|  |  |
| --- | --- |
| 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：计算机图像处理与虚拟现实技术**  **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：工程硕士

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**

在人们追求更快捷便利的服务形式的大趋势下，移动游戏出现在人们生活中。其中卡牌类手游已经成为国内手游市场的主流，在苹果App Store中国畅销榜前50的游戏中，卡牌类游戏占到了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，对主流开源手机游戏引擎进行了总结。针对作者所要开发的2D卡牌类游戏，选定Cocos2D-x引擎进行开发。但该引擎无论在CPU利用率，场景资源管理，以及UI控件开发效率上均有所欠缺，不能很好的满足卡牌类游戏的需求。作者首先对相关技术进行了详细的研究，之后为了贴合所开发的2D卡牌类游戏，在Cocos2D-x的引擎的基础上，做了大量的改进。主要工作如下所示：

1．作者改写了当前引擎的渲染模块，实现了游戏线程和渲染线程的分离，提高了游戏运行时的CPU利用率，游戏运行帧数。

2．作者重写了场景控制模块，利用双层状态机的思想管理场景加载和释放，减少了游戏开发难度，同时优化了场景资源的资源管理流程。

1. 作者重写了整个UI系统，新版UI系统分为配置文件管理模块，核心模块，

系统级控件，游戏级控件，UI编译器。新版UI系统更加贴合所要开发的2D卡牌类游戏，便利于开发者，大幅加快了开发效率以及降低了开发难度。

本文使用这个框架，利用国外著名页游的游戏UI，应用这个框架重新制作成手机端卡牌游戏。

**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**

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 Top 50 game chart of Apple App store of China,Card games 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.

Based on the current situation of mobile game market, this paper summarizes the mainstream open source mobile game engine.Then Author select Cocos2D-x as the game’s engine due to the characteristic of the game. But author found that the game engine can not meet the needs of the game, both in the resource utilization factor, and the defects in UI system and scene manager. First the author carried out a detailed study of the relevant technology, then in order to fit the development of the 2D card game.Based on the Cocos2D-x,author made a lot of improvements.The main work is as follows:

1. Author rewrites the rendering module of the current engine,separates the gameplay logic and the rendering logic with two thread.In the final,ours work improves the CPU utilization and the game performance.

2. The author rewrites the scene module, uses the double state machine to manage the scene.In the final,ours work reduces the difficulty of game development, and optimizes the management flow of scene resources.

3. The author rewrites the entire UI system.This new version UI system consist with configuration file management module, the core module,

System-level widget, game-level widget ,UI editor. This UI system is more suitable for the development of 2D card games, and greatly speeds up the development process and reduces the development difficulty.

The author uses this framework and famous game’s UI,rebuild into a mobile games.

**Keywords:**

Cocos2D-x,Computer graphic,openGL ES,multithread programming, Mobile game framework

**目 录**

[第1章 绪 论 1](#_Toc15283)

[1.1 选题背景及意义 1](#_Toc4804)

[1.2 本课题的发展及研究现状 2](#_Toc22656)

[1.2.1 智能手机发展现状 2](#_Toc815)

[1.2.2 卡牌类游戏发展及研究现状 3](#_Toc14702)

[1.2.3 手机游戏引擎技术研究 4](#_Toc13824)

[1.3 本文组织结构 4](#_Toc23497)

[第2章 改进Cocos2D-x引擎的需求分析 6](#_Toc27547)

[2.1 卡牌类游戏的需求 6](#_Toc32145)

[2.2 Cocos2D-x游戏引擎 6](#_Toc29854)

[2.2.1 Coscos2d-x游戏引擎介绍 6](#_Toc29974)

[2.2.2 Coscos2d-x的结构 7](#_Toc15747)

[2.2.3 Cocos2D-x工作流 8](#_Toc22195)

[2.2.4 Cocos2D-x的UI系统 10](#_Toc1007)

[2.3 引擎存在的问题 11](#_Toc24899)

[2.4 本章小结 12](#_Toc27615)

[第3章 Cocos2D-x引擎的优化 13](#_Toc12089)

[3.1 渲染模块的优化 13](#_Toc6864)

[3.1.1 游戏循环时间同步方式 14](#_Toc22920)

[3.1.2 几种多线程渲染方式 15](#_Toc29078)

[3.1.3 Cocos2D-x的渲染模块 18](#_Toc25278)

[3.1.4 改良后渲染模块 19](#_Toc383)

[3.2 场景更新模块的优化 22](#_Toc13310)

[3.2.1 分层状态机介绍 24](#_Toc32753)

[3.2.2 Cocos2D-x的场景更新模块 25](#_Toc697)

[3.2.3 改良后场景更新模块 25](#_Toc21421)

[3.3 事件分发模块的优化 26](#_Toc27998)

[3.3.1 Cocos2D-x的事件分发模块 26](#_Toc29007)

[3.3.2 事件分发模块优化 27](#_Toc20057)

[3.4 本章小结 27](#_Toc10452)

[第4章 针对卡牌类游戏的UI系统设计与实现 29](#_Toc12)

[4.1 新版UI系统总体设计 29](#_Toc21531)

[4.2 配置文件管理模块 32](#_Toc25012)

[4.3 核心模块 36](#_Toc5834)

[4.4 系统级控件 38](#_Toc11349)

[4.4.1 Image 38](#_Toc8700)

[4.4.2 Button 39](#_Toc24609)

[4.4.3 Label 40](#_Toc17047)

[4.4.4 Toggle 41](#_Toc5621)

[4.4.5 ScaleImage 42](#_Toc30539)

[4.4.6 InputEdit 42](#_Toc7650)

[4.5 游戏级控件 43](#_Toc29572)

[4.5.1 菜单控件(ListEntity) 43](#_Toc23261)

[4.5.2 面板跳转实体(LayerSelecter) 46](#_Toc24266)

[4.5.3 列表切换按钮(ListButton) 46](#_Toc3335)

[4.5.4 倒计时器(TimerEntity) 48](#_Toc23668)

[4.6 UI编辑器 48](#_Toc1843)

[4.6.1 UI编辑器的需求 49](#_Toc1427)

[4.6.2 UI编辑器的完成效果 49](#_Toc29070)

[4.7 本章小结 50](#_Toc3959)

[第5章 针对卡牌类游戏的游戏框架测试 51](#_Toc2713)

[5.1 改良后的引擎测试 51](#_Toc27470)

[5.2 UI系统测试 52](#_Toc8399)

[5.3 游戏性能测试 54](#_Toc20381)

[5.4 本章小结 54](#_Toc4786)

[第6章 结论与展望 55](#_Toc25255)

[参考文献 56](#_Toc20504)

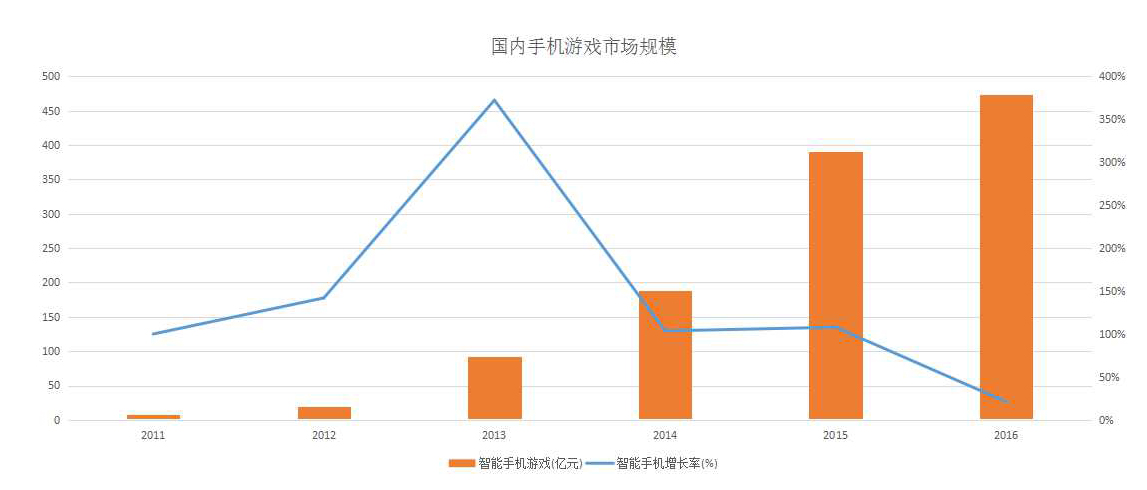
[作者简介 58](#_Toc13576)

[致 谢 59](#_Toc28386)

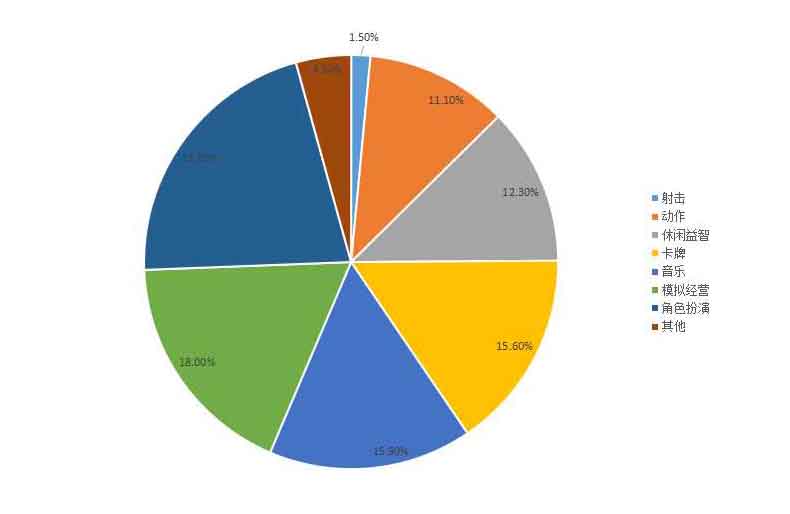
**第1章 绪 论**

**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，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掏出手机，方便快捷的使用app store或者google play安装所需软件，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。在人们日常娱乐方面，手机游戏发挥着举足轻重的作用。

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，很难说哪一种玩法潮流居于统治地位，每一类手游都在突破过去的旧模式，每一个突破都在今年掀起新的波澜。比如有人预言卡牌类游戏正在逐渐消亡[2]，但不得不说2016最赚钱游戏当属阴阳师莫属。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，产生了一种“我有智商靠搭配，血统爆发胜土豪”的错觉。同时在同类游戏中，画风，操作细节，声优，抽卡粘性均属上乘。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。综上所述的一系列优点，成为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，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，规模化的方向发展。

**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，年成长率为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，相信华为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，智能手机市场趋向饱和，新的手机用户增长越来越加缓慢，换机成为市场主流。

在智能手机发展过程中，同时带来手机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，以及国内的百度和360两家公司的应用市场分发数量快速增长，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基础上，添加了卡牌类角色战斗功能。相比桌游形式的卡牌类游戏，更易携带，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/)》横空出世，这款卡牌游戏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/)》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引擎为基础，针对卡牌类游戏的特点，在原有引擎的基础上对引擎进行优化，同时构造了新版的UI系统，并在其上构建了完整的卡牌游戏客户端。文章重点介绍了引擎的升级与改造，及UI系统的构造。文章组织结构如图1.3所示，每章安排如下:

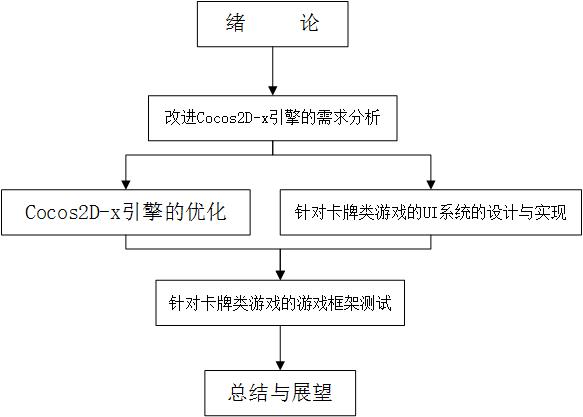


图1.3 文章组织结构

1. .绪论部分。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，对不同手游引擎做了简单的对比。
2. .改进Cocos2D-x引擎的需求分析。作者首先介绍了卡牌类游戏的需求，之后简要介绍了Cocos2D-x引擎的主要结构和其UI系统。结合卡牌类游戏特点，针对该引擎存在的问题，完成了对引擎优化和改进UI系统的需求分析。
3. .Cocos2D-x引擎的优化。把引擎的优化分为三个部分，详细介绍了优化的方法和具体实现。分别为渲染模块的优化，场景更新模块的优化，事件分发模块的优化。

第4章.针对卡牌类游戏的UI系统的设计与实现。总体概述UI系统各个模块的设计，之后详细介绍了各个模块的设计与实现。其中包括配置文件管理模块设计与实现，核心模块的设计与实现，系统级控件的设计与实现，游戏级控件的设计与实现，配套的UI编辑器。

第5章.针对卡牌类游戏的游戏框架测试。文章的测试部分，其中包括改良后的引擎测试，UI系统测试，以及在该框架基础上构建的手机游戏的性能测试。

第6章.总结与展望。总结了本文的主要工作，同时指出了工作的不足及进一步改进的方向。

**第2章 改进Cocos2D-x引擎的需求分析**

**2.1 卡牌类游戏的需求**

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，游戏不需要有太高帧率，一般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，卡牌类游戏属于大量耗费时间的游戏类型，玩家需要每天花费巨额的时间在做任务，收集卡片，升级卡片上。所以在游戏运行时，尽量减少电量的消耗，提高手机硬件的利用效率，减少不必要的资源加载。

手游开发中，不同手机有着不同尺寸，尤其手机和平板之间的尺寸比例和分辨率差别过大，如把手机游戏分为手机和平板两个版本开发，势必增加游戏开发的耗时。两个版本UI不同导致的代码不同，会造成开发上的不便，并且导致后期版本更新时出现问题。所以所有设备要使用同一个版本开发，该版本必须包含有效的屏幕适配机制。

在实际开发过程中，UI控件的布局和属性需要大量的调整。最好有可视化的方法实时修改UI控件的布局和属性，为此需要针对游戏引擎开发相应的UI编辑器，并且该UI编辑器可以读取游戏客户端使用的UI配置信息。该UI编辑器使用简单，可以由策划或美术负责修改UI控件布局和属性。这样可以降低引擎的使用难度，把主要时间花费在游戏玩法的优化上。

**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简单可靠。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**

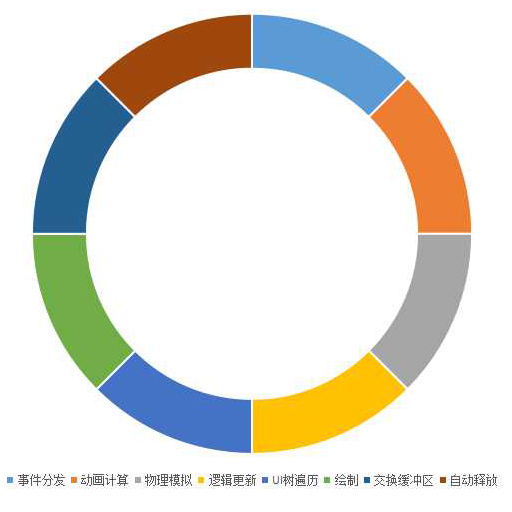
****

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。每一帧最开始时，分发上一帧触发的所有事件。在前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，PRIORITY\_SYSTEM取最小整数。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。

**2.2.4 Cocos2D-x的UI系统**

Cocos2D-x本身是一个巨大的代码库，整合了2D游戏常有的所有功能。当然自身也包含了简单的UI库，UI库基本结构如图2.3所示。

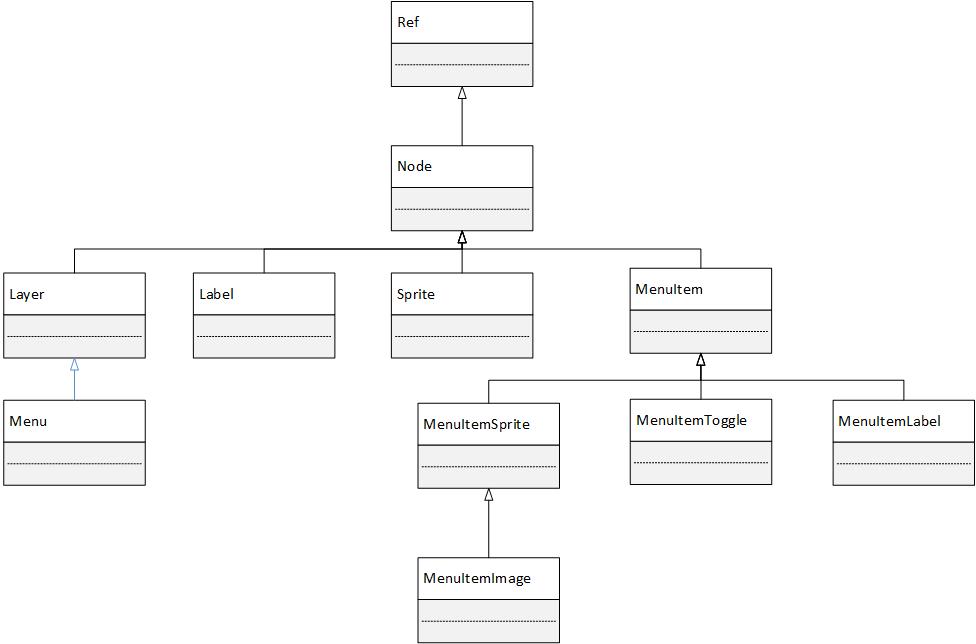


图2.3 Cocos2D-x的UI系统

在原生Cocos2D-x中，所有的UI控件继承于Node，Node提供了2D游戏中最基本的布局功能。比如说位置Position，锚点AnchorPoint，UI大小ContentSize等。Node类有一父类Ref，Ref类提供了Cocos2D-x的内存管理功能，所以所有UI控件统一受内存管理模块管理。引擎提供了最基本的UI控件，层(Layer),字体(Label),精灵(Sprite)，按钮(MenuItem)等。2D游戏中，场景中大部分的UI元素是由长方形纹理构成，每一个长方形纹理使用Sprite类加载。Cocos2D-x自身提供多样化的字体系统，根据使用方法的不同分为LabelTTF TTF文本标签控件，LabelAtlas自定义字体文本标签控件，LabelBMFont位图文本标签控件。或者可以使用他们统一的父类Label。在Label中选择想要的字体使用方式，之后根据具体使用方法配置相应的数据。游戏中的所有按钮有统一的父类MenuItem，根据使用按钮的样式不同分为MenuItemSprite，MenuItemImage，MenuItemLabel。其中MenuItemSprite，MenuItemImage由纹理构成，MenuItemLabel由字体构成。MenuItemToggle在内部拥有一个MenuItem数组，在触摸过程中可以展示不同的按钮状态。Menu类负责挂载MenuItem且只能挂载MenuItem，自身含有触发器，可以响应游戏中的触摸事件。

**2.3 引擎存在的问题**

Cocos2D-x为单线程引擎，手机硬件无法得到最大化的利用，在使用时难免有性能损耗，并且在调用GPU API时，CPU处于停等状态，大大浪费了时间。Cocos2D-x的场景管理模块过于简单，本身由单状态机配合栈管理场景，只有栈顶的场景才可以被显示。在实际使用时，很容易造成资源的重复加载，给管理这些不同场景中重复资源造成了极大的不便。

Cocos2D-x的UI系统功能简单，一般的小型游戏开发起来没有问题，但是对于大型游戏来说，未必太过简单了。并且Cocos2D-x原版引擎只是一个代码库，并不含有编辑器[10]。所以如果直接使用引擎，势必需要在游戏中以代码硬编译的方式存入UI位置，尺寸等信息，给开发者带来极大的不便，降低开发效率。

虽然现在Cocos2D-x官方提供了Cocos Studio和Creator等编辑器，但这些编辑器，只提供最基本的游戏UI组件。诸如菜单，进度条等大型控件还需要通过复杂的方式手动配置，扩展性不强，无法实现资源和代码的二次利用。且目前编译器功能不完全，并且存在Bug。

Cocos2D-x引擎的UI系统，只能通过位置和锚点等基础信息来完成UI的布局。这种方法不同于其他应用的布置方法，极容易出现错误。

在分析了卡牌类游戏的需求和引擎存在的问题以后，作者总结了以下几点需求:

1. 提高CPU资源利用率

为了降低手机电池的消耗，需要提高CPU资源的利用率。可以通过渲染线程和逻辑线程分离的方式，充分利用手机多核CPU。具体优化于3.1节所示。

1. 优化场景资源管理流程

减少场景过渡时，重复的资源加载，是优化场景资源管理的主要目标。在场景的切换中，容易造成资源不必要的卸载和加载，所以以双层状态机的思想改良了Cocos2D-x引擎的场景管理模块。具体优化于3.2节所示。

3.构建新型的UI布局方式

依靠位置和锚点进行UI布局的方式，在实际使用时存在大量的不便。所以需要构建一种新式的UI布局方式，通过设定当前UI控件和他的父亲UI控件边界的距离的方式来完成UI布局。具体设计与实现于第4章所示。

4.UI控件布局和代码的分离

UI控件布局尽量不要直接写在代码中。写在代码中，每次修改都需要重新编译代码，所以控件无法实时的显示。并且以代码的形式配置UI控件，给前期配置和修改造成了诸多不便。所以构建新版的UI系统，开发布局UI控件的UI编辑器，并且以特殊的配置文件作为UI编辑器和游戏客户端的中间件。具体设计与实现于第4章所示。

1. 提供UI系统更强的可扩展性

对于卡牌类游戏，Cocos2D-x原本的UI无法满足所有的UI需求。所以在原版UI控件的基础上，重新开发出新的UI控件，并且提供了简单的接口，方便在后期需要时添加更新的UI控件。具体设计与实现于第4章所示。

**2.4 本章小结**

本章首先阐述了卡牌类游戏的需求，接下来介绍了Cocos2D-x引擎相关的知识，包括引擎结构，引擎工作流和引擎的UI系统。针对卡牌类游戏需求和Cocos2D-x引擎特点，分析出引擎存在的问题，得出该引擎改进的方向。

**第3章 Cocos2D-x引擎的优化**

**3.1 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，增多手机CPU核心数量[11-12]。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核心频率降低，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，刷新过程中耗费了大量的运算资源，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可以看成一个外部硬件，渲染的过程就是CPU不停的给GPU发送各种命令[13]。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[14]。之后这个buffer会被清空，以便接受新的命令。GPU渲染管线对于游戏开发是黑盒的，不能修改其中的代码。但在渲染管线外边，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，以节省手机电量消耗为目的，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。

**3.1.1 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在跑，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[15]。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到响帧率的高低。早期游戏中，因为平台固定，只需要适当调整处理的信息量，便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，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。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，如果无法在规定时间内完成任务，睡眠时间会变成负值，程序便会出现错误。

3.动态间隔。

在游戏循环过程中，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，使用实际间隔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都不会出现问题，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不同可能会导致float运算结果不同，最终导致上层显示结果出现了偏差。

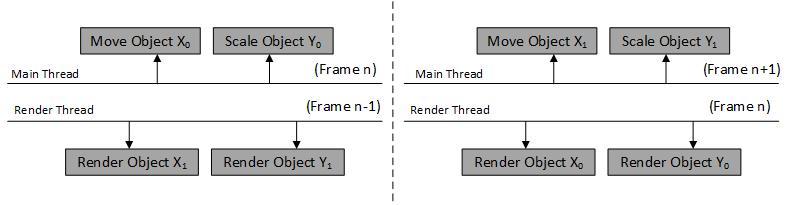
4.固定更新间隔，动态渲染间隔。

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。

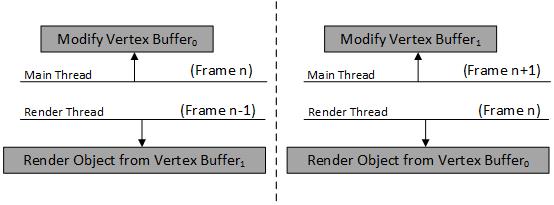
**3.1.2 几种多线程渲染方式**

1. OGRE

OGRE分为两种架构，一种为middle-level multithread，另一种为low-level multithread[16]。

图3.1 middle-level multithread

如图3.1所示，每个实体被分配成两份相同的单元，主线程和渲染线程交替处理同一份实体两份相同的单元，并在帧的尾部进行同步。这种解决方案中CPU逻辑计算不会影响渲染速度，但较差的GPU渲染速度会影响逻辑计算。并且两份实体的维护也相对复杂，多核心CPU的情况下，效果没有很大的提高。

图3.2 Low-level multithread

如图3.2所示，Low-level multithread将D3D对象复制成两份相同的单元，同样是在帧的尾部进行同步。该种方法的优缺点与前述方法基本相同。

两种多线程渲染的解决方案，均是在引擎层做文章。用户使用时，不需要注意多线程渲染实现时过多的细节，减少了开发的难度。

2.idsoftware

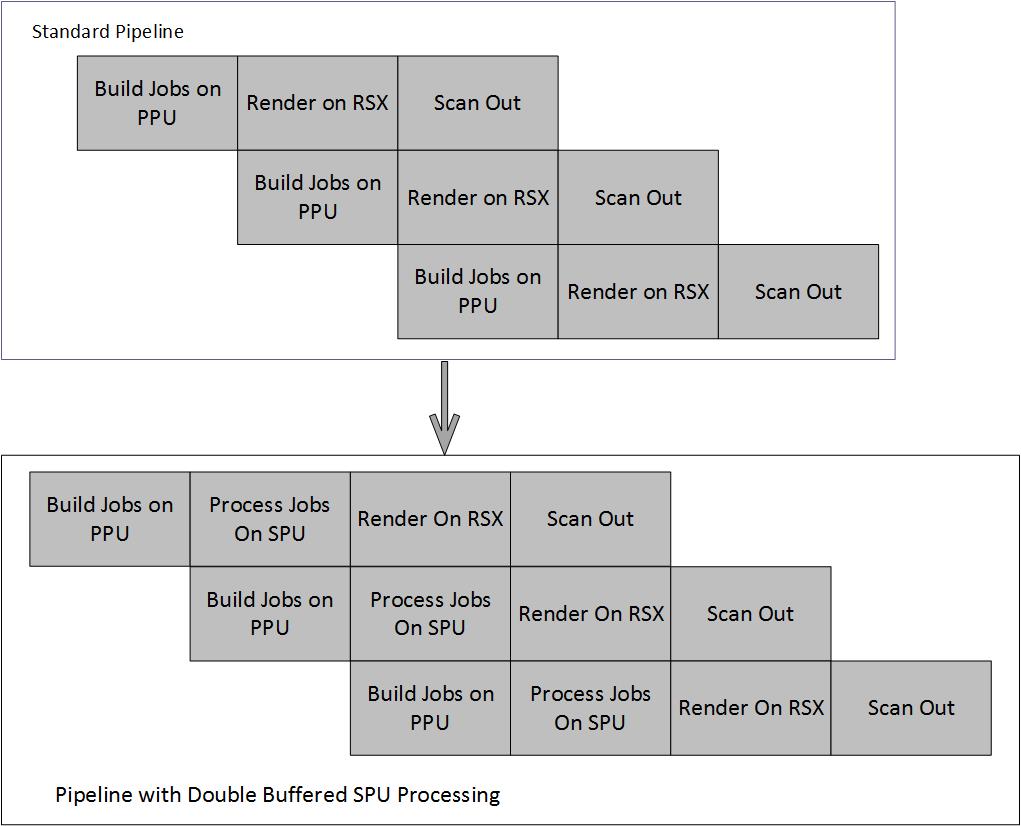


图3.3 PS4引擎架构

PS4的引擎结构与PC有较大的不同，PS4由PPU,SPU,RSX组成。其中SPU为Cell芯片的8个协处理器，拥有强大的并行能力。PS4游戏开发者把各项任务分配到不同单元中处理，其中SPU中负责骨胳动画，形变动画，顶点和索引缓冲的压缩。PPU负责物理计算,RSX负责渲染的具体工作，以此来最大限度的利用PS4的机能。

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，通过命令行-onethread来控制多线程渲染的关闭[17]。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[18-19]，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二帧的游戏逻辑线程修改了渲染到第一帧资源，造成资源错误的情况[20]。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。

**3.1.3 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，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。

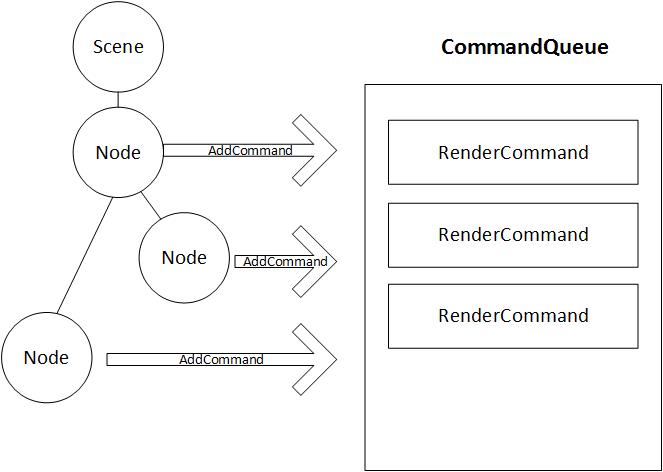


图3.4 Cocos2D-x的UI树遍历

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。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。

**3.1.4 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,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。

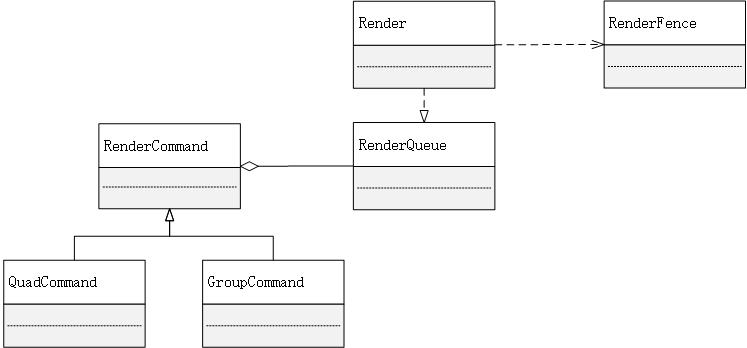
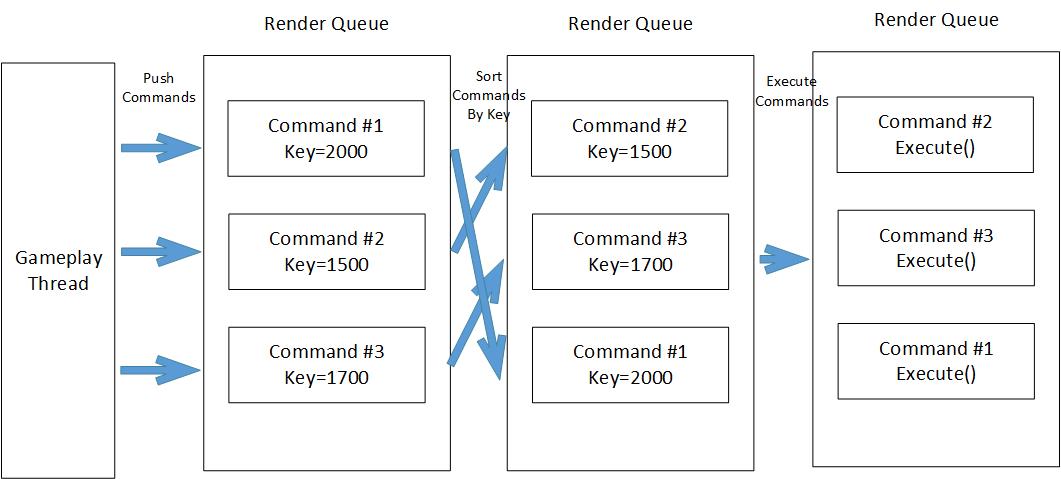


图3.5 渲染模块类图

图3.6 渲染队列根据渲染命令自身所生成的Key排序

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。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。渲染线程在绘制命令之前，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3.7 游戏线程工作流



图3.8 渲染线程工作流

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，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时使用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。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，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。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。

**3.2 场景更新模块的优化**

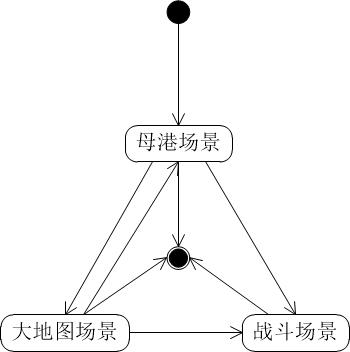
****

图3.9 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[21]。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，还需要前一个场景。这种情况中，没有必要释放当前场景的资源，所以把当前的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，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[22]。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，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，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。

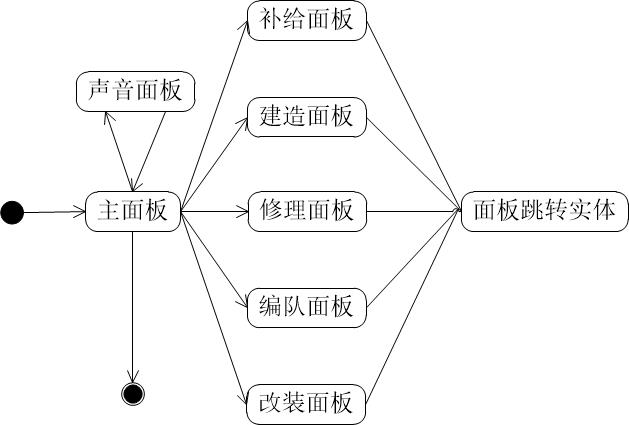
****

图3.10 某一场景下面板状态机的切换

**3.2.1 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，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，但假如玩家如厕指数上升到一定水平，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。

分层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，可以在定义更多的状态层以降低跳转链接数。

**3.2.2 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中包含一个场景以及所有UI控件，如按钮，图片，精灵等[23]。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。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。

**3.2.3 改良后场景更新模块**

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[24]。

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类如果想提供Panel类的进入和退出操作，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，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，为了达到设计目的，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，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下面的其他UI控件中。同时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。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，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。

**3.3 事件分发模块的优化**

**3.3.1 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[25-26]。订阅者模式可以减少软件内模块间的耦合，同时保持模块之间高效通信。游戏过程中，需要随时处理外界的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。并且也可以定义引擎独有事件。

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，分发过程中根据事件源找到相应的listenerID。要是用订阅者首先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，或者所绑定Node在UI渲染树中的位置去判断分发顺序。

**3.3.2 事件分发模块优化**

在游戏的过程中，有时需要暂停某一类事件的分发。比如在进行场景切换时，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。

**3.4 本章小结**

本章是本课题的工作重点之一，针对第二章所提出的前两点引擎改进需求，对Cocos2D-x引擎进行优化。引擎的优化分为三个部分，渲染模块的优化，场景更新模块的优化，事件分发模块的优化。每小节首先提及了该模块优化所使用的技术，其中包括多线程渲染，分层状态机和几种游戏循环同步方式。之后介绍了Cocos2D-x引擎中的该模块，最后详细描述了该模块优化及实现过程。

**第4章 针对卡牌类游戏的UI系统设计与实现**

**4.1 新版UI系统总体设计**

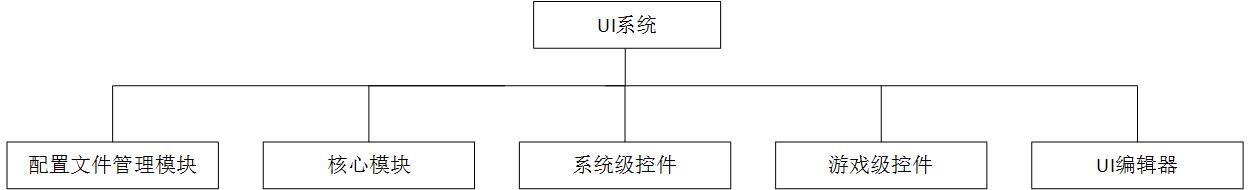
作者总结了原先Cocos2D-x的UI系统，以及引擎改进需求。最后设计出针对卡牌类游戏的新版UI系统，新版UI系统的整体结构如图4.1示。

图4.1 新版UI系统

新版UI系统分为5个大部分，配置文件管理模块，核心模块，系统级控件，游戏级控件和相对应的UI编辑器。

1. 配置文件管理模块。

在开发过程中，需要设计一套配置文件以方便UI控件的布局。使用配置文件布局UI可以大大减少重编译客户端代码的次数。当修改UI控件时，不需要直接修改UI关联代码，只需要修改对应配置文件，修改完毕后，重新加载配置文件即可。该套配置文件必须可同时被UI编辑器和游戏客户端加载。客户端读取配置文件以显示特定界面的UI控件。UI编译器读取配置文件以显示该配置文件在客户端中加载后的效果，并且可以在实时的条件下修改配置文件。配置文件可以提前加载到内存中，但仅当配置文件被使用时，系统才会加载该配置文件相关联的资源。

配置文件管理模块通过配置文件完成UI控件的加载，UI控件的显示,UI控件的释放。针对不同UI控件，还需要设置相应的初始化属性值，比如说控件位置，控件大小，控件是否缩放，控件是否响应触摸事件等。甚至可以通过修改属性值的方法，决定是否使用预先挂载在特定控件上的逻辑代码或shader。诸如此类的属性值也是配置在配置文件中，配置文件管理模块负责管理这些属性值的加载。

在UI管理过程中，也需要重新构建个UI树用来管理这些UI控件，每一个UI控件都是一个节点，与Cocos2D-x原先的UI渲染树同步更新，保证UI控件的遍历不会在中途出现问题。UI控件的加载需要做到最简单，使用统一的API完成对UI控件的加载，其余工作全部由UI系统自行完成。

1. 核心模块。

核心模块是新版UI系统最重要模块，其中包括了UI控件创建，布局管理等。

重新构建了一种UI控件布局管理方法，不再使用位置和锚点的方式，而使用当前控件距离父控件距离的方式管理UI控件的位置。此方法相较于位置和锚点的方式，有利于UI控件在不同机型中的适配。

同时核心模块包含有Canvas类的设计，Canvas类相当于原版UI系统中的Node类，具体设计与实现于4.3节所示。

3.系统级控件。

针对手机游戏，在Cocos2D-x原有控件的基础上，重新开发一套新的UI控件。这套控件为游戏底层控件，需要满足Cocos2D-x所能实现的各方面功能。

为此设计了如表4.1所示的几种基本的系统控件。

|  |  |  |
| --- | --- | --- |
| 类名 | 控件名 | 功能描述 |
| Image | 静态图片 | 以一张静态图片方式显示 |
| Button | 按钮 | 游戏中的按钮实体，点击按钮可以触发相应的工作，包括三种状态。未触摸，触摸开始，触摸结束，针对这三种状态可以显示不同的静态图片或文字 |
| Label | 文字 | 游戏中的字体控件，封装了Cocos2D-x的字体控件，在配置文件中可以调整不同的配置，达到显示不同信息的目的 |
| Toggle | 按钮触发器 | 由多个按钮组成，在不同按钮状态之间可以进行切换 |
| ScaleImage | 九宫格图片 | 九宫格图片控件，减少因图片过渡拉伸而造成毛边的可能性。适合按钮所挂载的控件 |
| InputEdit | 文本输入框 | 文字信息输入 |

表4.1 系统级控件介绍

4.游戏级控件。

随着游戏的开发，引擎需要越来越多的UI控件，大量的游戏UI控件是卡牌类型游戏所必须的，如在配置文件中通过系统级UI控件的方式去配置他们，势必需要大量的时间。所以为了增加UI系统的扩展性，减少开发难度，我们对所开发的游戏的UI控件进行总结，统计出常用的UI控件，比如菜单实体，倒计时器等，如表4.2所示。在UI系统和UI编辑器中添加对应的代码，使用系统级控件去构建。使用时可以直接通过对应的配置文件信息去管理。

|  |  |  |
| --- | --- | --- |
| 类名 | 控件名 | 功能描述 |
| 菜单实体 | MenuEntity | 游戏中的菜单界面，负责显示游戏中物品列表 |
| 面板跳转实体 | LayerSelecter | 不同面板之间的跳转按钮 |
| 列表切换按钮 | ListButton | 多个按钮排成一排的切换控件，同时只有一个按钮可以被选中 |
| 倒计时器 | TimerEntity | 可以提供倒计时功能的时间显示控件 |

表4.2 游戏级控件介绍

1. UI编辑器。

UI编辑器负责编辑和管理配置文件。开发者把配置文件加载入UI编辑器，通过可视化的方式查看和修改配置文件。

**4.2 配置文件管理模块**

配置文件选用XML[27]。为了保证游戏中的配置文件加载速度，游戏中使用了RapidXml加载XML。RapidXml相较于其他XML解析库。他的速度能快十倍以上，并且设计简单，在实际使用时，并不占用过多的内存空间，如图4.2所示，此为单个控件配置信息，一个控件的配置信息，由控件标识，控件属性和它所挂载的其他控件组成。

如图4.1所示，该图为游戏中新版UI系统的设计图。XMLMgr负责统一管理配置文件，XMLParser封装了解析XML常用的API。因为XML占内存容量小，在游戏初始化过程中，便加载所有的配置文件，统一放在XMLMgr的内存池中，每一个配置文件为一个XMLData，以加载过程中的路径作为索引ID。

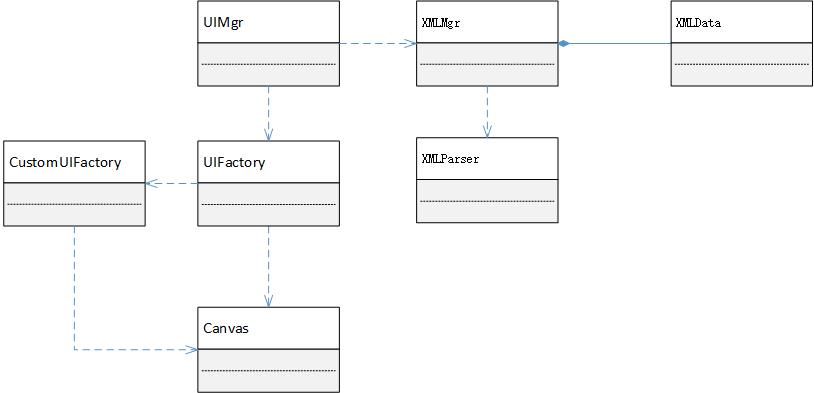


图4.1 新版UI系统类图

配置文件管理模块为了加大UI控件的扩展性，UI控件的创建使用了工厂模式。其中所有的系统级控件使用UIFactory类创建，所有的游戏级控件使用CustomUIFactory创建。工厂类根据从XML中解析出来的文本标签来判断该UI控件属于何种类型，运行相应组件的构造和初始化函数。

其中暴露给开发者的是UIMgr类。UIMgr负责加载UI控件，负责管理所有的UI控件。UIMgr根据配置文件中设定的UI控件的id属性找到相应的UI控件。在加载场景时，通过使用该类的API进行加载操作，下面描述加载的全过程。

如图4.3所示，该图为UI加载过程的流程图。首先在XMLMgr中调用加载函数API，遍历XMLMgr内存池，如果在池中发现了对应的XMLData，返回当前XMLData，如果池中不存在，则根据当前路径从文件中加载XMLData。判断当前节点节点是否解析成功。如果不成功直接返回。如当前节点解析成功，解析出当前节点的标识符，把当前节点标识符以字符串的形式传入工厂类。工厂类分为UIFactory和CustomUIFactory，UIFactory负责系统级控件的构造，CustomUIFactory负责游戏级控件的构造，这两个类通过责任链设计模式连接[28-29]。控件创建成功后，解析出当前文本标签所带有的所有属性名和属性值，以字符串队列的形式传入控件的Parse函数中，在Parse函数中分析两个队列值并且完成字符串的解析，调用适合的属性修改函数。完整这步之后，一个独立控件的加载完成，但一个UI框架是由一个UI树组成，所以还需要以相同方式处理配置文件的下一个节点，直到整个UI树解析完毕[30]。

IMG_256

图4.2 配置文件控件配置信息

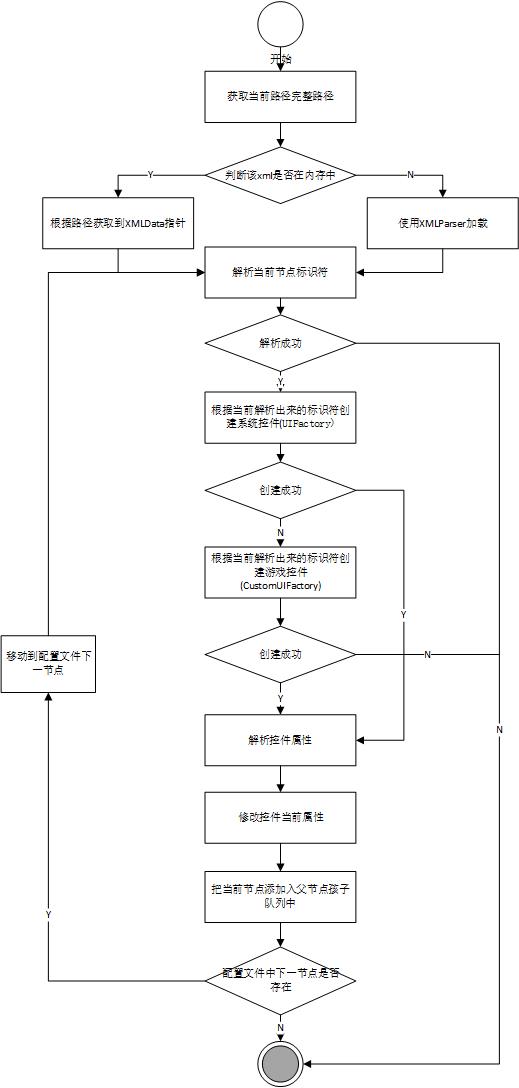


图4.3 UI控件加载过程流程图

**4.3 核心模块**

为了方便UI系统的布局，简化开发难度，重新设计了UI系统中布局的方式。我们借鉴了其他UI系统中的布局方式，提出了新式的UI布局方法。在新版的UI系统中，定义了四个变量Left,Right,Top，Bottom。代表着当前UI控件距离父UI控件边界的距离。这种通过设定UI控件之间距离的布局方式，在不同屏幕分辨率手机的适配过程中，减少了UI冲突的可能性。为了照顾实际使用时的特殊情况，配置文件中仍然保留着以位置和锚点信息定位UI位置的接口。

Canvas为新版UI系统中的基础类，相当于原版Cocos2D-x中的Node。Canvas直接继承于Node，提供了Node基础上进一步的封装。同时Canvas内部保存着一颗独立的UI树，与原本UI树为不同的两颗树,但两棵树保持同步。

Canvas可以通过配置文件配置大量信息，其属性表如表4.3所示。

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| Id | String | 控件名称索引 |
| Width | Float | 控件宽度 |
| Height | Float | 控件高度 |
| ContentSize | Bool | 是否使用宽度和高度约束控件大小 |
| Position | Vec2 | 控件位置 |
| Z | Float | 控件的GlobalOrder大小 |
| Left | Float | 控件距离父控件左边界距离 |
| Right | Float | 控件距离父控件右边界距离 |
| Top | Float | 控件距离父控件上边界距离 |
| Bottom | Float | 控件距离父控件下边界距离 |
| Anchor | Vec2 | 锚点位置 |
| IgnoreAnchor | Bool | 是否使用锚点，不使用情况下使用默认锚点为(0.5,0.5) |
| Visible | Bool | 控件初始化时是否可见 |
| Touchabled | Bool | 控件是否响应触摸事件 |
| Scale | Vec2 | 控件缩放尺寸比例 |
| SwallowTouch | bool | 控件的子控件是否响应触摸事件 |
| TouchExtend | Int | 触摸区域是否向外延展 |

表4.3 Canvas属性表

(1)原先Cocos2D-x变量

Position指代UI控件的位置，Anchor和指代锚点的位置， IgnoreAnchor指代是否忽略锚点属性，如果为True则忽略，锚点选用默认的(0.5,0.5)。这套方法是原先UI系统中布局UI的方式，如果在配置文件中， Left,Right,Bottom,Top任意一个属性存在，则忽略原先Position,Anchor的配置。

(2)ID

通过在配置文件中提前设置ID的方式，作为UI控件的映射。如在UI系统中想要寻找到某一控件，无需提前保持该控件的指针，直接调用UIMgr中的FindUIByID函数，该函数会遍历当前UI树，并且匹配每个UI控件的ID值。如果发现某个UI控件的ID值与想要找的控件ID值相同，则返回该控件。

(3)Left,Right,Top,Bottom

该四个变量为新版UI系统屏幕适配的核心所在，Left指代当前控件距离父控件左边界距离，Right指代当前控件距离父控件右边界距离，Top指代当前控件距离父控件上边界距离，Bottom指代当前控件距离父控件下边界距离。如果当前UI控件是整个UI树最顶层控件，则四个属性分别指代屏幕的四个边界。

(4)Width,Height,ContentSize

Width和Height,来控制当前控件大小，如果Width,Height设定的值比当前控件的边界大，则当前控件做居中处理，剩下部分做透明处理。使用空的Canvas配合Width和Height可以制作一个空白控件，方便他的子控件布局。ContentSize属性负责控制Width和Height属性是否有效，默认为True。如果设成False，则Width和Height的大小不会影响控件的大小。

(5)Touchabled,SwallowTouch,TouchExtend

Touchabled,SwallowTouch,TouchExtend负责控件的触摸管理，Canvas相较于Node，Canvas自身携带触摸事件的监听器，能够监听触摸事件的分发，UI控件分发的顺序为渲染UI控件的相反顺序。Touchabled负责开关触摸响应，默认Touchabled为False，如果为True则该控件接收触摸事件的分发。在实际使用时，有可能多个UI控件会叠加在一起，通过设定SwallowTouch，阻止触摸事件分发到当前UI控件下面的那个控件，阻止触摸事件的进一步传递。TouchExtend是为了延伸触摸区域大小，触摸区域可比控件实际渲染区域更大，方便了小控件的触摸响应。但实际使用时，要注意UI控件延伸区域重叠问题。

**4.4 系统级控件**

**4.4.1 Image**

Image为游戏中最基本的控件，提供了一张静态图片的显示。

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| SourcePath | String | 图片资源路径 |
| Rect | Vec2 | 图片显示范围 |

表4.4 Image属性表

(1).SourcePath

SourcePath为图片资源加载路径，在UI配置文件加载时，并不进行图片纹理资源的加载[31]。而是等待加载该UI控件时，使用Cocos2D-x默认的资源管理类统一加载，所以该纹理资源的内存管理不由XMLMgr负责，由资源管理类负责。

(2)Rect

在实际使用过程中，有可能需要对图片进行裁剪。对此使用Rect来设定图片显示范围。

**4.4.2 Button**

Button为游戏中按钮控件，是游戏中最常用的控件之一。按钮控件一共有三种状态，未被触摸状态(Up)，被触摸状态（Down）,以及移出响应区域状态(Disabled)。三种状态可以挂载三种不同的控件，当处于特定状态时显示当前控件的状态。

Button由三个Canvas组成，分别指代三种不同状态下的Canvas。Button重构了OnTouchBegin和OnTouchEnded来达到设计的功能。实际使用中，首先通过FindUIByID找到该Button，之后使用SetCallBack为该Button添加特定的回调函数，根据按钮触摸情况，分发事件至回调函数中。

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| ScaleMode | Bool | 是否开启按钮缩放模式，如果开启，点击按钮之后，按钮会缩放固定倍率 |
| ScaleModeFactor | Float | 按钮缩放倍率，需要提起开启按钮缩放模式 |
| OriginScale | Float | 按钮原始倍率 |
| Up | Value | 按钮未被触摸时，所显示的控件，可以为字体控件或者图片控件等 |
| Down | Value | 按钮被触摸时，所显示的控件，可以为字体控件或者图片控件等 |
| Disabled | Value | 按钮触摸后，移出响应区域，所显示的控件，可以为字体控件或者图片控件等 |

表4.5 Button属性表

1. ScaleMode,ScaleModeFactor,OriginScale

使用ScaleMode来开启按钮缩放模式。当ScaleMode为True时，触摸按钮以后，按钮会缩放固定倍数比例。缩放倍率由ScaleModeFactor属性设置，而原有控件缩放倍率为OrginScale。

(2)Up,Down,Disabled

指代按钮三种状态下所显示的控件，该控件可以为文字，静态图片等。Up，Down,Disabled所指代的控件挂载在按钮配置文件对应节点的子节点中，当解析一个按钮控件，需要查看他的子节点是否含有Up，Down，Disabled标签，三个标签所指代的控件为Button控件三种不同的状态。

**4.4.3 Label**

Label是游戏中基本的组件，指代游戏中的字体控件。本Label是在原先UI系统的Label上封装一层，并且提供了利用配置文件修改Label基本属性的功能。

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| Text | String | 字体控件默认字符串 |
| FontName | String | 字体控件字体名称 |
| FontSize | Int | 字体控件字体大小 |
| FontColor | Vec4 | 字体控件字体颜色 |
| Valign | String | 字体控件垂直对齐方式，可以上对齐或是下对齐 |
| HAlign | String | 字体控件水平对齐方式，可以左对齐或是右对齐 |
| OutLine | Int | 字体控件外发光尺寸，默认为0 |
| OutLineColor | Vec4 | 字体控件外发光颜色，默认为白色 |

表4.6 Label属性表

(1)Text

字体控件默认的字符串，最后显示在窗口中的静态字体。

(2)FontName,FontSize,FontColor

字符控件的基础属性设置。

(3)Valign,HAlign

字符控件可以指定其Width和Height的大小，在控件中字体无法填满控件的情况下,Valign,Halign两个属性可以设置如何对齐。Valign有两个选项Top和Bottom，分别代表了对齐到上边界和对齐到下边界。Halign有两个选项Left和Right，分别代表了对齐到左边界和对齐到右边界。

(4)OutLine,OutLineColor

指代字符控件是否开启外描边以及外描边的颜色。

**4.4.4 Toggle**

Toggle的功能类似于原先UI系统中的MenuItemToggle。Toggle在内部拥有一个Button数组，在内部负责展现不同的按钮状态，通过触摸事件完成不同状态之间切换,在切换过程中可以触发相应状态的回调函数。

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| Size | Int | 所挂载的按钮大小 |
| Button | Button\* | 所挂载的按钮 |

表4.7 Toggle属性表

Size指代所携带的Button数量。之后根据Size数量解析固定长度Toggle子节点，该子节点为Button类型控件，指代Toggle控件之间的各个状态。

**4.4.5 ScaleImage**

直接对静态图片进行缩放，会导致对整个纹理进行拉伸，放大尺寸过大时会造成毛边等效果。针对一些UI元素，如按钮或是进度条，通常只需要对纹理的一部分进行拉伸。比如说按钮，在放大时，只需要对中间纯色区域拉伸，Cocos2D-x自带九宫格图片解决了这一问题。ScaleImage就是对九宫格图片的封装。

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| SourcePath | String | 所挂载资源的路径 |
| CapInsets | Vec4 | 划分九宫格图片区域的Rect |

表4.8 ScaleImage属性表

SourcePath为所挂载资源的路径，原理跟Image一样。CapInsets为划分九宫区域的Rect大小，其中Rect中元素以‘，’隔开。

**4.4.6 InputEdit**

InputEdit为游戏中的输入框控件，在登录游戏，修改编队名称功能时都需要输入框控件。输入框控件自带监听器，可以响应触摸事件的分发。

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| DefaultText | String | 输入框控件提示字符串 |
| Text | String | 输入框控件输入字符串 |
| DefaultColor | Vec4 | 提示字符串颜色 |
| FontName | String | 输入框控件字体名称 |
| FontSize | Int | 输入框控件字体大小 |
| FontColor | Vec4 | 字体控件字体颜色 |
| MaxNum | Int | 输入框控件可输入的最大字符数 |
| PasswordOpen | Bool | 密码框开关 |

表4.9 InputEdit属性表

(1)DefaultText,DefaultColor,Text

在字符框为空时，字符框中可有提示字符串。指示玩家该字符框需要输入字符串的具体要求，提供交互性良好的界面。DefaultText指代提示字符串，DefaultColor指代提示字符串颜色，Text指代输入字符串，其他系统模块取这个值以完成具体需求。

(2)FontName，FontSize,FontColor

字符串最基本的设置信息。

(3)MaxNum

在输入过程中，对输入字符串数量有明确的限制， 比如说用户名不能超过某一固定字数，如果超过固定字符，则禁止继续输入。MaxNum这一属性指定了当前字符框所能输入的最大字符数量。

(4)PasswordOpen

对于密码这类输入，要以‘\*’或‘。’来代替明文显示，PasswordOpen用来切换明暗文显示。如果为True则为暗文显示，为False则为明文显示。

**4.5 游戏级控件**

**4.5.1 菜单控件(ListEntity)**

菜单控件用于在游戏中选择和修改物品，如图4.4所示。游戏中一共有三个系统需要用到菜单控件，分别为编队系统，修理系统，建造系统。菜单可以实现移入移出功能，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。菜单用于显示所表示物品的基本信息，并提供根据特定物品属性对物品排序的功能。

菜单类为ListEntity，为了分离各部分功能，实现模块间解耦，菜单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。ListEntity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中每一个物品以简要信息的形式展示，称作一个Cell。一个物品占据一行，每一行相当于一个小的UI控件，可以用SetCellCallFunc的方式指定每一行的UI控件。菜单中保存着队列，存储着所有物品的指针，队列中指针的顺序既为控件在菜单中显示的顺序,控件的排序方式需要提前使用SetSortCallfunc提前设定，使用时通过Menu上的Toggle切换。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Cell。



图4.4 菜单模块界面

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| Gap | float | 菜单单元之间间隔，默认为0 |
| Align | String | 对齐方式，提供左对齐和右对齐 |
| PageCapacity | Int | 每一页最大容量，需要提前开启分页模式 |
| Paging | Bool | 是否开启分页模式 |

表4.10 ListEntity属性表

(1)Gap,Align

Gap指代每个Cell之间的间距，默认为0。Align指代菜单中Cell中对齐方式。一共有两种对齐方式可以选择，左对齐和右对齐。

(2)PageCapacity,Paging

如果该菜单所含物品数量庞大，单一界面未必能全部显示，需要使用分页模式。设置Paging为TRUE，同时由PageCapacity设定一页中Cell大小。开启分页模式，分页菜单由ListButton实现。

**4.5.2 面板跳转实体(LayerSelecter)**



图4.5 面板跳转实体界面

在游戏的菜单场景中，为了方便不同面板的跳转，提供了一个面板跳转实体。该实体负责在不同面板间进行切换，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，如图4.5所示。

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。在配置SelecterUnit时，需要提前指定该Unit指代的面板。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 列表切换按钮(ListButton)**



图4.6 编队切换按钮界面

列表切换按钮是一系列按钮以一定序列间隔排布的控件。通过在不同按钮之间切换的方式切换状态，同时只能有一个按钮被选中。如图4.6所示，是游戏中的编队列表切换按钮。按钮的最大数量和初始按钮位置可以提前设置。切换列表所调用的回调函数通过SetCallTouchCallfunc提前指定。

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| MaxButtonSize | Int | 最大可见的按钮数量 |
| StartButtonNum | Int | 初始按钮位置 |
| EnableButtonSize | Int | 已经开启的按钮数量 |
| Up | Value | 编队按钮单元未被选中时，所显示的控件 |
| Down | Value | 编队按钮单元被选中时，所显示的控件 |
| Disable | Value | 编队按钮单元未被开启时，所显示的控件 |

表4.11 ListButton属性表

(1)MaxButtonSize，StartButtonNum，EnableButtonSize

MaxButtonSize指代列表序列中最大可见按钮的数量，EnableButtonSize指代序列中已经开启的按钮数量，未开启的按钮以Disable状态显示。StartButtonNum指代列表切换按钮初始化时，当前列表中触发按钮的位置。该值不能大于已开启的按钮的数量。

(2)Up，Down，Disable

Up，Down，Disable是按钮的三种状态，Up为未触发状态，Down为触发状态，Disable为未开启状态。

**4.5.4 倒计时器(TimerEntity)**



图4.7 倒计时器页面

在游戏的修理系统和建造系统中，需要倒计时控件，倒计时控件会在时间到时，回调提前注册好的回调函数。通过SetCallFunc这个函数提前指定回调函数。

为了防止游戏切入后台再切回以后时间显示出现问题，提前计算出倒计时完成的时间戳，每次刷新时通过当前时间戳减倒计时完成时间戳来获得剩余时间。游戏中扩展了时间管理类，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修改了系统时间而改变。

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| StartTime | String | 倒计时间 |
| Multiple | Float | 和现实时间的倍率，默认为1.0 |

表4.12 TimeEntity属性表

(1)StartTime

倒计时间以字符串的形式指定，七小时三十分钟剩余时间以”07:30:00”的形式配置。读取到该字符串以后，通过转换函数转换为时间戳的形式。

(2)Multiple

Multiple属性指代与现实时间流失的倍率，如multiple值为2.0，则现实时间每倒计时2秒钟，倒计时器只走了1秒。

**4.6 UI编辑器**

对于一个完整的UI系统，需要提供一个专门的UI编辑器。UI编辑器对于一个UI系统是很重要的一部分，首先配置文件通常不采用手动的形式编写，手动编写容易出现错误。其次如手动形式编写配置文件，势必给检验UI编写是否正确带来了麻烦，只能通过运行游戏的方式检验UI是否正确，但并不是每个界面的触发都是简单的。在实际使用过程中，费时费力，大量消耗了开发时间。所以需要为UI系统提供一个独立的UI编辑器。

**4.6.1 UI编辑器的需求**

在原有客户端代码的基础上，使用Qt开发出了UI编辑器。使用原有客户端的代码，是为了最大限度减少编辑器与客户端中UI控件显示差异，在开发编译器过程可以不涉及到过多的代码重构，在日后的开发中，客户端新增控件也可以很容易的添加至编辑器中。我们给UI编辑器提了以下需求：

1. 加载配置文件所指代的UI界面，并且提供修改配置文件的功能

加载配置文件所指导的UI界面，并且编辑器中的UI界面和游戏中UI界面布局完全相同。在编译器中可以新增游戏控件，并且修改当前控件的属性。

(2)配置文件修改参数时能实现实时更新

每当新添加控件，或者修改了控件属性，编译器上的UI元素可以实时刷新，方便使用，不用重新载入配置文件。

**4.6.2 UI编辑器的完成效果**

当前编辑器已经实现了大部分基本功能。包括打开配置文件，保存配置文件，查看配置文件所有控件，显示当前控件的所有属性，新建控件，修改控件属性，删除控件等功能。



图4.8 UI编辑器界面

**4.7 本章小结**

本章是本课题的工作重点之一，针对第二章所提出的后三点引擎改进需求，构建了针对卡牌类游戏的新版UI系统。首先总体概述了新版UI系统的设计，把UI系统划分为五个模块，配置文件管理模块，核心模块，系统级控件，游戏级控件，UI编辑器，其中系统级控件和游戏级控件由十种不同的控件组成。之后分别介绍了这五个模块的设计与实现。

**第5章 针对卡牌类游戏的游戏框架测试**

**5.1 改良后的引擎测试**

在游戏系统测试之前，先对改良之后的多线程渲染版本的Cocos2D-x引擎进行测试。查看引擎功能是否完善，并且从渲染帧率和CPU利用率上与原本单线程Cocos2D-x引擎做了对比。

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5.1所示，测试结果如表5.2所示。

|  |  |
| --- | --- |
| 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5.1 引擎测试环境配置表

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

表5.2 改良前后引擎在不同drawcall下渲染时间对比

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CPU利用率上，如表5.3所示，多线程渲染引擎的CPU利用率要比单线程引擎CPU利用率高。

|  |  |  |
| --- | --- | --- |
| drawcall数量 | 单线程引擎CPU使用率 | 多线程引擎CPU使用率 |
| 0 | 32.48% | 41.20% |
| 1320 | 36.79% | 46.94% |
| 2376 | 42.88% | 55.21% |
| 5021 | 50.36% | 59.11% |

表5.3 改良前后引擎在不同drawcall下CPU利用率对比

所以得出结论，在实际应用上，多线程渲染引擎无论是在渲染帧率上以及CPU利用率上，相对于单线程引擎都有着很大的优势。

**5.2 UI系统测试**

UI系统测试是为了测试UI系统功能是否运转正常，以及使用新版UI系统开发的复杂度。分别使用新版UI系统和原版引擎构建相同场景，计算开发时间，以及代码行数，选用所开发游戏的建造面板进行开发。测试时选用不同系统多种机型进行测试，如表5.4所示。用以测试UI系统在不同机型下，屏幕适配是否正确。

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |
| 屏幕分辨率 | 1136\*640 | 2048\*1536 | 1920\*1080 |

表5.4 测试用机配置一览

|  |  |  |
| --- | --- | --- |
| 实现方式 | 本人开发时间(小时) | 代码行数 |
| 原版引擎开发 | 162 | 9000 |
| 新版UI系统 | 70 | 5000 |

表5.5 新版UI系统与原版引擎开发复杂度对比

如表5.5所示，新版UI系统开发效率大大优于原版引擎开发效率。在新版UI系统时，不需要使用代码硬编译UI配置文件，减少了代码量和整体调试时间。在实际开发过程中，开发者只需把时间花费在游戏逻辑层上，界面可以交由其他人员负责。

在屏幕适配过程中，未发现错误。由于不同模块UI拼接时，提前预留了距离，未发现不符合逻辑的UI重叠。



图5.1 iPhone5C适配画面



图5,2 iPad Mini4适配画面



图5.3 Sony Z3适配画面

**5.3 游戏性能测试**

|  |  |  |  |
| --- | --- | --- | --- |
| 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 | 68,80% | 105 | 58 |

表5.6 游戏性能测试结果

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5.6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较高。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手机的存储空间要求。

**5.4 本章小结**

本章讲述了文章的测试部分。首先对改良后引擎进行测试，测试结果表明改良后引擎在CPU利用率还是帧率上满足了预期。之后对新版UI系统进行测试，新版UI系统功能正常，大大缩短了开发时间。之后对在此框架基础上构建的手机游戏进行测试，游戏功能正常。

**第6章 结论与展望**

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。首先绪论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引擎做了简要的介绍，同时针对卡牌类游戏和Cocos2D-x引擎的特点，分析了引擎中存在的问题，针对问题提出了进一步改进的需求分析。之后对该引擎的优化和实现进行了说明，其中优化的模块包括，渲染模块，场景管理模块，事件管理模块。接下来作者在优化过的引擎基础上构建了针对卡牌类游戏的新版UI系统，作者讲述了新版UI系统的组成及其中各个模块的设计与实现。最后对引擎和游戏进行了测试。

本文主要的成果如下:

(1).构建了新版的UI系统，新版UI系统包括配置文件管理模块，核心模块，系统级控件，游戏级控件，UI编辑器。新版UI系统更加贴合所要开发的2D卡牌类游戏，并且大幅加快了开发效率，降低了开发难度。

(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，同时提高了手机帧率。

(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。

在项目开发过程中也发现了一些问题，这些问题需要进一步的改进:

文章中UI系统中UI控件构建采用了继承模式，继承模式有一个严重缺陷，子类控件必须知道父类控件的所有实现，并且随着游戏的开发，最后一个控件的设计也会变得臃肿不堪。针对这个问题，可以模仿Unity实体管理中的组件模式去构建UI控件。把显示，运动，触发等功能做成独立的组件，根据具体数据来决定向UI控件中添加哪些组件，通过消息队列的方式联系不同组件间的通信。

**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. Brian W. Kernighan.The Practice of Programming[M].Addison-Wesley，1999
      11. 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
      12. 霍常伟.基于cocos2d\_x引擎的移动游戏UI系统设计及应用[D].北京交通大学，2012
      13. Dan Ginsburg ，Budirijanto Purnomo，Dave Shreiner，Aaftab Munshi.OpenGL ES 3.0 Programming Guide[M].Addison-Wesley Professional，2014
      14. Tomas Akenine-Moller，Eric Haines ，Naty Hoffman.Real-Time Rendering[M].3rd CRC Press，2008
      15. Jason Gregory.Game Engine Architecture[M].A K Peters，2009
      16. 痞子龙3D编程.多线程渲染[EB/OL].2011.http://blog.csdn.net/pizi0475/article/details/6243064
      17. Unreal Engine 4 Documentation[EB/OL].Epic Game.2016.https://docs.unrealengine.com/latest/CHN/index.html
      18. 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
      19. 房燕良.虚幻4渲染系统结构解析[EB/OL].2016.http://geek.csdn.net/news/detail/106495
      20. 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
      21. 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
      22. 秦春林.我所理解的Cocos2D-x[M].北京：电子工业出版社，2014
      23. Shekar，Sidharth.Cocos2D-x cross-platform game development cookbook[J].2016
      24. Martin Fowler.重构:改善既有代码的设计[M].北京：人民邮电出版社 2010
      25. 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
      26. 沈志超.基于Cocos2D-x的RPG手机游戏开发[D].东南大学，2015
      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，母校拥有良好的学习氛围，并且有着丰富多彩的校园生活。在这个环境中，我能不断的充实自己，提升自己。