括360软件管家，豌豆荚，应用宝等。

**1.2.2 卡牌类游戏发展及研究现状**

卡牌类游戏起源于桌游，比如说经典的万智牌，游戏王都属于卡牌类的一种。甚至扑克，麻将也有很多人把他们归类为卡牌类游戏。卡牌类手游在原有的卡牌类桌游的基础上，添加了卡牌类角色战斗的功能。相比桌游形式的卡牌类游戏，更容易携带，玩法更多样，画面更绚丽。

卡牌类手游兴起于2013年。2013年《[我叫MT](http://game.91.com/zq/mt/" \t "http://game.91.com/chanye/news/_blank)》横空出世，这款卡牌在当时凭借优秀成熟的玩法引爆了整个手游市场，成为了当时现象级的手游产品，同时代领了整个手游市场的发展。得力于《[我叫MT](http://game.91.com/zq/mt/" \t "http://game.91.com/chanye/news/_blank)》的巨大成功，无论是大公司还是小团队，都把手上的卡牌类游戏尽可能的砸向市场，市场对卡牌游戏的肯定达到了前所未有的高度。

卡牌类手机游戏现在依然是当今游戏市场上的绝对主角，2016年仍然有不少的卡牌手游赢得市场和玩家的喜好，其中包括《拳皇98终极之战OL》，《阴阳师》。尤其是《阴阳师》，《阴阳师》有着浓厚的日式风格，同时在核心玩法上融合了RPG+回合制的元素，走出了自己的风格。

**1.2.3 手机游戏引擎技术研究**

除了各个公司的自研引擎以外，市场上的手游引擎主要有Unity和Coscos2D-x[3]。相较于Coscos2D-x，Unity是一套完整的解决方案，相关工具齐全，并且提供了相对应的上层脚本语言，极大加快了开发速度。但是内部代码并非开源。所以对于Unity来说，如果需要在后期根据具体游戏对引擎内部进行优化，是不可能完成的任务。Coscos2D-x相较于Unity，只能说是一个基本的代码库，自身不含有编辑器，也没有形成一套完整的开发工具链。但是相较于Unity，Coscos2D-x的包体的更小，并且引擎是开源的，适合卡牌类这种需要较小包体，在渲染和功能上要求不高的游戏。在实际开发过程中，开发者可以在Coscos2D-x引擎的基础上，针对所开发游戏，对引擎内部代码进行适当的修改，并在其上封装一套自己的框架。

**1.3 本文组织结构**

本文是以Coscos2D-x为基础，在原有的引擎的基础上对引擎进行改造，并在其上实现了完整的卡牌类手游客户端框架。文章重点介绍了引擎的升级与改造，以及卡牌类游戏框架所设计重点场景和模块的构建。每章节安排如下:

第一章.引言部分，介绍了选题的背景和选题的意义，课题的发展和课题的研究现状，论文的组织结构。

第二章.技术论述，对文章所涉及到的技术做了介绍。包括主流游戏引擎，寻路算法，游戏相关数据存储。

第三章.引擎的优化与实现，讲述了作者对引擎的改进。包括相关的需求和优化的目标，逻辑线程和渲染线程，场景更新模块，事件管理模块。

第四章.游戏设计与实现，讲述了游戏中重要模块的构建。作者从需求和实现连个角度来分析。

第五章.资源优化，讲述了作者如何对游戏资源进行后期优化。包括纹理优化和加载纹理数据优化两个方面。

第六章.游戏测试，讲述了如何进行游戏测试。包括改良后的引擎测试，游戏单元测试以及游戏性能测试。

1. **引擎的优化与实现**

**2.1 相关需求分析和优化目标**

卡牌游戏是一种低强度的游戏类型。卡牌游戏的中心一般放在卡牌的收集和卡牌的升级上。尤其在手游端，卡牌类型游戏多以回合制为主，仅提供极其少量可供玩家操作的功能。针对这种游戏模式，游戏不需要有太高的帧率，一般每秒20帧左右足够。同时没有必要设计类似于动作类或射击类游戏复杂的同步机制。但在此同时，开发者要尽量提高手机硬件利用效率，并且降低电池的消耗量，可以在当前游戏模式的基础上，提供低电量功能，降低手机亮度，帧率，游戏特效到最低，以降低电池的消耗。所以我们对引擎的优化中心放在相关功能的扩展和电量的消耗上。

**2.2 Cocos2D-x游戏引擎**

**2.2.1 Coscos2d-x游戏引擎介绍**

Cocos2d-x是一个开源2D游戏引擎(也有3D版本)，在MIT许可证下发布的[4]。以Cocos2d-x为引擎的游戏具有开发快速，简易等优点。Cocos2d-x由cocos2d-iphone进化而来，作者重写了原引擎中的OC代码，把原先OC代码重构成C++代码。以cocos2d-x引擎为核心，使用C++代码作为平台代码，可以很容易的建立和运行在IOS,Android，黑莓Blackberry等操作系统之上。Cocos2d-x还支持Windows,Mac和Linux等桌面操作系统[5-7]，因此开发者编写的源代码可以很容易在桌面操作系统中编译和调试。为了加快开发速度，Cocos2d-x还提供了lua,JS,Html版本。

Cocos2d-x相较于其他引擎，最大的优点在于他的可靠及简单。在实际的C++使用中，动态内存分配是一把双刃剑，一方面动态内存提高了应用程序的性能及内存使用的灵活性，但由于程序没有正确地分配和释放会造成内存泄露，所以Cocos2d-x仿造了Object-C语言的引用计数式内存管理方式。实际使用时，只要继承了Ref类，该对象会自动被引擎管理，大大简化了C++的使用难度。

**2.2.2 Coscos2d-x的结构**

图2.1 Cocos2d-x引擎结构

如图2.1所示，Cocos2d-x是一个跨平台的2D游戏引擎，底层代码使用C++编写，通过C++代码调用平台代码，上层可以选用lua或者js[8]。支持主流的移动和PC平台，如IOS,Android，Win32等。Cocos2d-x使用了一系列第三方开源库，这些库支持了文件解压，图片解析，网络支持，物理引擎，音频等功能，从而使得Cocos2d-x能更加专注于整体架构和渲染部分开发。

在此基础上，Cocos2d-x封装了一些与平台无关的接口，如文件的加载，纹理的解析，线程操作等，这使渲染和其他系统的设计可以与平台无关。此外，Cocos2d-x封装了一些基础功能模块，这些功能可以供上层游戏系统使用，包括内存管理模型，纹理资源的加载和缓存，Cocos2d-x还提供了一组数据容器，使STL中的容器可以与Cocos2d-x的内存管理模型相融合。

渲染系统是Cocos2d-x的核心部分，它包括了场景中元素的层级结构，提供元素的管理，遍历，绘制等功能。这些元素包括Node,Sprite,Layer,TileMap,Particle等。这一层是与OpenGL ES交互部分。

Cocos2d-x有一套优秀的动画系统，它不仅支持一些第三方设计工具，使用起来非常方便。Cocos2d-x将一些非动画的特性与动画结合起来，大大简化了游戏的开发工作，甚至连苹果公司的Sprite Kit都借鉴了Cocos2d-x中动画系统的设计，这也是Cocos2d-x在2D领域非常出色的原因。

Cocos2d-x支持非常丰富的第三方工具，如TexturePacker，SpriteHelper,Tiled,zwoptex,ParticleDesigner,GlyphDesigner,CocosBuilder等。而Cocos2d-x可以使用可视化GUI进行开发，它继承了场景编辑，动画设计，粒子特效，骨骼等Cocos2d-x的大部分功能，还在朝着与Cocos2d-x深度集成的方向前进。

**2.2.3 Cocos2d-x工作流**

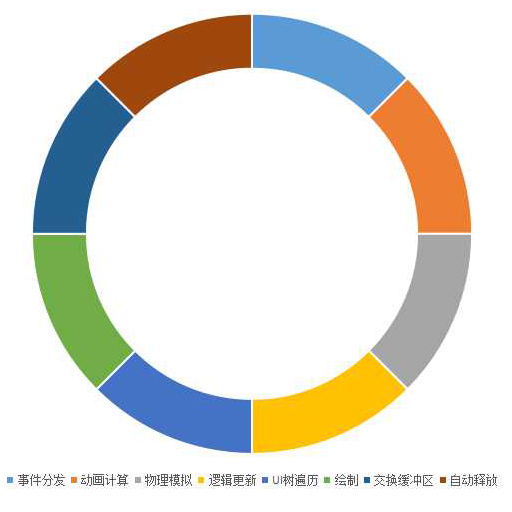
****

图2.2 Cocos2D-x引擎工作流

Cocos2d-x把渲染模块和gameplay分开，中间通过命令模式连接在一起[9]。分开处理游戏引擎最重要的两大部分，有利于我们后期整体做优化，减少每帧渲染时的DrawCall。

图2.2是cocos2D-x引擎的工作流。每一帧最开始，我们分发上一帧触发的所有事件。在一帧到下一帧的过程中，前一帧触发的事件先暂存在一个队列中，之后在这帧结束时统一分发。Cocos2d-x引擎把所有的Input全部封装成事件的形式，外加引擎不同模块之间的少量调用做成的自定义事件。使用EventDispatcher分发所有的事件。监听器功能全部封装在EventListener中，需要提前注册到逻辑代码里。第二步是逻辑更新，逻辑更新依据我们提前定义的更新顺序来制定执行的顺序，由Schedulue统一管理。Cocos2d-x把动画计算和物理引擎模拟全部放在逻辑更新模块中。只不过动画以及物理引擎被指定一个特殊的优先级，PRIORITY\_SYSTEM和PRIORITY\_SYSTEM+1.所以在实际使用时，保证动画和物理引擎模块的刷新会优先执行。第三步渲染部分，主要包含三个步骤:生成绘制命令，对绘制命令进行排序，执行绘制命令。渲染结束以后交换前后缓冲区，等待特定时间以后执行下一帧。

**2.3 渲染模块的优化**

多核CPU目前是大势所趋，尤其在嵌入式端，手机作为移动载体，散热量和功耗都受到严格的限制。手机端散热器设计，电池容量，CPU主频都有着极大的限制。硬件厂商只好降低CPU主频降低，增多手机CPU核心数量。近些年来，市面上4核心，8核心CPU的手机已经变成了标配，甚至一些硬件厂商推出了16核心的手机CPU。不可否认的是，多核CPU提高了手机的续航能力，降低了整体的功耗比。但是由于手机CPU核心降低，单核的运算效率下降。Ccocos2d-x引擎本身即为单线程游戏引擎，在游戏运行过程中，只有一个核心能够得到充分的利用，其他硬件资源无法得到合理的应用。常常出现游戏卡顿，但CPU整体占用率不高的问题。 所以我们把引擎游戏逻辑gameplay线程和渲染线程分离，把他俩放到不同的线程。

为何要把渲染部分放到一个单独的线程中去呢？游戏作为实时交互的可视化界面，为了保证游戏不会出现明显的卡顿，每秒需要刷新数十张画面，需要耗费大量的运算资源，其中渲染部分所占的运算资源比例最大。在渲染的过程中，GPU可以看成一个外设，渲染的过程就是CPU不停的给GPU发送各种命令[12]。系统给GPU分配固定的存储器作为显存，渲染过程中，通过总线把渲染资源从内存传送至显存中，之后使用GPU来调用显存进行渲染。当然不排除PS4这种通过huma统一寻址的硬件设备。在GPU渲染的过程中，CPU不断调用各种API，修改此时的渲染状态，当渲染状态调整到适应的模式。会有一个flush的过程，buffer中的数据夹带着命令压入GPU渲染管线进行渲染，每个渲染命令都需要从头到尾走一遍渲染管线，最后传入帧缓冲区或是渲染缓冲区中[13]。之后这个buffer会被清空，以便接受新的命令。GPU渲染管线对于我们游戏开发是黑盒的，不能修改其中的代码。但我们在渲染管线外边，通过合并相同类型渲染命令的方式，减少整体DrawCall。把渲染状态相同而顶点不同的DrawCall合并到同一个batch里。渲染过程中大量的花费调整渲染状态上，而且调整渲染状态需要调用渲染管线的API，容易导致CPU的阻塞，影响游戏逻辑更新的正常进行，造成明显的卡顿。所以在渲染过程中，可以利用把渲染部分划分到一个独立的线程中这种方式，把CPU等待GPU的时间充分利用起来，或者至少不会让这些等待耽误到游戏逻辑线程的运行。

多线程渲染是个典型的生产者与消费者模式，生产者游戏逻辑线程把渲染命令不断的加到一个队列中，渲染线程不停的从这个队列中读取命令并执行。通过合适的同步方式把两个线程连接。实现的过程中需要注意两点，游戏逻辑线程中的逻辑计算不影响渲染速度，较差的GPU渲染速度的低下不影响逻辑计算速度。只要逻辑计算不受到影响，即使游戏出现了少量的卡顿，也不会出现逻辑上的问题，并且可以针对卡牌类型游戏设计出相应的低电量模式。低电量模式即为，为了节省手机电量消耗的目的，保持游戏逻辑线程高速运转的同时，降低游戏渲染线程帧率。

**2.3.1 游戏循环时间同步方式**

不同于其他应用程序，游戏即使玩家停下输入也继续再跑，所以在游戏循环中，处理游戏输入，但不可持续等待游戏输入，游戏循环始终在一个while循环里运转。常见的游戏循环至少包含处理输入，更新，渲染等几个步骤。用帧率来衡量游戏循环的速度[14]。游戏循环的很快，帧率便高，游戏循环的很慢，帧率便低。一轮循环中处理信息量的大小以及底层平台的运算水平都会影响帧率的高低。早期游戏中，因为平台固定，只需要适当调整处理的信息量，我们可以轻松开发出恒定帧率的程序。但现在由于平台的复杂性，我们不能只是通过调整处理的信息量来达到最佳的方式，因此提出了以下几种方式。

1. 固定间隔，没有同步。

老式游戏中，没有时间同步方法，采用调整信息量的方式确定时间间隔，但如果面对不同平台，在较快的机器上会快的令玩家不知道游戏在做些什么，在慢的机器上一会变慢一会变快，所以这种方式基本被抛弃。

1. 固定间隔，有同步。

假如说你想在一秒钟处理60帧，并且保证平台一定能在规定时间内完成这个任务，甚至提前完成任务。我们可以算出处理一帧所花费的时间，如果提前处理完，则等待相应的差值时间，直到满足一帧所需要的时间。这种方式简单方便，并且sleep保证了即使过快的处理完一帧，游戏也不会运转的太快，但是这样会出现一个问题，如果无法再规定时间内完成任务，睡眠时间会变成负值，程序会出现错误。

3.动态间隔。

在每次游戏循环过程中，每次循环进行一次渲染操作，但把这次循环与上次循环的间隔时间传入逻辑操作，使用实际的时间去进行逻辑操作，接下来游戏引擎负责将游戏世界更新到这个时间增量的下一个状态。这种方法解决了循环花费时间过长的问题。这样以来在不同平台上物理更新不会出现问题，并且高端平台还可以得到更流畅的体验。但这种更改会让游戏变得不稳定而且不确定。每次update的delta可能会相差很多。所以在计算物理引擎的action时，不同客户端中update的delta不同导致float运算结果不同，可能会导致上层显示结果出现偏差。

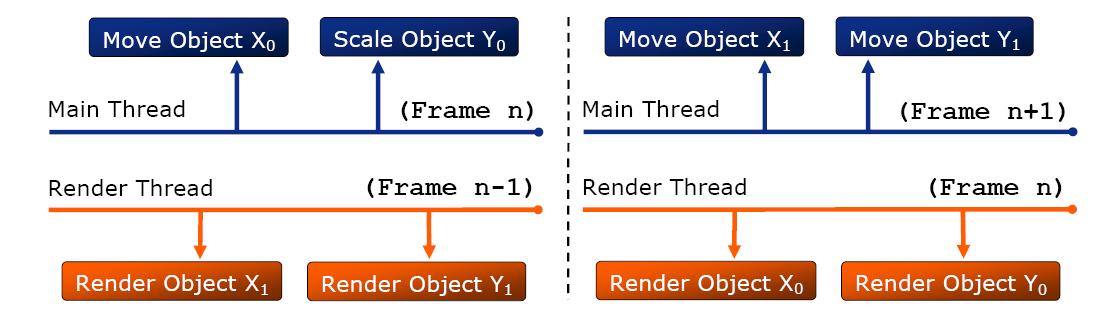
4.固定更新间隔，动态渲染间隔。

渲染部分是引擎中不会受变时迭代影响的部分。由于渲染引擎表现的是游戏时间中的一瞬间，所以它不会关心距离上次渲染过去了多长时间。所以我们可以把逻辑运算和渲染运算分开进行。逻辑运算使用固定的时间间隔。每次游戏循环都进行一定时间的累加，并且每次累加都进行渲染，当累加到一定程度满足逻辑运算的时间间隔时，运行一次逻辑运算。虽然是引擎循环仍然是单线程的，但是实现了逻辑部分的与渲染部分的分离，解决了逻辑部分小数点运算上潜在的问题。但实现过于复杂，需要考虑考虑高端机与低端机匹配的问题，高端机时间间隔尽量小，低端机单次update的时间不要超过总的间隔。

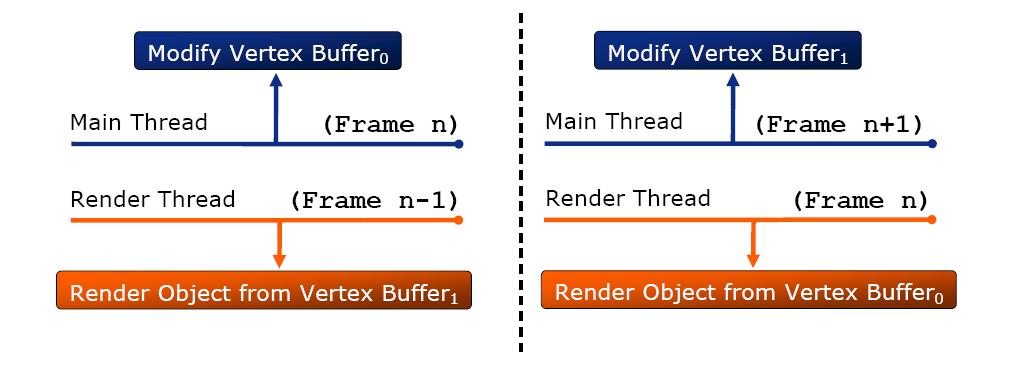
**2.3.2 几种多线程渲染方式**

1. OGRE

OGRE分为两种架构，一种为middle-level multithread，另一种为low-level multithread[15]。

图3.1 middle-level multithread

如图3.1所示，每个需渲染的实体被复制成了两份，主线程和渲染线程交替更新和渲染同一个实体的两个备份，并在一帧结束时同步，这种解决方案达到了第一个目标而并没有达到第二个目标，同时两份实体的维护也相对复杂，没法为更多核数的CPU进行扩展优化。

图3.2 Low-level multithread

如图3.2所示，将D3D对象复制两份，同样是在帧结束时同步并交换，和上面的优缺点类似。两种多线程Ogre的解决方案都是在引擎层完成的，对上层应用透明，对于用户而言无需考虑多线程细节，这点是非常不错的。

2.idsoftware

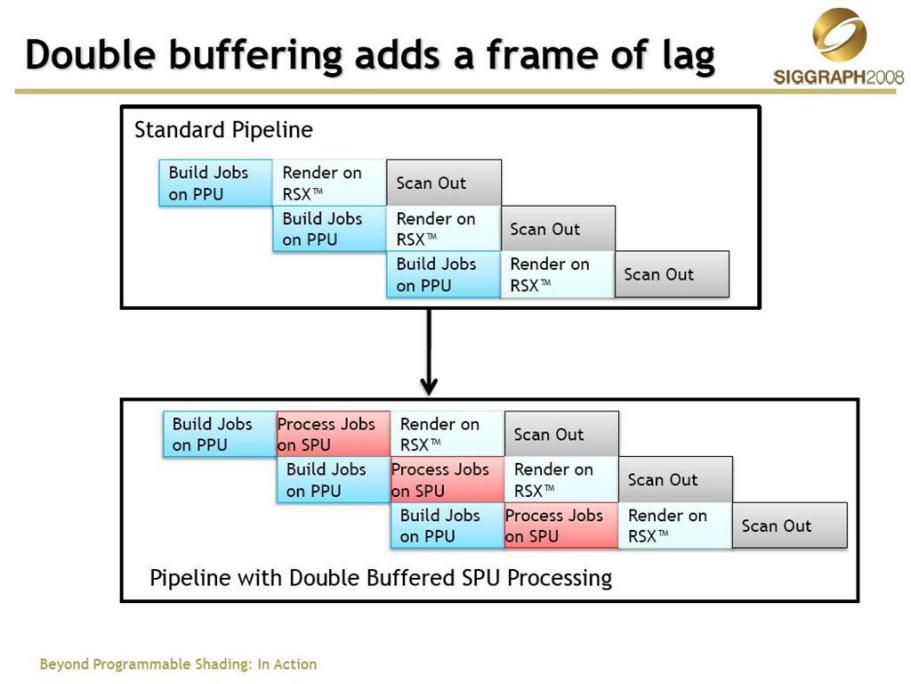


图3.3 PS4引擎架构

这里是已PS4的引擎结构为例的，与PC有较大的差别，其中SPU是Cell芯片的8个协处理器，拥有强大的并行能力，id的解决方案在SPU上进行了诸如骨骼动画、形变动画、顶点和索引缓存的压缩、Progressive Mesh的计算等诸多内容，同时与PPU上的物理计算RSX上的渲染工作交错进行，最大化的利用了PS4的硬件结构。

3.UE4

UE4无论是在编译器模式还是游戏中，都是默认开启多线程渲染的，通过命令行-onethread来控制多线程渲染的关闭[16]。UE4使用命令缓冲来实现多线程渲染，UE4把每一个命令封装到RenderCommand派生类来实现渲染命令，通过一些列的宏，可以很方便的追加新的地方而不需要修改其他地方。使得每一个Command都能生成唯一对应的派生类。虽然在有些IDE上宏命令的调试并不是很方便，并且宏命令的代码难以理解，但是我们可以很轻松的在游戏逻辑线程中找到主线程中调用这个指令的相应位置[17-18]，可以减少渲染相关操作IDE调试的难度。

在实际游戏中，游戏逻辑线程负载通常比渲染线程低很多，所以应设计适当的同步方式，使得游戏逻辑线程在跑的过快的情况下，停下来等待渲染线程执行。UE4提供了一个叫做RenderCommandFence的类，保证游戏逻辑线程不会跑的太快，最多超过渲染线程一帧，好比我们在前台看到的画面N帧，渲染线程正在渲染第N+1帧，游戏逻辑线程最多处理N+2帧数。

游戏过程中，如使用同一份资源，可能会出现处理到第一帧的游戏逻辑线程修改了渲染到第二帧的资源，造成资源错误的情况[19]。所以为了降低引擎中资源管理的复杂度，在UE4引擎中，渲染线程和游戏逻辑线程中采用分离资源，在游戏逻辑线程里去处理一个游戏对象会在渲染线程里边生成一个对应的Proxy对象。该Proxy对象的状态数据在游戏的每一帧更新都在变，保持与游戏逻辑线程中资源的相对同步。

**2.3.3 Cocos2D-x的渲染模块**

Cocos2d-x支持在屏幕上绘制精灵，文本，形状，粒子，地图等。所有的这些元素都继承于Node类。通过UI渲染树把他们联系在一起。

一个场景中的元素天然的适合用树来表示，树的根节点为Scene类，UI树中的每个节点都是一个Node的实例对象。每个Node节点具有一个children集合及一个parent节点。其中Scene的parent节点为空。通过这样构建UI树，我们可以很容易的添加，删除，修改场景中的元素。并且方便游戏模块和渲染模块的分离。渲染分为三个步骤，生成绘制命令，对绘制命令进行排序，不同Node节点绘制的先后顺序通过逻辑深度localZOrder来指定，执行绘制命令。采用中序的的深度优先算法进行遍历。同时在此过程中生成MV矩阵。遍历过得节点根据具体的节点类型，把他们基本属性以及对应的MV矩阵封装到一个RenderCommand中，RenderCommand作为一个绘制命令，它定义了如何绘制一个UI元素。在执行绘制命令之前，引擎需对栈上的绘制命令进行排序。在执行绘制命令的过程中，相同纹理，相同着色器，同混合方式的绘制命令在渲染过程中可以合并。合并之后的命令统称为一个Batch。

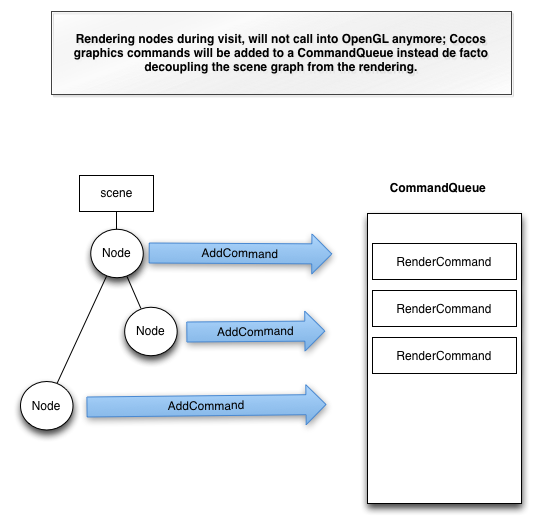


图2.3 Cocos2d-x的UI树遍历

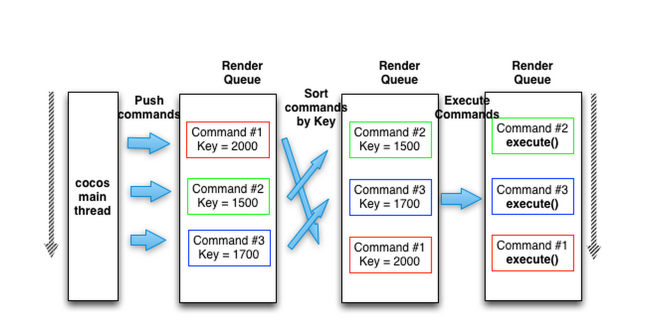
Coscos2d-x 3.0版本对渲染模块进行了大量更新。相比2.0版本，更新之后的引擎实现了渲染部分和游戏部分的分离，对引擎进行集中优化更方便。在2.0版本引擎中，如果场景需要大量渲染同一纹理图片，必须使用Batch。但更新之后的渲染模块，引擎在渲染前会自动对渲染队列排序，借此实现了自动Batch[10]。这么做加快了渲染效率，同时降低了使用难度。但受限于引擎框架，Cocos2d-x的工作流仍然是单线程。

**2.3.4 改良后渲染模块**

渲染系统管理下的每一次绘制命令都继承于RenderCommand,我们提供QuadCommand,GroupCommand等子类来代表不同的渲染命令。QuadCommand用于绘制一个矩形区域，在2D游戏中，大部分的游戏组件都是由矩形贴图构成，每个矩形都是纹理的一部分。

一个矩形有4个顶点，每个顶点由结构体V3F\_C4B\_T2F指代，顶点坐标占3个float，颜色参数占4个float，纹理坐标占2个float。同时还包含所映射纹理的纹理编号，着色器程序，以及混合模式。相同纹理编号，着色器程序，混合模式的QuadCommand，可以在渲染线程中Batch。GroupCommand通常不具有具体的渲染命令，他只是指向一个RenderQueue，当渲染线程绘制一个GroupCommand他能找到相应的RenderQueue，然后执行其中的RenderCommand。我们把一组RenderCommand放入GroupCommand就可以实现特殊绘制，比如深度模板本质上就是一个GroupCommand。

场景中的每一个UI元素的绘制命令RenderCommand将被发发送到一个叫作RenderQueue的绘制命令栈上，RenderQueue存储着几组RenderCommand。一个RenderQueue对应一帧所有的渲染命令，在帧切换的过程中，UI元素会发生改变。所以RenderCommand不会以指针的方式存储数据。渲染线程再绘制命令之前，会对RenderQueue中的绘制命令进行排序，然后按照新的顺序执行它们。所以绘制命令中被执行的顺序不一定是UI元素被遍历的顺序。2D渲染器绘制方法多基于画家算法，所以绘制的时候，从队列中第一个命令，直到完成一个RenderQueue中所有命令的绘制。

 图3.4 渲染队列根据渲染命令自身所生成的Key排序

游戏线程负责游戏中的逻辑循环，事件分发，UI树遍历，渲染命令的压栈。渲染线程负责渲染命令的排序，渲染管线的状态更新，调用GPU相关API和完成渲染。对于多线程渲染模块，游戏线程可以领先渲染线程数帧，所以在引擎中，我们可以保存多个RenderQueue，游戏线程每循环一次，多一个RenderQueue。渲染线程每循环一次，减少一个RenderQueue。游戏线程循环速度比渲染线程快的多，所以在游戏线程的开始我们使用RenderFence来给游戏线程加锁。通过判断RenderQueue的数量来衡量游戏线程领先渲染线程多少帧。我们一般设定为最多2帧领先。渲染线程以回调函数的方式运行，在渲染线程开始渲染之前，我们需要判断RenderQueue中是否为空，并且当前RenderQueue是否还处于游戏线程状态，在RenderQueue队列为空，或当前RenderQueue处于游戏线程压栈过程中。需要wait直到当前游戏线程压栈完成。

游戏线程以及渲染线程中，需要制定时间同步规则。我们采用动态间隔的方法进行时间同步，因为卡牌类型游戏不涉及零误差的困扰，在动作状态机上可以允许一定的误差，并且单机游戏不涉及多个客户端，而固定间隔update会增加游戏gameplay代码编写的复杂度。时间同步放在游戏线程中，如果当前游戏线程在渲染命令压栈结束之后，还没有达到固定两帧之间设定等待的时间间隔，则等待一定的时间。

**2.4 场景更新模块的优化**

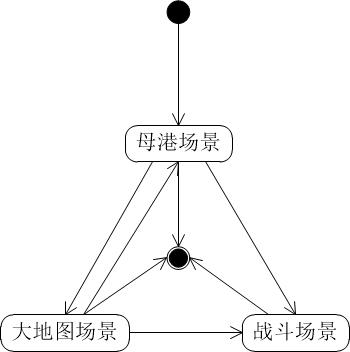
****

图3.5 场景状态机的切换

Cocos2d-x自带了场景管理系统,Cocos2d-x中场景分为Scene和Layer，其中一个 Scene下可以挂载任意个Layer,Layer下也可以继续挂载任意个Layer[20]。我们通过一个公共的栈来控制Scene的加载，UI渲染树每次只固定渲染栈顶场景。使用过程中异步加载场景类中的所有数据，并把该场景类的指针存在栈中。或是当切换场景时，切换到的场景只是短暂显示，之后还需要放回切换前的场景。比如配置场景中，在完成配置修改后，我们还需要前一个场景。这种情况中，我们没有必要释放当前场景的资源，所以我们把当前的Scene保留在栈中，向栈中压入新的Scene指针，同时加载新的Scene资源。当需要切换回原先的Scene，我们再把当前场景出栈。

该场景在设计上有自己的好处，首先符合Cocos2d-x自己的内存管理模式。从栈中删除的场景资源，被GC自动回收。利用单层状态机去控制场景切换，整体设计简单，在调试的过程可以在场景进出栈的时候打log[21]。但在卡牌类游戏中，界面的场景切换相比更复杂，在单个Scene中，我们需要保证数个Layer不动的，切换其他Layer。如果使用原有的场景更新模块，势必在编写gameplay代码时带来极大的不便。针对这种问题，我们把一个Scene拆分成多个Scene去控制场景切换，但势必会有重复的资源加载。所以在这里我重构了场景更新模块，使用分层状态机的方法去管理场景切换。

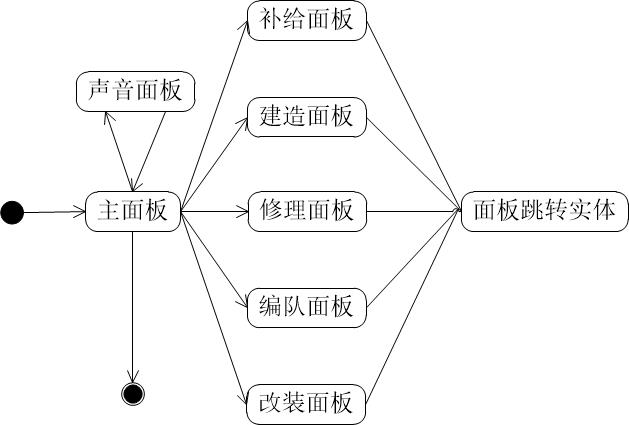
****

图3.6 母港场景下面板状态机的切换

**2.4.1 分层状态机介绍**

分层状态机本身是为了处理状态机中状态太多不好维护的问题，把不同的状态抽离出来，分离成几堆不同的状态集合。每个集合里边使用一个小状态机，再使用一个大的状态机去控制状态集合的切换。

比如说当我们要设计一个模拟人生的玩家AI，玩家有吃饭，睡觉，学习，玩电脑游戏，洗澡，上厕所等不同的状态，我们首先根据玩家所处的场景把这些状态分类成不同的状态集合。在玩家移动的过程中，同时进行大状态机的切换，在一个场景下我们可以进行一系列小状态的切换。比如说在卧室内，可以从睡觉切换到学习，但加入玩家如厕指数上升到一定水平，我们需要先利用大状态机把玩家的位置状态从卧室切换到厕所，再把小状态切换到上厕所。

层次状态机限制了状态的跳转，状态集合内部的状态不需要关心集合外部的状态。一定程度隔离了无关的状态。游戏gameplay的bug大部分都出现在了AI逻辑上，使用分层状态机的方法，可以大大减少状态机的复杂度，从而大大减少Debug时间。同时如果双层状态机不够，我们可以在定义更多的状态层以降低跳转链接数。

**2.4.2 Cocos2D-x的场景更新模块**

在Cocos2d-x中，一个场景是一个以Scene为根节点的UI树，Scene中包含一个场景的所有UI元素，如按钮，图片，精灵等。所有的Scene通过栈来控制，同一时间只能有栈顶的一个Scene被渲染，栈中的Scene全部是加载入内存的。我们通过压栈的方式来把新Scene加载，通过退栈的方式来释放Scene的内存。Cocos2d-x对于场景管理主要提供了三个API。ReplaceScene()直接删除当前的场景，并且把新的场景压栈。PopScene()直接把栈顶场景出栈。PushScene()把新的场景入栈，不管之前的场景。

**2.4.3 改良后场景更新模块**

我们在Scene类保留的基础上，提出了Panel类，Panel类是多个Layer的组合，相当于特定场景中多个Layer切换的最小单位。通过在Scene中以双向队列的方式存储，一个Scene可以挂载多个特定的Panel，这些特定的Panel只能在该Scene下使用，这里使用特定的Panel类需预先指定他所对应的场景类模版。当我们想定义一个该场景类下的Panel，新的类继承于被场景类特化的Panel类。

Template<class Scene>

class Panel:public Node{…};

Panel上可以挂载多个Layer,Panel类的切换和显示通过场景类所含的小状态机完成。我们为Scene提供了新的API,ShowPanel,GetPanel,ErasePanel以及SetPanelVisible。这些API控制了Panel的切换，在ShowPanel的过程中，通过该Panel类枚举的方式来指定Panel类型，首先去到该Scene下的双向队列去寻找该Panel类，如果没有找到，新建该Panel类并添加到队列中。之后运行Scene下的小状态机，完成Panel的加载和释放。Panel继承于Node，在定义新的Panel类如果我们想提供Panel类的进入和退出操作，我们需要重构Node所提供的OnEnter和OnExit，这两个函数相当于状态的退出以及进入操作，我们可以根据当前Panel类来指定相应的退出和进入操作。同时在这个过程中，为了达到我们的设计目的，我们可以在ShowPanel的API中指定，前一个Panel的内存是否释放，以及如果不释放该内存，前一个Panel是否显示。这就完成了之前我们所说到的，不同Layer切换，某些Layer不变的需求。相较于原先cocos2d-x所提供的场景管理方式，一个渲染批次最多只有一个场景类被渲染，我们可以人为指定一个场景类中所携带的Panel类是否渲染。

场景类的控制，依然依靠Cocos2d-x自身的场景类栈控制。但场景类下面，并不直接挂载Layer类。而是挂载Panel类。为了方便UI渲染树的遍历，场景类下的子节点队列同时保留有Panel类的指针。

我们提供了一个叫做LayerCover的类，该类以组合的方式挂载在Panel类上， 该类自带遮罩层，为了防止触摸事件传递到该Panel下面其他UI对象中。同时我们可以设定该遮罩层的透明程度，从全黑到全透明。

相较于之前Cocos2d-x的场景管理方式，重构后的代码有着极大的优势。首先方便了Layer的管理和切换。我们解耦了Layer组切换中各种过程代码，直接可以把它们写入自己模块中的OnEnter和OnExit中。给我扩展新的界面提供了极大的方便。同时在切换过程中，我们可以保持某个UI模块不变的同时，实现另外一个UI模块加载。

**2.5 事件分发模块的优化**

**2.5.1 Cocos2D-x的事件分发模块**

Cocos2d-x的事件分发模块，基于订阅者模式设计。订阅者模式可以减少了软件内模块之间的内聚，同时保持模块之间高效通信。游戏过程中，我们随时要处理外界的Input，比如说触摸点击，键盘输入，加速球移动等等。所有外界的Input全部是由事件管理模块分发。并且我们也可以定义引擎独有的事件。

订阅者模式主要包括三个部分，订阅者，事件以及分发者。Cocos2d-x中订阅者统一继承于EventListener,每个EventListener由回调函数，订阅者类型type，以及一个listenerID组成。当然回调函数可能不止一个，比如说Touch事件的订阅者提供了onTouchBegin,onTouchMoved,onTouchEnded三个回调函数。listenerID对应着一个事件源，分发过程中我们根据事件源找到相应的listenerID。要是用订阅者首先我们应该新建相应的订阅者同时注册它的回调函数。并使用分发者的API把他添加到分发者的队列中。Cocos2d-x有唯一的分发者类EventDispatcher,这个类负责添加，删除订阅者，向订阅者分发事件。订阅者保存在分发者的队列中，在一次完整的事件分发过程中，首先从IOS或者安卓层主动调用函数，把事件的相关信息传入到C++层，之后我们依据信息新建相应的事件，通过EventDispatcher的分发函数进行分发，传给对应的listenerID的订阅者上。在分发过程中一个listenerID可能对应多个订阅者，多个订阅者依据自身事件的优先级，以及所绑定Node在UI渲染树中的位置去判断分发顺序。

**2.5.2 事件分发模块优化**

在游戏的过程中，我们有时需 要暂时某一类事件的分发。比如在进行场景切换时，我们需要禁止触摸事件的分发。但原有的事件分发模块并没有提供相应的功能。所以我们需要在原有引擎的基础上进行改良。首先可以通过listenerID指代一种事件所对应的订阅者，为了加快订阅者遍历速度我们在EventDispatcher中新建一份映射表(map),根据listenerID去映射到他所对应所有订阅者，同时事件模块还保持另外一份映射表，从listenerID映射到EventPauseGuard类。EventPauseGuard类用来管理一个listenerID所对应的所有事件是否允许分发。依靠我们定义的两个宏命令EVENT\_DISPATCHCER\_RESUM(listenerID)和EVENT\_DISPATCHCER\_PAUSE(listenerID)来控制。内部保留一个计数器，计数器最少为0，每调用一次pause计数器加1，每调用一次resume计数器减1 。当计数器为0时，事件可以分发。使用计数器的方法使得嵌套停止或者继续事件分发成为了可能。

1. **游戏系统的需求分析**

**3.1 游戏业务描述**

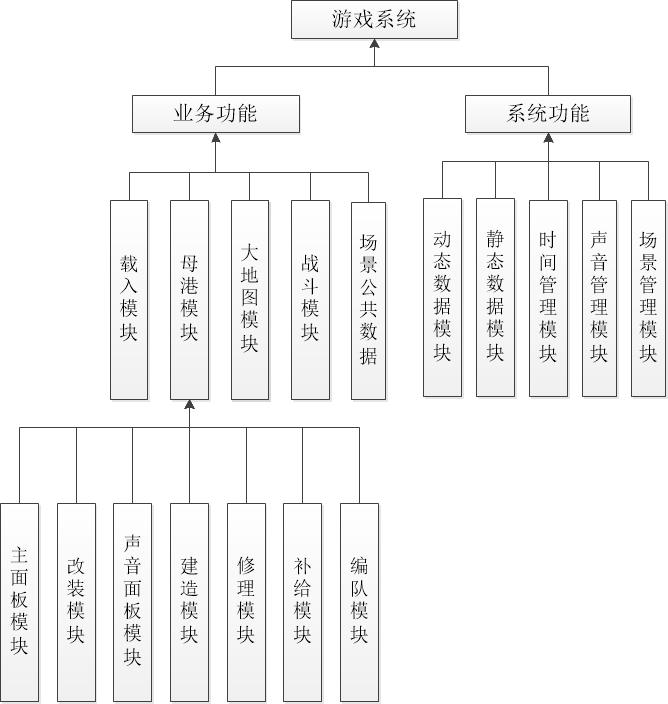


图4.1 游戏系统功能模块图

游戏中一共有四个场景。载入场景(LoadScene)，母港场景(PortScene),寻路场景(NaviScene)，战斗场景(BattleScene)。玩家在母港场景中升级卡片，强化卡片特有属性。在战斗过程中，会消耗卡片的耐久程度，所以母港场景中也提供了修理卡片耐久度的功能。玩家需要从所有的卡片中挑出几张卡片编成一队，所以母港场景中也提供了编队的功能。我们依据不同的功能设计成不同的面板类，各个面板下面还对应着自己的数据层，同时共享着从动态数据和静态数据中装载的共享数据。玩家编队完成以后，单击出击按钮，会自动切换到寻路场景。寻路场景中最重要的是一张大地图。大地图由很多个小的节点组成，玩家在地图的一个点初始化，会自动寻路到某个目标点，玩家无法自行选择移动方向。节点分为补给点，攻击点以及特殊功能点。玩家路过攻击点，会自动进入战斗场景。战斗过程采用回合制的方式，己方单位为初期编队选择的卡片，敌方单位的数据在战斗开始时从静态数据中加载。战斗结束以后，玩家可以选择返回母港场景，或者在寻路场景上继续向Boss点前进。Boss战结束后，玩家自动返回到母港场景。

游戏底层数据分为动态数据和静态数据，动态数据泛指玩家特有的数据，存于数据库中。静态数据泛指游戏中所有玩家相对一样的数据，包括基本的卡片，装备属性及游戏运行时并不可少的配置数据。这些数据在实际开发过程中通过Excel或特定开发工具配置，配置完成之后，由脚本工具自动生成二进制文件或者对应的静态文本文件。本文中的所有静态数据由作者在Excel中配置好，之后转为CSV文件。所有的静态数据全部在游戏初始化时加载[22]。

**3.2 游戏非功能需求分析**

**3.2.1 声音系统需求分析**

Cocos2d-x 引擎提供了自带的CocosDesion库，负责游戏内部的声音管理。但在实际使用时会有诸多不变，并且在多个音效同时播放时会出现问题。所以我们在Cocos2d-x声音接口的基础上，自己封装一层，构建了声音管理系统。并且针对音效叠加问题等进行了一定的优化。

我们另外提供了声音调节面板，实现拖拽条实体。通过拉动不同的拖拽条来控制背景音乐或音效的大小。

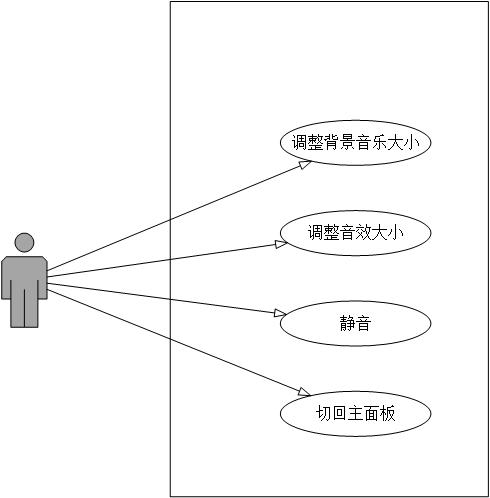
****

图4.3 声音系统用例图

**3.2.2 时间管理系统需求分析**

在修理系统和建造系统中，都需要间隔一段时间运行一段函数。为了方便后期整体添加功能和优化，我们基于Cocos2D-x原有的时间管理类添加一个新的时间管理系统。新的时间管理系统负责游戏GamePlay方面的定期刷新。我们需要提前注册回调函数，并且相应的更新时间。该模块还会提供一个稳定的时间戳，该时间戳不会因为玩家修改系统时间而改变。

**3.3 游戏功能需求分析**

**3.3.1 菜单模块需求分析**

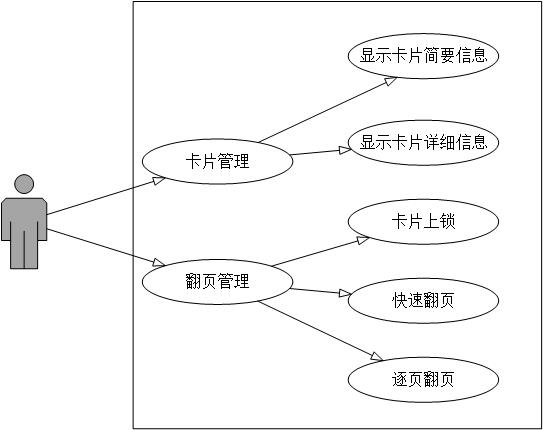
菜单模块用于在游戏中选择和修改卡片。游戏中一共有三个系统需要用到菜单模块，分别为编队系统，修理系统，建造系统。菜单模块分为子菜单和母菜单，子菜单和母菜单都实现移入移出功能，在移入和移出过程中屏蔽所有的按键事件，并且附加背景遮罩。****

图4.5 菜单模块用例图

母菜单用于显示玩家所含卡片的基本信息，而且可以根据特定卡片属性对卡片排序，并且提供一个锁定按钮，凡是锁定伤的卡片。我们不能从玩家背包中摧毁。由于玩家卡片可能无法在一张页面上显示，所以母菜单需要提供翻页功能。

子界面显示卡片的详细信息，并且提供了进一步修改卡片信息的按钮。

**3.3.2 主界面模块需求分析**

在主面板模块开启之前，我们需要完成游戏的加载过程,加载中各个步骤如图3. 所示。主面板模块是玩家最先接触到的面板，主面板模块主要为了切换到各个系统提供了方便。主面板模块包含了界面背景，边框等UI元素。同时含有五个不同系统的动态切换按钮和声音修改模块的切换按钮。以及界面右边的动态立绘。

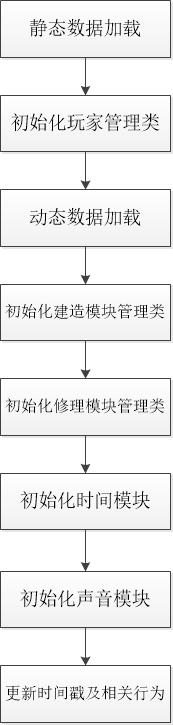
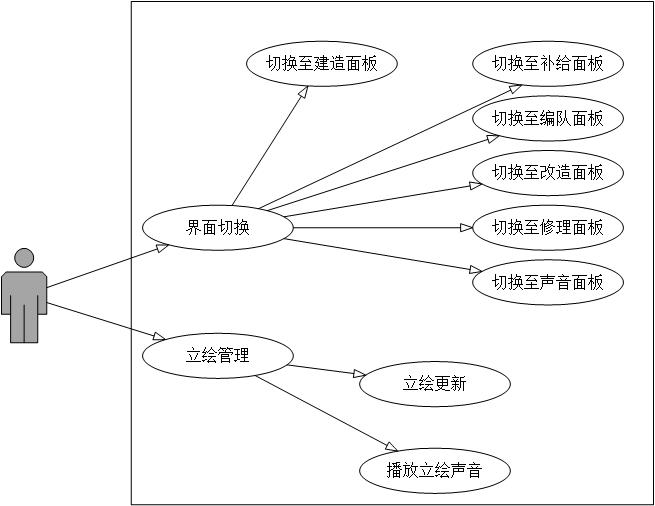


图4.7 客户端初始化步骤

****图4.8 主面板模块用例图

**3.3.3 面板跳转模块需求分析**

在菜单场景中，为了方便不同面板的跳转，提供了一个面板跳转模块。该模块在主界面中不显示，但在其他界面显示。该实体负责在不同面板进行切换，并实现切换过程中的动画操作。其中包含渐隐渐现效果，屏蔽Touch命令，面板跳转实体移动等。

**3.3.4 补给系统需求分析**

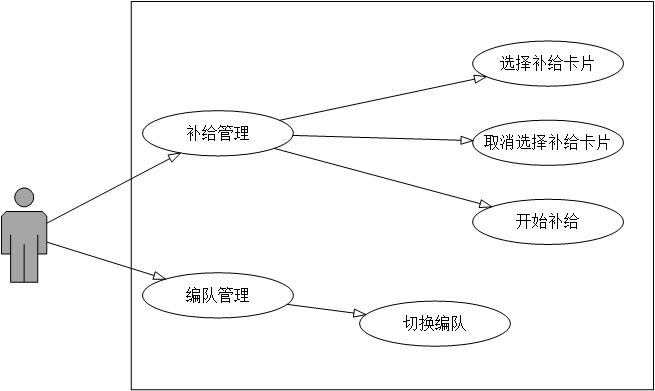
****

图4.10 补给系统用例图

玩家出击过程中，每在一个节点进行战斗，就会消耗当前编队中所有卡片相应的资源。如果在战斗时，当前卡片的资源槽是空的，卡片的伤害会被大幅度降低。所以在每次出击之前，我们需要补充资源。所以游戏相应提供补给系统。补给系统用例图如图3. 所示。

补给系统可以同时补给多个卡片，想要补给的卡片应在补给面板中提前选中，如果当前玩家所持有的资源不足，则无法选中。补给面板可以显示编队中卡片的基本信息，比如该卡片现有携带资源数量和可以携带的总资源量，每张卡片携带多种资源。补给面板还需要提前绑定添加卡片动画，移除卡片动画，补给动画等。

**3.3.5 编队系统需求分析**

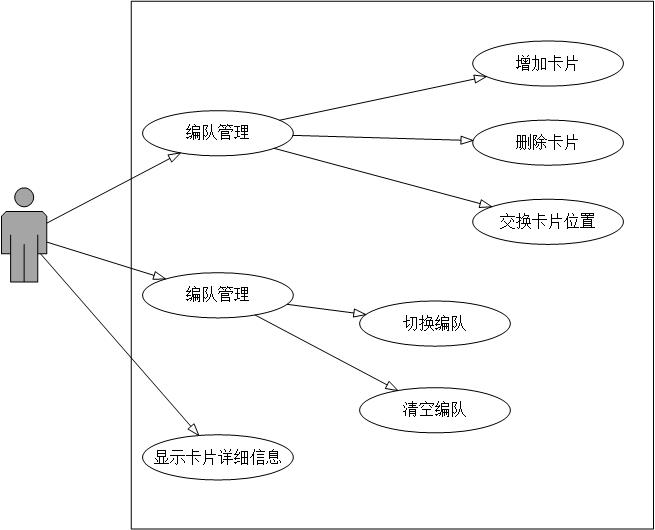


图4.12 编队系统用例图

游戏中，不同地图中不同节点里的敌人设定是不一样的，我们要针对不同敌人选择不同的迎击卡片。所以需要制作一个系统，可以选择和调整编队方案。

游戏中玩家可以持有多个编队，针对玩家等级，我们提供了不同的编队数量，在玩家等级提升的过程中总编队数量慢慢增多，在编队解锁以后，编队面板提供切换到不同编队的按钮。编队面板可以显示当前编队内所有卡片的基本信息，尤其是与战斗相关的属性值。编队面板还需要新增卡片动画，移除卡片动画，切换卡片动画等。

**3.3.6 建造系统需求分析**

游戏中，我们可以通过推图和建造的方式获得新的卡片。建造一张卡片需要资源，不同的资源组合所建造出来的卡片不一样，越是稀有的卡片需要的资源越多，越是稀有的卡片所需要的建造时间越长。最多有4个建造位置，每次建造过程中，占用一个建造位置。随着玩家等级的提升，建造位置越来越多。我们专门提供了一个建造面板。

1.触摸建造按钮或者面板跳转实体的建造按钮用以跳转到建造面板。

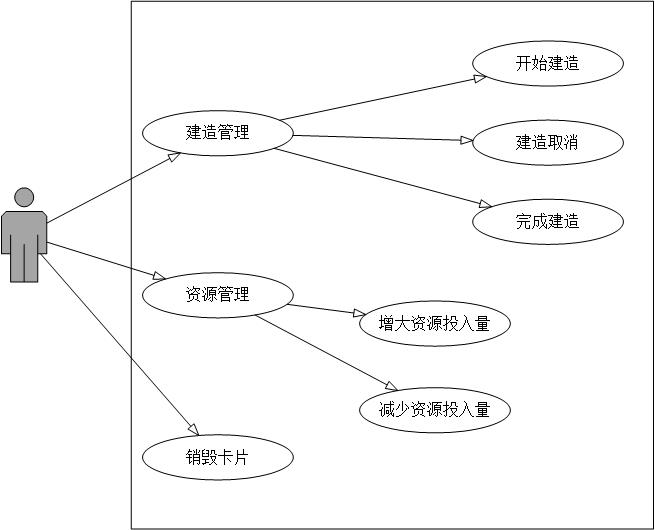
2.提供了建造以及删除卡片的两大类功能，分别通过跳转按钮进入当中。

3.建造卡片总共需要四种资源，每种资源对应一个UNIT，通过触摸UNIT上的按钮，我们可以增多或减少资源。

4.开始建造到建造未完成这段时间，建造占据一个建造位置。建造时间达到，该建造位置显示建造完成，需要人为再单击一次按钮，把建成卡片添加到背包中。

5.提供删除卡片的功能，如果该卡片禁止修改锁被打开，我们不可以删除该卡片。

6.提前绑定建造面板动画，包括建造中动画，建造结束动画，建造开始动画。

****图4.14 建造系统用例图

**3.3.7 修理系统需求分析**

战斗过程中，敌方单位会发动攻击，如果攻击被击中。我方卡片会被扣除相应的血量。所以在推图结束之后，我们可以修理受损的卡片。卡片的修理需要适当的时间，时间通过卡片的等级，受损程度，卡片的种类计算得来。每个卡片修理占用一个修理槽，可以实时看到剩余修理时间。

1.触摸修理按钮或者面板跳转实体的修理按钮给以跳转到修理面板。

2.构建修理槽，单击修理槽的按钮跳转到菜单模块，此时我们只能选择受损的卡片。

3.修理结束以后，修理槽自动返回空闲状态，卡片恢复到满血。

4.提前绑定修理面板动画，包括修理中动画，修理结束动画，修理开始动画。

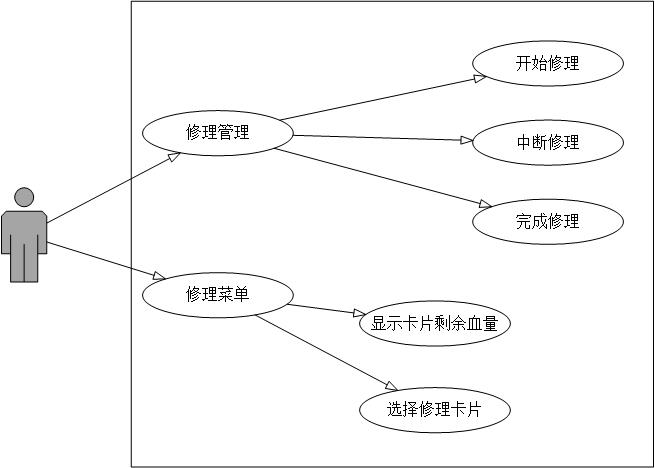
****

图4.16 修理系统用例图

**3.3.8 大地图系统需求分析**

当玩家设定好适应的卡片编队，保证卡片处入最佳的状态，我们便可以进入大地图场景。游戏提供多个地图供选择，不同的地图掉落不同的卡片。而在游戏中，我们获得卡片的主要方式之一就是推图，稀有卡片只有很小的掉落几率，玩家为了获得自己心仪的卡片需要大量的推图。并且随着游戏的难度逐步提升，推图的难度越来越大，玩家只有选择适当的卡片组合才能推图成功。玩家可以中途退出推图，但稀有卡片只在boss点掉落，但中途的战斗点不可以跳过，并且在大地图场景中，我们不提供修理卡片和补给卡片的功能。而卡片的修理度和补给度影响着战斗环节的流程。所以在前面的战斗点中，我们要保证卡片不会有太多损耗，才能在最后boss点时，有几率打赢boss。当然其中存在大量运气成分，比如说大地图场景移动路线都是事先随机生成好的，玩家不一定能到达最终的Boss点，当然卡片的阵容组合也是很重要的。这种基于运气和配合的游戏模式，正是卡牌类游戏的魅力所在[27]。

1.提供进入大地图场景的方法，选择不同的地图进入不同的大地图场景。

2.构建一个大地图场景，其中包括玩家所面临的关卡点。

3.不同的关卡点实现不同的功能。

4.大地图场景下，玩家通过自动寻路的方式，选择出一条最优的路线到达目标点。

5.在到达战斗点后，我们可以选择进入战斗模式的阵型，不同阵型影响玩家的战斗环节。

6.提供掉落面板。

7.提供大地图场景所需要的动画，包括玩家移动动画，玩家进入移动动画，玩家到达目标点动画。

**3.3.9 战斗系统需求分析**

战斗场景为游戏核心部分，所有的战斗环节全是在战斗场景中完成的。战斗环节采用回合制的战斗机制，分为几个核心阶段，如图4.21所示。索敌过程，第一轮预备打击，第二轮预备打击，第一轮打击，第二轮打击，收尾阶段。全部战斗流程由系统自动生成。当对方摧毁我方卡片，我发卡片需要从我方数据库中移除。为了降低游戏的难度，当我方卡片血量过低的时候，会自动触发低血量保护，敌方攻击力会大幅度降低但不是完全没有。这样加大了我方卡片被消灭的难度。该战斗场景是游戏的核心玩法，敌方的阵容是固定不变的，所以玩家要在进战斗前，提前配置好战斗阵容，同时由于在大地图不同战斗点之间不能进行更改编队，修理卡片，补给卡片等操作。所以我们针对敌方所有战斗点的敌方配置，来调整我方最佳阵容。这种玩法为游戏提供了大量的可玩性。

1. 提供进入战斗场景的方法，不同战斗点，加载的背景场景和敌人卡片不同。

2.通过状态机去控制游戏中各个阶段。

3.提供战斗场景中战斗单位和游戏中加载的动态数据以及数据库之间的交互方式。

4.设计各个回合玩家卡片和敌人卡片之间的交互操作。

5.提供战斗场景所需要的大量动画。

****

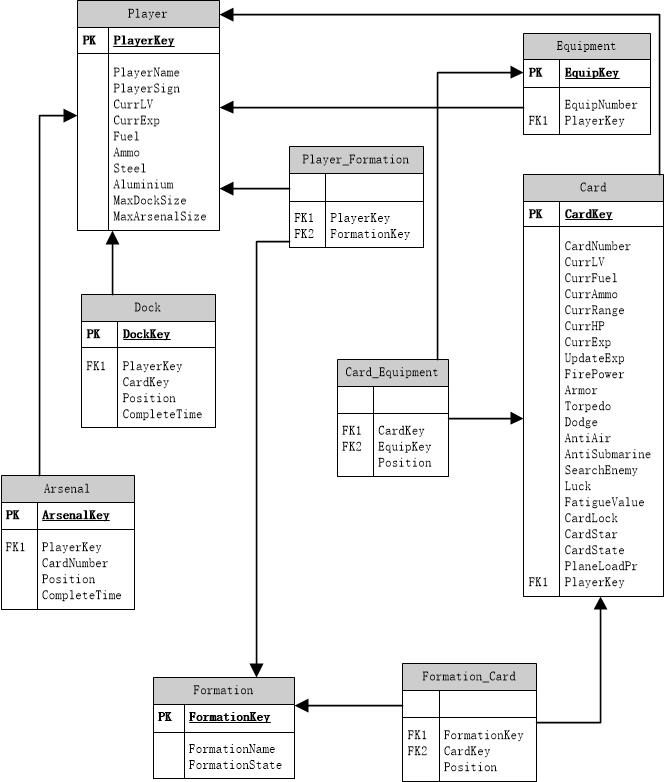
图4.21 战斗流程所有阶段

**第4章 游戏设计与实现**

**4.1 系统体系结构设计**

**4.2 系统数据库设计**

游戏中需要存储大量动态数据，态资源是指根据不同玩家，需要随时改变的数据。比如说玩家所拥有的装备种类和数量，玩家的等级，玩家通关情况等等。通常使用数据库来保存玩家相关的动态数据，本游戏使用本地的sqlite3数据库。Sqlite3是一款轻型的数据库，是遵循ACID关系型的数据库管理系统，适合手机，嵌入式等轻量级产品，占用资源非常的低，通常几百K内存就足够了，比起大型的数据库他的速度要快很多。图4.1为本游戏核心数据库表单。

 图4.1 本游戏核心数据库表单

**4.3 系统静态资源模块设计**

静态数据是指游戏中游戏配置，装备信息，道具信息，技能信息等恒定不变的数据。静态数据在游戏运行过程中不需要改变，游戏中只需要提供静态数据的读取方式。所以只需要保证静态数据能被高速的读入内存中，并且占用的空间不会太大。作者主要使用了两种类型静态数据XML和CSV，CSV格式可以直接从excel表格导出，不用编写转换脚本，并且CSV作为最简单的数据类型，它所占的内存空间非常小。在游戏中，只要对应简单的写入函数就可以完成CSV格式的加载，但是CSV格式不支持树状数据结构的存储，比如说地图寻路信息，就不可以用CSV格式存储。所以对于其中含有复杂数据结构的资源，使用XML存储。XML比CSV支持的数据类型更多，并且配合第三方库RapidXml加载速度也很快。在静态资源加载和压缩的过程中，同时需要对静态资源加减密，游戏中我们使用MD5算法。图2.4为本游戏核心静态数据表单。

在游戏中，有一些信息虽然不和特定玩家有关，但是需要保留当前信息，以便下一次游戏启动时使用。比如说游戏当前的时间戳，声音的大小，背景音乐的曲目，这些信息可以存在游戏客户端可读写的XML中。作者使用SAX来读写该类XML资源。

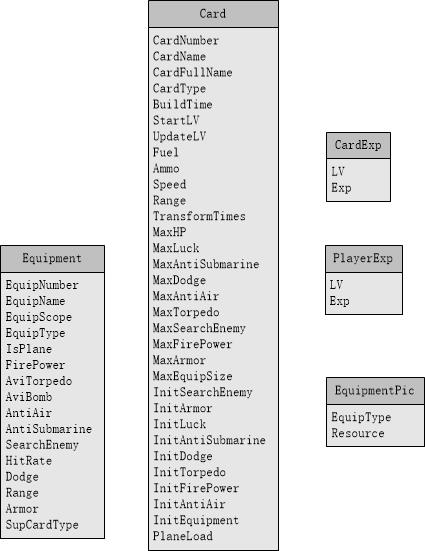


图4.2 本游戏核心静态数据表单

**4.4 游戏非功能设计与实现**

**4.4.1 声音系统设计与实现**



图4.3 声音面板界面

声音系统把声音分为音效和背景音乐两种。背景音乐在初始化时便把相关资源加载入内存，音效在即将播放时才把相关资源载入内存。在游戏运行过程中，一帧中可能触发多个音效，相同的音效会出现声音叠加，影响了最后听到的效果。所以在游戏运行时，需要合并同一帧数或者相邻几帧相同的音效。这里采用把每一个触发的声音封装成对象的方式，对象里边存入调用的音效的id，音效命令保存于一个队列中，在播放声音前合并相似的音效，再调用引擎提供的相应API播放音效。

同时我们为声音模块提供了上层的界面SoundPanel，该Panel属于主界面场景类的特有面板，并且利用Cocos2d-x的事件管理实体构建了SoundBar。依靠SoundBar，我们可以修改音量的大小。音量大小的管理统一通过SoundInfo来完成。

**4.4.2 时间管理系统设计与实现**

我们提供TimeMgr时间管理类，管理类中保存着一个队列，队列存储着和时间刷新相关的信息。我们需要前期注册回调函数以及间隔时间，以毫秒为单位，管理类会把相关信息构建成对象放到队列中。每隔固定间隔时间，调用事先注册的回调函数。同时管理类也会自动生成此回调函数对应的时间戳，存到静态表中。因为假如游戏切到了后台中，或者游戏关闭了，我们是无法通过游戏循环进行时间刷新。所以每当游戏重新切到前台，或者游戏重新启动，我们会去静态表中寻找该回调函数上一次调用时的时间戳，之后计算出这段时间间隔中，回调函数需要调用多少次。

时间管理类可以获得当前时间戳，当前时间戳由std::chrono::steady\_clock::now()来得到，相比传统的时间戳获得方式，这种通过计算设备CPU运行时间内时间钟数量来获得时间戳的方式，时间戳的值不会因为我们修改了系统时间而改变。

**4.5 游戏功能设计与实现**

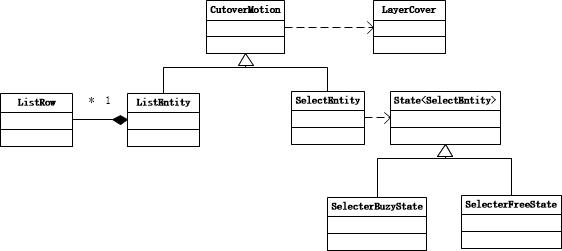
**4.5.1 菜单模块设计与实现**

如图4. 所示，本文菜单模块包括了子菜单模版类SelectEntity，母菜单模版类ListEntity。子菜单负责编辑和展示特定的一张卡片，母菜单负责展示玩家所有的所有卡片。为了分离各部分功能，实现模块间解耦，子菜单，母菜单间外部通信以及模版的内部逻辑，大量使用回调函数。同时在可能新增功能的地方预留了slot[23]。

ListEntity和SelectEntity均继承于CutoverMotion，CutoverMotion提供了实体移入移出功能，需要提前提供实体的原始位置和切入位置，以及所触发的切入事件和切出事件。CutoverMotion类带有LayerCover实体，负责捕捉切出动作的事件并且同时提供了遮罩功能。ListEntity中每一张卡片以简要信息的形式展示，每一张卡片占据一行，这一行封装成ListRow。ListRow同时保持着切换至相对应子菜单的监听器。母菜单中保存着队列，存储着所有卡片的指针，队列中指针的顺序既为卡片母菜单显示卡片的顺序。顺序的判断方法可以通过预先放在界面上的Toggle按钮进行切换。母菜单保留着一个模版队列，并且设有脏指针，在每帧中进行判断，如果模版队列与实际队列不相同，则重新加载所有的ListRow中信息。



图4. 菜单模块界面

图4. 菜单模块设计类图

子菜单分为两种状态SelecterBusyState和SelecterFreeState，带指子菜单是否被激活。在使用过程中，我们只需要把新的子菜单类继承于该子菜单模版，子菜单的特殊操作在子菜单子类的slot中定义，并且重载子类按钮所调用的回调函数[24]。重新定义静态Create函数以及Init函数，便可以完成设计的目的。

**4.5.2 主界面模块设计与实现**

图4.6 主面板界面

主面板模块中所有的触发器均继承于父类 MainPanelButton,根据功能不同分为NormalMainButton和BattleMainButton。在MainPanelButton中，我们使用Cocos2d-x自带的监听器构建出特殊的触发器。重构了onButtonBegin, onButtonMoved,onButtonEnded三个函数。主面板模块上一共有6个不同的触发器，为了针对6个触发器自己的大小和样子不相同这一特点，我们需在Init中提前定义触发器的位置和大小，并在Init中指定回调函数，回调函数调用了面板状态机的切换API。我们还需要提前注册UI动画效果，以及卡片立绘SignBoardGirl的动作。我们针对玩家的编队中首张卡片设定了脏标识，如果首张卡片更新，触发回调函数，更新卡片立绘[25]。 主面板模块的所有背景贴图全部封装在FurnitureBG中。

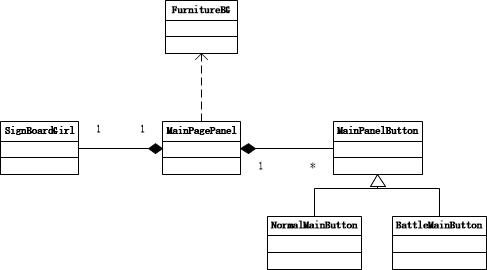


图4. 主面板类图

**4.5.3 面板跳转模块设计与实现**



图4.17 面板跳转实体

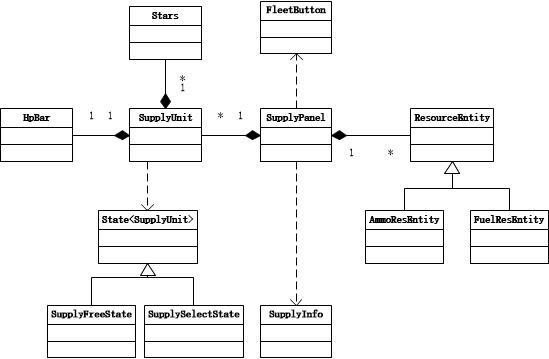
该类为LayerSelecter，由数个SelecterUnit组成，每一个SelecterUnit用做跳转至一个面板的跳转按钮，该面板必须为PortScene专属面板。

在配置SelecterUnit时，我们需要提前指定该Unit代之的面板。在切换过程中，由LayerSelecter来控制一系列操作。包括面板界面的渐隐渐消，LayerSelecter上指示符的移动，面板的切换，禁止触摸事件触发，声音的播放。其中面板界面的渐隐渐消，通过调整提前挂载到面板上的蒙版的Alpha值来控制，预先Alpha为0,蒙版为全黑，当需要执行渐隐操作，把Alpha值缓慢升至100，当需要执行渐消操作，把Alpha值缓慢降至0。在Cocos2d-x中透明实体不会影响渲染效率，因为该引擎在处理透明实体时不会再开辟新的队列，而是把透明实体作为一个不透明实体处理。不在渲染管线或是CPU层次上做面片剔除操作。渲染管线仅仅剔除在屏幕之外的图元。其他全部用画家算法进行叠加，无论其是否透明或者不透明。

**4.5.3 补给系统设计与实现**

图4.9 补给面板界面

补给系统类图由图4. 所示，补给面板由SupplyPanel类构成，继承于Panel<PortScene>，为PortScene专属的Panel的其中之一。补给系统所需要的数据，存于SupplyInfo中。游戏中需要提供两种资源的补给，我们分别实现两种资源补给单元实体和他们所对应补给动画，由这两种资源统一的父类ResourceEntity实现[26]。两种资源的补给单元实体继承于这个父类，AmmoResEntity和FuelResEntity。切换编队的功能是由FleetButton实现，FleetButton本质上是由多个MenuItemToggle实现，我们需要提前注册编队改变以后所触发的回调函数。当前编队的每一张卡片占一个SupplyUnit，总共有两种状态SupplyFreeState和SupplySelectState。在SupplyPanel刷新过程中，如果碰到数据上的问题需要进行判断，统一调用SupplyInfo中的API。AmmoResEntity和FuelResEntity碰到数据上的问题，也去SupplyInfo中寻求答案。每添加一张卡片，把相应的卡片信息封装成结构体，存到SupplyInfo的映射表中，同时由AmmoResEntity和FuelResEntity控制播放补给动画。 补给按钮触发时，我们从SupplyInfo映射表中读数据，接下来触发面板上对应实体的响应动作。

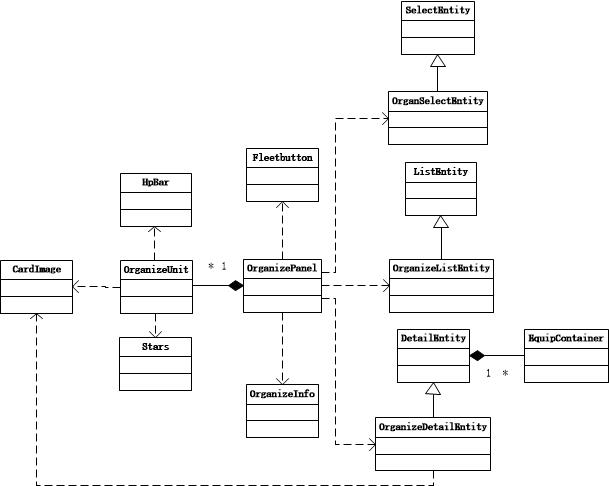
图4. 补给系统设计类图

**4.5.5 编队系统设计与实现**

图4.11 编队面板界面

补给面板由PortOrganizePanel类构成，继承于Panel<PortScene>，为PortScene专属的Panel其中之一。编队系统底层信息存在OrganizeInfo中。

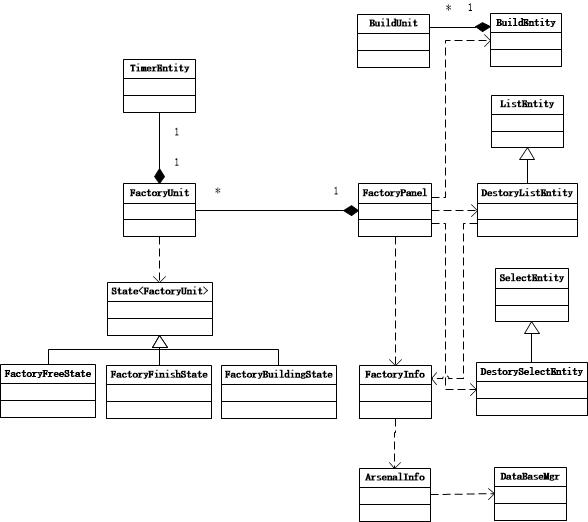
当前编队的每一张卡片占用一个OrganizeUnit，该单元本身由小型的状态机构成。一共有两种状态，NoUnit,HaveUnit。编队系统中的菜单由OrganizeListEntity和OrganizeSelectEntity组成，同时编队面板上也提供了OrganizeDetailEntity用作查看当前选中卡片的详细信息。使用FleetButton进行不同编队切换。编队过程中，首先我们要保证第一个编队的第一个卡片不能为空，并且编队中卡片和卡片中间不能有空闲，如果中间出现了空闲，把后面的卡片依次前移，在一次编队修改过程中，与原先样本组中编队位置进行比对，如发现不同，修改数据库中编队相关数据，并且调用相应的动作函数。在动作函数结束之后修改编队系统数据。编队切换过程中需要构建类似于百叶窗开合的切换动作，其中使用了模版缓冲区来实现多余UI的裁剪。

图4. 编队系统设计类图

**4.5.6 建造系统设计与实现**

图4.13 建造面板界面

建造面板由PortFactoryPanel类构成，继承于Panel<PortScene>，为PortScene专属的面板之一。建造系统所使用数据存于FactoryInfo类中。建造系统提供有两个功能，建造卡片和删除卡片。删除卡片的菜单为DestoryListEntity,DestorySelectEntity。建造槽本身蕴含一个小型的状态机，包括FactoryFreeState,FactoryBuildingState,FactoryFinishState三个状态。当建造槽空闲时状态为FactoryFreeState，建造槽正在建造卡片时状态为FactoryBuildingState，建造完毕之后状态为FactoryFinishState。FactoryBuildingState到FactoryFinishState的转换由时间管理模块中预先注册完毕的回调函数控制。建造功能上，我们需要构建提供不同的资源的实体。这个界面提供了增加资源，减少资源等一系列相关API。BuildUnit是这个父类的面板，我们在父类中预留InitIcon,GetPlayerRes,MinusPlayerRes，几个slot来提供不同资源重载。建造过程中，在开始建造时，把建造信息加入数据库，当建造完成时，数据库中数据不变。当玩家取出卡片时，把建造信息从数据库中删除并添加新的卡片数据。

****

**图4. 建造系统设计类图**

**4.5.7 修理系统设计与实现**

图4.15 修理面板界面

修理面板由PortRepairPanel构成，继承于Panel<PortScene>，为PortScene专属的面板之一。修理系统底层数据由RepairInfo负责管理。修理面板上所挂载的菜单由RepairListEntity和RepairSelectEntity提供。每个修理槽为一个RepairUnit。RepairUnit由一个小型的状态机控制，分为RepairFreeState,RepairRepairingState两种状态。两种状态随着当前修理槽的状态改变而改变，修理槽空闲时为RepairFreeState,修理槽有卡片时是RepairRepairingState。修理槽上倒计时功能由TimeEntity提供。修理过程中，在开始修理时，把修理信息加入到数据库，当修理完成时，把修理信息从数据库中删除并回复原有卡片血量。

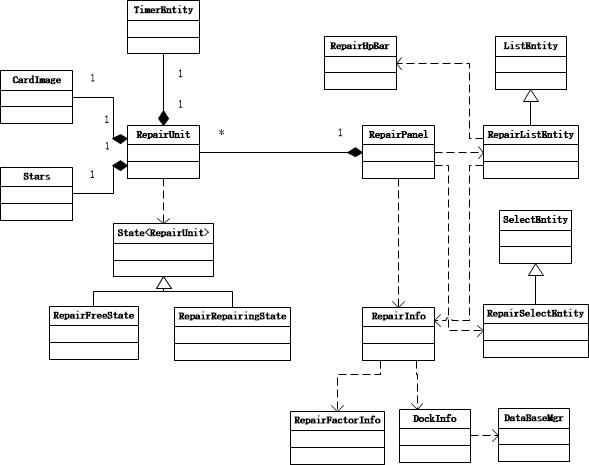


图4. 修理系统设计类图

**4.5.8 大地图系统设计与实现**

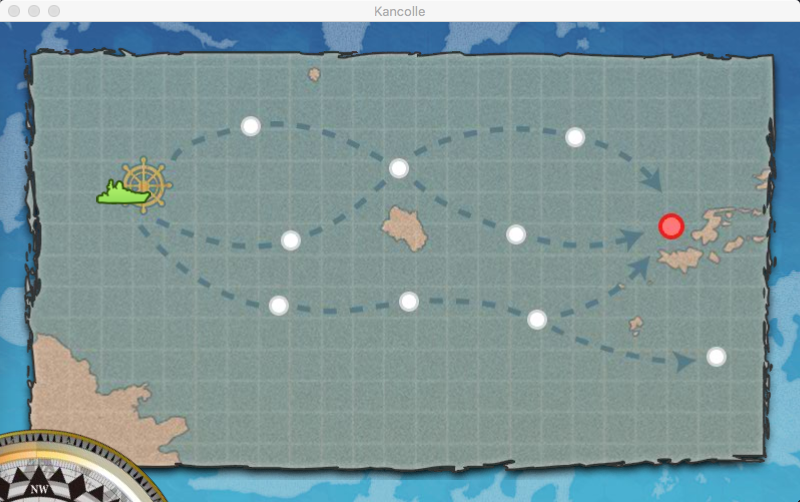


图4.18 大地图场景界面

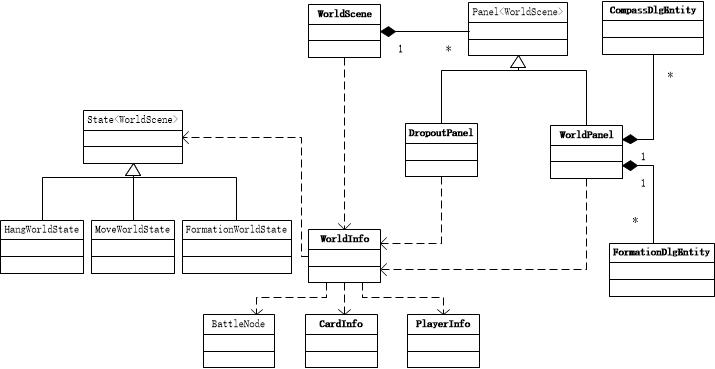
****

图4.19 大地图场景设计类图

如图4.19所示，大地图场景由WorldScene组成，并且可以挂载两个Panel，一个为WorldPanel，一个为DropoutPanel。WorldPanel负责大地图场景寻路界面，DropoutPanel负责掉落界面。WorldPanel在DropoutPanel触发时并不释放内存。大地图场景中所有的玩家数据全部放在WorldInfo中。我们进入WorldScene后，先通过相应的地图名字，读取对应的XML数据。并把XML数据解析，每个节点封装到BattleNode中，BattleNode包括节点名称，点的种类，他的子节点指针队列，他的父节点指针，位置(以该张地图为标准)。在过程中把潜在的Boss点指针存入队列中，在解析XML完成之后，随机从中挑选一个点作为Boss节点。之后使用A\*算法，得出我们的寻路路径，用队列来表示，并且把该路径存入WorldInfo中。

大地图中玩家角色需要从一个节点走到另一个节点，我们使用状态机去控制玩家。状态机为环状状态机，一共有SHIP\_HANG, SHIP\_MOVE,SHIP\_FORMATION三个状态。SHIP\_HANG负责移动判断是否满足移动条件，如果不满足移动条件直接退出。同时提供了罗盘动画，全部罗盘动画封装在CompassDlgEntity类中。SHIP\_MOVE为玩家移动过程时的状态，SHIP\_FORMATION当玩家移动到目标时出发这个状态，针对目标点不同类型我们提供不同命令。如果该点为战斗点，触发阵型动画，所有的动画封装在FormationDlgEntity里，玩家选择阵型之后，从该状态进入战斗模式。其中我们可以从SHIP\_HANG到SHIP\_MOVE，从SHIP\_MOVE到SHIP\_FORMATION,从SHIP\_FORMATION到SHIP\_HANG。每一步转换时都有判定是否满足转换到下一个状态的条件，如果不满足切回菜单场景。

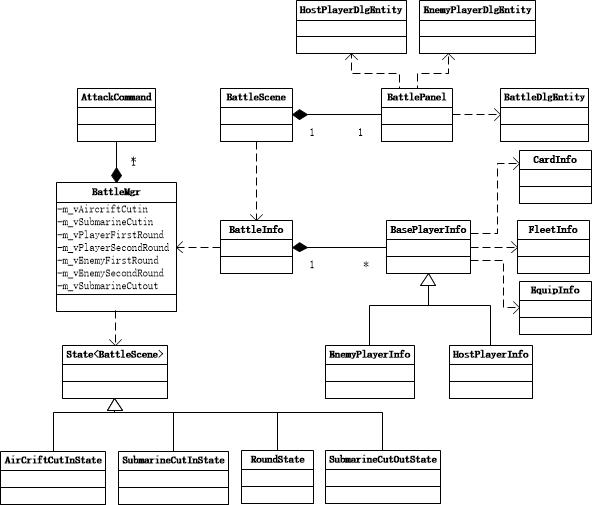
**4.5.9 战斗系统设计与实现**



图4.20 战斗场景界面

如图4.22所示，战斗场景类是BattleScene，其中只挂载了一个BattlePanel。但Panel上含有了大量的实体类，其中一部分实体类是为了满足游戏中的动画操作。战斗场景中所有的数据全部存储在BattleInfo中。

卡片游戏相对于其他游戏的一个优点。所需要与服务器的交互数量少，所以我们在加载战斗场景的同时，直接生成接下来战斗过程，把战斗过程的每一个战斗命令通过命令模式的方式封装成AttackCommand[28-29]，存入到不同战斗阶段的队列中。同时判定最后战斗评分，生成对应的奖励品[30]。之后一次性的更新数据库。接下来战斗环节中我们只是依次读取队列中数据，单纯的播放一遍战斗流程动画。所有的Command存在BattleInfo中，一个Command中包括攻击发起者是敌是友，攻击种类，攻击伤害，是否闪避，是否暴击，发动攻击者，被攻击者等数据。存在不同的战斗流程队列中，在播放战斗动画时，我们依次从不同战斗流程队列中读取命令对象，使用统一的动作机去播放动作。通过这种方式我们优化了与数据库之间的通信方式，把与数据库的通信降到了最低，并且解决了战斗环节中游戏退出之后带来的游戏判定相关问题。

图4.22 战斗场景设计类图

为了不与游戏中公共的动态数据混淆，我们需要先把游戏中需要参与战斗的动态数据复制一份，单独参与运算。其中包括玩家编队及卡片数据，还有敌人编队及卡片数据。我们为其提供了HostCard和EnemyCard两个类，他们共同继承于BaseCard。全部存在BattleInfo中。

战斗流程的控制采用单向状态机，只可从前一个阶段转到下一个阶段。分别有索敌过程，第一轮预备打击，第二轮预备打击，第一轮打击，第二轮打击，收尾阶段六个阶段。状态机提供三个函数Enter，Update，Exit。阶段切换通过阶段中的bool值去控制，所有的状态更新全部写在相应的Update中。并且提供相应的EnterComplate和ExitComplate控制标识。

**第6章 游戏测试**

**6.1 改良后的引擎测试**

在游戏系统测试之前，我们先对改良之后的多线程渲染版本的cocos2d-x引擎进行测试。查看引擎渲染功能是否完善，并且从渲染帧率和电量消耗上与原本单线程cocos2d-x引擎做了对比。

我们构建了一个测试场景，场景由不同的游戏实体组成。每一个实体为一个节点，该节点可能为Layer,Sprite，Text,Button。仿照正常游戏时复杂的场景管理机制，一个节点上挂载任意数量的其他节点，并且在游戏过程中会新加和删除节点[32]。本测试在MAC平台上进行，IDE选用Xcode并同时使用Instruments工具对游戏过程进行监控，Instruments可以检测当前模拟器是否存在内存泄露。相关测试信息如表6.1所示，测试结果如表6.2所示。

表6.1 引擎测试环境配置表

|  |  |
| --- | --- |
| 软件名称 | 所用软件 |
| 测试平台 | OSX El Capitan |
| 测试IDE | Xcode |
| 测试模拟器 | iPhone6 |

表6.2 改良前后引擎在不同drawcall下渲染时间对比

|  |  |  |  |
| --- | --- | --- | --- |
| drawcall数量 | 单线程引擎渲染时间  (微秒) | 多线程渲染引擎渲染时间  (微秒) | 加速比 |
| 0 | 5213 | 9775 | 1.8 |
| 1320 | 17809 | 15112 | 0.8 |
| 2376 | 33537 | 21562 | 0.64 |
| 5021 | 63098 | 42560 | 0.67 |

在场景中不含有任何实体时，多线程版本引擎在耗时上高于单线程版本。但随着场景中的DrawCall上升，加速比稳定在0.6到0.7之间。并且渲染场景无错误。在耗电量上，多线程渲染引擎的耗电量普通cocos2d-x引擎的70%。

所以我们得出结论，在实际应用上，多线程渲染引擎无论是在渲染帧率上以及耗电量上，相对于单线程引擎都有着很大的优势。

**6.2 游戏单元测试**

单元测试主要是对游戏中功能点进行测试，是对软件中最小的单元是否正确进行检验。单元测试的工作目标是程序正确，单元逻辑无误，并且可以对其中不正确的输入输出进行有效的处理。游戏测试是开发过程中很重要的一节，不但验证了开发是否正确，并且为我们进一步优化程序提供了数据上的支持。

游戏单元测试中，我们对需求分析阶段一条条需求进行扩展，把他们拆成最小可测试的单元。之后以面板为单位，做成表格，逐条测试该游戏单元是否会出现问题，并且针对具体的问题，提出相应的修改方案。经过游戏测试和修改之后，游戏可以达预期的效果。

**6.3 游戏性能测试**

表6.3 测试用机配置一览

|  |  |  |  |
| --- | --- | --- | --- |
| 实验硬件环境 | iPhone5C | Ipad Mini4 | Sony z3 |
| CPU | 苹果A6 | 苹果A8 | 高通骁龙801 |
| 内存 | 1GB | 2GB | 3GB |
| 系统 | IOS 7.0 | IOS 9.0 | Android 4.4 |
| 存储器大小 | 16G | 64G | 16GB |
| 电池容量 | 1510mAh | 6930mAh | 3100mAh |

表6.4 游戏性能测试结果

|  |  |  |  |
| --- | --- | --- | --- |
| 测试设备 | CPU使用率 | 平均内存(M) | 平均帧率 |
| iPhone5C | 52.23% | 79 | 59 |
| iPad mini4 | 58.12% | 85 | 57 |
| Sony z3 | 48,80% | 105 | 58 |

性能测试使用Instrument配合腾讯GT。分别统计了不同机型下，CPU使用率，平均内存以及平均帧率的数据。具体情况如表6.4所示。

从测试结果得出，三种机型均可以流畅运行，安卓机型平均占用内存率高。并且在运行过程中，游戏运行良好，没有出现卡机，退出等情况。所以游戏能够在大部分机型下正常运行。

分别对iOS和安卓进行出包，安装包大小大约为150MB，其中IOS的比安卓小10MB。可以满足市面上大部分的手机的存储空间要求。

# 结 论

本文主要介绍了基于cocos2D-x引擎的卡牌类游戏框架的设计与实现。首先引言部分介绍了选题背景，选题的意义，以及当前国内外领域发展情况。接下来对cocos2D-x引擎以及文中所设计的技术进行简要的介绍，技术包括寻路算法，静态数据和动态数据存储技术等。接下来文章就引擎优化与实现部分做了详细的说明，其中主要有两点，引擎渲染多线程优化，以及场景更新模块优化。然后作者详细说明了游戏系统的组织结构，针对其中重点模块的设计目的和游戏需求做了说明，同时描述了该模块的基本组建的实现过程和游戏逻辑的实现过程。作者接下来对游戏资源的后期优化做了一定的展开，其中包括纹理优化和加载纹理数据优化两个方面。最后对引擎和游戏进行了功能性的测试和性能测试。

本文主要的成果如下:

(1).构建了一套完整的卡牌类手游的游戏框架，其中包括完整的数据库和静态数据处理模块。卡牌类手游本身业务逻辑相差不大，我们可以在此基础上，进行简易的修改，基于本框架二次开发出新版本的卡牌类游戏。这对于大大缩短开发时间减少开发成本有着大量的好处。

(2).优化了引擎的渲染模块，cocos2D-x本身为单线程引擎，本文通过适当的方法实现了渲染线程和游戏线程的分离。提升了手机CPU的利用率，同时降低了整体的渲染时间以及电量损耗。

(3).优化了引擎的场景管理模块，作者重写了cocos2D-x的场景管理模块，使用双层状态机的思想去管理场景切换。减少了场景切换时不必要的资源加载和释放，同时简化了游戏业务逻辑的开发难度。在游戏业务逻辑代码的编写过程中，实际开发时间大大减少。

(4).作者对纹理加载进行了一定的优化，使用PVR格式对游戏中图片进行资源大小上的优化，同时构建了一个简单的资源加载模块，提升了游戏整体流畅程度。

在项目开发过程也发现了一些问题，这些问题需要进一步的改进:

(1).cocos2D-x引擎自带的声音引擎功能并不是很强大，后期可自行替换。

(2).该客户端为单机客户,前期预留了接口，后期如果需要，可以改成网络通信版本。

**参考文献**

* + - 1. 陈维.出货量占全球超四成国产游戏能否打破七年之痒[J].中国品牌与防伪，2016
      2. 尚云.站在十字路口的卡牌游戏[J]. 计算机应用文摘 2014
      3. 满硕泉.Cocos2D-x 权威指南[M].北京： 机械工业出版社，2013
      4. Cocos2D-x开发组.Cocos2D-x文档[EB/OL].触控科技.2014.http://cn.coco2d-x.org/article/index
      5. 施伟，王硕苹，郭鸣.跨平台移动应用中间适配层设计与实现[J].计算机工程与应用，2014
      6. 潘晓梦，邓建华，苏厚勤.一种跨平台移动应用方案的研究与实践[J].计算机应用与软件，2013
      7. Shekar，Sidharth.Cocos2D-x cross-platform game development cookbook[J].2016
      8. Roberto Ierusalimschy.Programming in Lua[M].3rd ed：Lua.org，2016
      9. Robert Nystromm.Game Programming Patterns[M].Genever Benning，2014
      10. Cocos2D-x 3.0优化提升渲染速度 Auto-batching[EB/OL].2014.http://blog.csdn.net/kaitiren/article/details/30478695
      11. Rajiv Eranki.Searching using A\*(A-Star)[OL].2002.http://www.mit.edu/eranki/www/tutorials/search
      12. Dan Ginsburg ，Budirijanto Purnomo，Dave Shreiner，Aaftab Munshi.OpenGL ES 3.0 Programming Guide[M].Addison-Wesley Professional，2014
      13. Tomas Akenine-Moller，Eric Haines ，Naty Hoffman.Real-Time Rendering[M].3rd CRC Press，2008
      14. Jason Gregory.Game Engine Architecture[M].A K Peters，2009
      15. 痞子龙3D编程.多线程渲染[EB/OL].2011.http://blog.csdn.net/pizi0475/article/details/6243064
      16. Unreal Engine 4 Documentation[EB/OL].Epic Game.2016.https://docs.unrealengine.com/latest/CHN/index.html
      17. 李坤，廖华丽.基于虚幻引擎的ROV实时运动仿真[J].计算机与现代化，2014
      18. 房燕良.虚幻4渲染系统结构解析[EB/OL].2016.http://geek.csdn.net/news/detail/106495
      19. 多线程渲染(Multithreaded- rendering)3D引擎实例分析 : FlagshipEngine[EB/OL].2010.http://blog.csdn.net/jinlking/article/details/56697492
      20. 李勇.基于Cocos2D-x引擎的游戏架构设计与实现[D].北京邮电大学，2015
      21. 秦春林.我所理解的Cocos2D-x[M].北京：电子工业出版社，2014
      22. Shekar，Sidharth.Cocos2D-x cross-platform game development cookbook[J].2016
      23. Martin Fowler.重构:改善既有代码的设计[M].北京：人民邮电出版社 2010
      24. 苏志同，石绍坤，李晋宏.手机游戏开发架构的研究[J].计算机工程与设计，2010
      25. 沈志超.基于Cocos2D-x的RPG手机游戏开发[D].东南大学，2015
      26. Brian W. Kernighan.The Practice of Programming[M].Addison-Wesley，1999
      27. 刘军.基于Cocos2D-x引擎的卡牌手机网游客户端的设计与开发[D].电子科技大学，2014
      28. Eric Freeman等.O’Reilly Taiwan公司 译.Head First 设计模式[M].北京：北京电力出版，2007
      29. Erich Gamma，Richard Helm，Ralph Johnson，John Vlissides.Design Patterns: Elements of Reusable Object-Oriented Software[M].Pearson Education，2000
      30. Kim Pallister.Game Programming Gems 5[M].Charles River Media，2005
      31. Andrew C. Beers，Maneesh Agrawala，Navin Chaddha.Rendering from Compressed Textures[C].ACM SIGGRAPH，1997
      32. Glenford Myers，et al.王峰，陈杰 译.软件测试的艺术[M].2nd ed.北京：机械工业出版社，2006

**作者简介**

张岩林，男，汉族，1992年8月10日出生，北京人。

2010年9月至2014年7月，就读于吉林大学软件学院，本科

2014年9月至今，就读于吉林大学软件学院，研究生

联系方式：zhyl19920810@gmail.com

**致 谢**

虽然研究生三年很短，但是我得到的收获很多。三年的实验室生活，我不仅在专业知识领域快速提高，还通过项目锻炼了自己的能力，在此我衷心感谢所有每一位帮助过的朋友们。

首先，我要感谢我的导师李文辉教授。本次论文是在李教授的亲切关怀和悉心指导下完成的。他严肃的科学态度，严谨的治学精神，精益求精的工作作风，深深地感染和激励着我。从论文的开始到最终完成，李老师都给予我细心的指导和不懈的支持。

其次，我要感谢王莹老师，王老师渊博的学识深深影响着我，使我专业知识和眼界都得到了提高。王老师的教诲给了我无尽的启迪，更是一笔可贵的精神财富。在此谨向其他实验室老师表示我最诚挚的敬意和感谢。

之后，我要感谢实验室的师兄弟姐妹们，在我硕士学位期间给予我帮助和支持。我还要感谢身边的朋友们，硕士的学习和生活，让我们共同进步，感谢你们给予我的所有关心和帮助，同窗之谊，永生难忘。

最后我要感谢我的母校——吉林大学，母校良好的学习环境和浓厚的学习氛围给了我一个良好的学习平台，让我不断充实自己，提升自己。

**2.3 寻路算法**

在游戏的大地图上，玩家需要以最快的方法自动寻路到目标点，因此寻路算法要尽可能的高效，可靠。寻路算法已经发展的很成熟，目前的寻路算法主要分为盲目搜索和启发式搜索。

盲目搜索算法又叫无信息搜索算法，其中最有代表性的要属Dijkstra算法。Dijkstra算法能找到节点间的最短路径，是典型的最短路径算法。算法通过深度优先搜索或者广度优先搜索两种方式，深度优先搜索对每一条分支路径深入搜索 到无法再深入为止，之后发生回溯。广度优先搜索先保证遍历完该节点所有相邻的节点，之后再向更深一层遍历。盲目搜索算法在搜索过程中，常按着既定的方法和策略进行搜索，没有结合具体问题，所以搜索效率比较低，所以只适合解决简单的问题，不适合复杂地图的寻路工作。

启发式搜索是通过加入启发函数的方式对寻路过程进行减枝，加快和简化了整个搜索的过程。最著名的启发式搜索算法是A\*算法[11]。加入了启发函数以后，我们在搜索的时候对每一个状态位置进行评估，使得每次搜索都能朝着最有希望的方向进行。



....................(2,1)

(2,1)公式为A\*算法简化公式，g(n)为从起始点到任意节点的实际代价，h(n)是启发函数，为从起始点到任意节点最佳路径的估计代价，其中估计代价不考虑障碍等一些影响问题。启发函数的设计有很多种，启发函数的设计对启发式搜索的效率至关重要，特定的环境要使用特定的启发函数，同时我们要保证启发函数运算消耗很小。常见的估算方法有曼哈顿距离，欧几里得距离，对角距离等，文中寻路算法我们使用曼哈顿距离，(2,2)公式是曼哈顿距离公式。

..................(2,2)



1. 新版UI系统

3.1 UI系统需求分析

3.1.1 游戏UI需求

3.1.2 卡牌类手游UI需求

3.1.3 该游戏UI需求

3.2 Cocos2D-X的UI系统

3.2.1 UI系统介绍

3.2.2 存在的局限性

3.3 新版UI系统总体设计

3.4 载入模块设计与实现

3.5 核心模块的设计与实现

3.6 系统级控件的设计与实现

3.6.1 Image

3.6.2 Button

3.6.3 Label

3.6.4 Toggle

3.6.5 SlideBar

3.6.6 InputEdit

3.7 游戏级控件的设计与实现

3.7.1 菜单控件

3.7.2 卡片头像

3.7.3 切换编队按钮

3.7.4 倒计时器

3.8 UI编辑器

3.8.1 UI编辑器的需求

3.8.2 UI编辑器的设计实现

问题:回掉的顺序怎么弄？