中国矿业大学徐海学院

本科生毕业论文

姓 名： **卢振雨** 学 号**：** **22110449**

学 院： **中国矿业大学徐海学院**

专 业： **计算机科学与技术**

设计题目： **基于COCOS2D安卓版泡泡龙游戏的**

**\_\_\_\_\_\_\_\_\_\_\_\_设计与开发\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

专 题：

指导教师： **谢红侠** 职 称： **副教授**

2015年6月 徐州

中国矿业大学徐海学院毕业论文任务书

专业年级 计算机2011 学号 22110449 学生姓名 卢振雨

**任务下达日期： 2014年12月19日**

**毕业设计日期： 2015年2月26日至2015年6月13日**

**毕业设计题目：基于COCOS2D安卓版泡泡龙游戏的设计与开发**

**毕业设计专题题目：**

**毕业设计主要内容和要求：**

基于Cocos2d的安卓版泡泡龙游戏的设计与开发。

本游戏主要采用Cocos2d-x游戏引擎、Microsoft Visual Studio 2012开发工具和Eclipse开发工具进行开发。此款软件必须具有基本的泡泡龙射击类游戏的功能，如发射泡泡功能、泡泡消除功能等。在此基础上还需添加自己的一些小创意，使游戏可玩性高。此外，本游戏开发是基于Android智能手机，游戏需要处理图片等资源，所以要熟练掌握Eclipse等开发工具的打包，并借助PS等辅助工具完成本游戏的开发与设计。

指导教师签字：

郑 重 声 明

本人所呈交的毕业论文，是在导师的指导下，独立进行研究所取得的成果。所有数据、图片资料真实可靠，尽我所知。除文中已经注明引用的内容外，本毕业论文的研究成果不包含他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确的方式标明。本论文属于原创。本毕业论文的知识产权归属于培养单位。

本人签名： 日期：

中国矿业大学徐海学院毕业设计指导教师评阅书

指导教师评语（①基础理论及基本技能的掌握；②独立解决实际问题的能力；③研究内容的理论依据和技术方法；④取得的主要成果及创新点；⑤工作态度及工作量；⑥总体评价及建议成绩；⑦存在问题；⑧是否同意答辩等）：

成 绩： 指导教师签字：

年 月 日

中国矿业大学徐海学院毕业设计评阅教师评阅书

评阅教师评语（①选题的意义；②基础理论及基本技能的掌握；③综合运用所学知识解决实际问题的能力；③工作量的大小；④取得的主要成果及创新点；⑤写作的规范程度；⑥总体评价及建议成绩；⑦存在问题；⑧是否同意答辩等）：

成 绩： 评阅教师签字：

年 月 日

中国矿业大学徐海学院毕业设计答辩及综合成绩

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 答 辩 情 况 | | | | | |
| 提 出 问 题 | 回 答 问 题 | | | | |
| 正 确 | 基本  正确 | 有一般性错误 | 有原则性错误 | 没有  回答 |
|  |  |  |  |  |  |
| 答辩委员会评语及建议成绩：  答辩委员会主任签字：  年 月 日 | | | | | |
| 学院领导小组综合评定成绩：  学院领导小组负责人：  年 月 日 | | | | | |

**摘 要**

智能手机如今已经是很普遍，Android、IOS、Window Phone三大操作系统几乎占据了全部的手机市场；面对日益繁重的工作压力，人们在闲暇之余特别热衷于玩手机游戏，所以开发一款能够跨平台的休闲射击类游戏具有很高的商业价值。由于Cocos2d-x是一个跨平台开源的二维手机游戏框架，采用Cocos2d-x引擎开发的项目能够在Android、IOS、Window Phone等支持C++的平台下创建和运行。

本游戏主要是面向Android用户开发的安卓版本，为了便于以后的跨平台移植以Cocos2d-x为游戏引擎，在Microsoft Visual Studio 2012开发环境下采用C++语言进行开发，在Eclipse下调试并在Android智能手机上运行。

本文主要论述了开发泡泡龙游戏的各个阶段所做的详细工作，主要包括可行性分析、系统需求分析、系统概要设计、系统详细设计以及系统的编码和测试等内容。在可行性分析中，主要对系统是否值得开发进行各方面的分析，其中包括技术可行性、经济可行性、操作可行性和法律可行性；在系统需求分析中，主要对系统的功能进行需求分析并辅以系统功能结构图、系统用例图、系统流程图等加深理解；在系统设计中，主要对系统的游戏执行模块、泡泡模块、发射模块、消除模块等进行详细设计并画出类图、顺序图以及状态图；在系统的编码中，主要讲述怎样实现系统设计阶段提出的各个功能模块；在测试中，对软件成品进行黑盒测试。经过测试后，达到了可玩性良好、休闲益智的效果，最终得出所设计的游戏。

关键词：泡泡龙； Cocos2d-x； Android智能手机；

**ABSTRACT**

Smart phones and now is very common, Android, IOS, Phone Window three operating system almost occupied the entire mobile phone market, the face of increasingly heavy work pressure, people in her spare time particularly keen on playing games, so the development of a Cross-platform casual shooter with a high commercial value. As cocos2d-x is a cross platform open source 2D mobile game framework, the engine development project to Android, IOS, windows phone support for C + + platform to create and run ;

This game is mainly development for the Android user, in order to facilitate future cross-platform portability to Cocos2d-x for the game engine and development of C++ language in the environment of Microsoft Visual Studio 2012, debug under Android and run on Eclipse Smartphone.

This paper mainly discusses the development of bubble dragon game in all stages of the work done in detail, Including the feasibility analysis, system requirements analysis, system outline design, system design and system details of the coding and testing. In the feasibility analysis, the main system is worthy of the development of all aspects of the analysis, including technical feasibility, economic feasibility, operational feasibility and legal feasibility; In the analysis of the demand of the system, mainly the function of the system of demand analysis and combined with the system function structure diagram, the system use case diagram and system flow chart to deepen understanding; in the system design, mainly the game execution module, bubble module, a transmitting module, a cancellation module of detailed design and draw a class diagram, sequence diagram and state diagram; in the coding of the system, mainly about how to realize each function module of the system design stage, this paper. In the test, the finished product of software of black box testing. After the test, it reached the effect of good play, leisure and the puzzle, and finally got the game design.

**Keywords: Bubble Bobble; Cocos2d-x; Android Smartphone.**

目 录

[1 绪论 1](#_Toc421649798)

[1.1课题背景 1](#_Toc421649799)

[1.2泡泡龙游戏的历史 1](#_Toc421649800)

[1.3面临的主要问题 2](#_Toc421649801)

[1.4开发工具及技术 3](#_Toc421649802)

[1.5课题亮点 4](#_Toc421649803)

[1.6论文主要结构 4](#_Toc421649804)

[2 泡泡龙游戏的可行性分析 6](#_Toc421649805)

[2.1目标与方案 6](#_Toc421649806)

[2.2可行性分析 6](#_Toc421649807)

[2.2.1技术可行性 6](#_Toc421649808)

[2.2.2经济可行性分析 6](#_Toc421649809)

[2.2.3操作可行性 7](#_Toc421649810)

[2.2.4法律可行性 7](#_Toc421649811)

[2.2.5小结 7](#_Toc421649812)

[3 泡泡龙游戏的系统分析 8](#_Toc421649813)

[3.1功能需求分析 8](#_Toc421649814)

[3.1.1系统基本功能 8](#_Toc421649815)

[3.1.2系统功能结构 9](#_Toc421649816)

[3.1.3系统用例图 9](#_Toc421649817)

[3.2系统流程图 10](#_Toc421649818)

[3.3性能需求分析 11](#_Toc421649819)

[3.3.1游戏性能分析 11](#_Toc421649820)

[3.3.2游戏运行平台需求 12](#_Toc421649821)

[3.4游戏快速原型图 12](#_Toc421649822)

[3.4.1游戏主界面 12](#_Toc421649823)

[3.4.2选关模块界面 13](#_Toc421649824)

[3.4.3排行榜界面 14](#_Toc421649825)

[3.4.4关于界面 14](#_Toc421649826)

[3.4.5帮助界面 15](#_Toc421649827)

[3.4.6游戏界面 15](#_Toc421649828)

[3.4.7暂停、过关和失败界面 16](#_Toc421649829)

[4 泡泡龙游戏的系统设计 17](#_Toc421649830)

[4.1系统基本思路 17](#_Toc421649831)

[4.2软件设计目标 17](#_Toc421649832)

[4.3系统功能模块设计 17](#_Toc421649833)

[4.3.1游戏功能结构图 17](#_Toc421649834)

[4.3.2游戏执行模块设计 18](#_Toc421649835)

[4.3.3玩家泡泡发射模块的设计 18](#_Toc421649836)

[4.3.4泡泡模块的设计 18](#_Toc421649837)

[4.3.5彩蛋模块的设计 18](#_Toc421649838)

[4.3.6排行榜模块设计 18](#_Toc421649839)

[4.3.7计时模块的设计 18](#_Toc421649840)

[4.3.8设置模块的设计 18](#_Toc421649841)

[4.3.9界面特效模块的设计 19](#_Toc421649842)

[4.3.10碰撞检测模块的设计 19](#_Toc421649843)

[4.3.11音乐播放模块的设计 19](#_Toc421649844)

[4.3.12游戏关于模块的设计 19](#_Toc421649845)

[4.3.13游戏帮助模块的设计 19](#_Toc421649846)

[4.4系统优化设计 19](#_Toc421649847)

[4.4.1加载动画的优化 19](#_Toc421649848)

[4.4.2纹理的优化 20](#_Toc421649849)

[4.4.3排行榜时间存储的优化 20](#_Toc421649850)

[4.4.4游戏界面刷新帧率的优化 20](#_Toc421649851)

[4.5系统中类的设计 20](#_Toc421649852)

[4.5.1系统中类的说明 20](#_Toc421649853)

[4.5.2系统类图 21](#_Toc421649854)

[4.6系统游戏框架图 22](#_Toc421649855)

[4.7系统中关键部分状态图 23](#_Toc421649856)

[4.7.1状态图 23](#_Toc421649857)

[5 泡泡龙游戏的实现 25](#_Toc421649858)

[5.1安装开发环境 25](#_Toc421649859)

[5.2菜单界面的实现 26](#_Toc421649860)

[5.3泡泡射击的实现 28](#_Toc421649861)

[5.4游戏音乐及音效的实现 29](#_Toc421649862)

[5.5游戏暂停的实现 30](#_Toc421649863)

[5.6游戏核心的实现 30](#_Toc421649864)

[5.6.1消除算法 30](#_Toc421649865)

[5.6.2碰撞检测 33](#_Toc421649866)

[5.6.3计时系统 34](#_Toc421649867)

[6 泡泡龙游戏的测试 35](#_Toc421649868)

[6.1测试方法介绍 35](#_Toc421649869)

[6.2测试项目 35](#_Toc421649870)

[6.1.1菜单界面测试 35](#_Toc421649871)

[6.1.2选关测试 35](#_Toc421649872)

[6.1.3关卡测试 36](#_Toc421649873)

[6.1.4游戏设置测试 37](#_Toc421649874)

[6.1.5加载界面测试 37](#_Toc421649875)

[6.1.6其它测试 38](#_Toc421649876)

[6.3游戏帮助 38](#_Toc421649877)

[6.2.1游戏背景 38](#_Toc421649878)

[6.2.2游戏玩法 38](#_Toc421649879)

[7 结论 39](#_Toc421649880)

[参考文献 41](#_Toc421649881)

[英文原文 42](#_Toc421649882)

[中文译文 51](#_Toc421649883)

[致 谢 60](#_Toc421649884)

# 1 绪论

## 1.1课题背景

面对如今蓬勃发展的移动互联网，手机等移动终端不再仅仅扮演通讯工具这一单一角色。3G互联网如今基本已经普及，4G的时代也已登上舞台。回首近几年移动互联的发展，就国内而言，大到三大运营商和一线互联网公司，小到民营企业和个人开发者，都从移动互联这一块大蛋糕分得满意的成效。

自苹果公司发布了能够承载移动互联网功能的智能手机iphone之后诞生了一大批移动应用和移动游开发商。其中，以《疯狂的小鸟》为代表的第一批手机游戏进入人们的视野并大获成功。之后，手机游戏以比雨后春笋还要疯狂多倍的速度蓬勃发展起来。这时，Google发布的Android操作系统的智能手机更强化了这一浪潮，Android智能机以高性价比很快在全球普及开来，借助于该系统的游戏不管是质量上还是数量上也不断每月翻新。

《我叫MT》 、《捕鱼达人》 、《保卫萝卜》 、《三国杀》以及腾讯的《天天酷跑》 等天天系列游戏让人印象颇深，游戏玩家只增不减，盈利也是相当可观。

据CNET报道，Android和IOS市场占有率已达到96%，Android竟从81.4%一路飙升到83.6%。可以看出，安卓市场是十分广阔的。

移动互联的发展促进了人们多元化的娱乐方式，面对日益紧张的工作，人们很乐意这种小巧、轻松、随时随地的娱乐方式，这也是手机游戏的强大群众基数；

再看游戏厂商这一块，游戏厂商在游戏产品研发中将会投入更多资金和人员，更精品化的游戏会越来越多；游戏产品将偏向重度产品，运营时间长，生命周期会延长，一款游戏将会创造更多的价值。

## 1.2泡泡龙游戏的历史

最早的泡泡龙出现在1986年，是由TAITO开发的街机游戏。主角是两只绿色和蓝色的可爱小恐龙（名字分别叫Bub和Bob），会吐“泡泡”，故称“泡泡龙”。 泡泡龙可以一人或二人进行游戏，共有100关，每一关的游戏范围只有一个屏幕的大小，没有任何滚动，而部份关卡玩家可从下面的两个洞“回到”屏幕上。过关方法为消灭画面上所有的敌人，一般是用吐出的气泡包住敌人，然后将气泡弄破即可消灭。

在最初的街机泡泡龙大红大紫之后，开始不断有公司在PC端开发泡泡龙系列游戏。例如火拼泡泡龙，水晶泡泡龙，风火泡泡龙等。从这个阶段开始，泡泡龙正式进入了三消的模式。当玩家主动令游戏区形成3个或以上的同色泡泡相连时，这些泡泡将会爆掉。如果爆掉的泡泡造成其他连接的泡泡出现断裂，则断裂的泡泡都会下坠，自动消除掉。泡泡也可以利用两侧垂直的墙壁进行反弹，这是越过障碍击打上层泡泡的重要技巧。这个时期的游戏都是大同小异，单一的玩法，和单调的画面，不过在当时来说也是非常风靡的休闲游戏了。

随着科技的发展，手机的不断更新迭代。手机游戏成为公交，地铁以及休息时间的重要娱乐项目也是大势所趋。泡泡龙作为经典、传统的休闲游戏，当然也不甘落后。大批的泡泡龙系列游戏进入了手机市场，玩法也是在不断的创新。比较经典的游戏例如亚特兰蒂斯泡泡龙、旋转泡泡龙、开心泡泡龙等等。这些游戏都代表着泡泡龙进入手机市场的这个时代。

## 1.3面临的主要问题

本次系统的开发，主要面临以下几个问题：

（1）要学习Cocos2d-x游戏引擎，由于自身并没有接触过Cocos2d-x游戏引擎，所以要以最快的速度了解Cocos2d-x引擎以及Cocos2d的组件，并在短时间内能够使用Cocos2d-x的引擎框架。一方面，由于Cocos2d-x不像Unity3D引擎有一套很规范的开发标准和统一的处理界面，所以要学会其他编辑工具的使用和在windows搭建运行环境；另一方面，Cocos2d-x的内存管理机制和图层渲染还不是很完善，有可能会造成内存泄露从而导致程序崩溃，因此在游戏开发的初期就要处理好内存管理。

（2）本次开发是第一次进行游戏开发，以前都是开发应用软件，所以对游戏机制还不是很了解，要对其进行学习和掌握。另外也需要对Cocos2d-x中的导演、场景、层以及精灵等新接触的组件进行相应的学习和了解。

（3）由于本身不是美工出身，游戏引擎中的地图系统牵涉到一些和专业无关的美术知识，所以游戏中的图片和音频资源均来源于网络并作相应的处理。本次设计没有商业用途只是作为毕业设计答辩之用并没有对原资源的原作者造成侵权。

（4）本次游戏开发环境使用的是Eclipse，并在Microsoft Visual Studio 2012下进行编译，开发语言是C++，由于Java和C++ 之前使用的是NDK进行交叉编译的，所以要对NDK环境进行相关配置，在编译的同时由于Android需要模拟Linux环境，还要安装配置python，代码的连接是采用JNI，所以还需要了解JNI的使用和以及他们不同编程语言的交互。

（5）由于Cocos2d-x是一个跨平台的游戏引擎，由于本身以及移植技术不是非常成熟，在移植Android时可能会出现很多意想不到的问题，比如引擎环境的搭建、python交叉编译环境的搭建、目录的选择、Cocos2d-x扩展组件的使用、Android.mk编译C++文件的使用，这些都是要做好相应的准备的。其中在 Eclipse下交叉编译的话，要新建Builds配置，其配置如下图（1.1），另外采用Eclipse开发并使Cocos2d-x-3.0引擎需要引入扩展环境防止编译Android.mk文件报错，其配置见下图（1.2）：

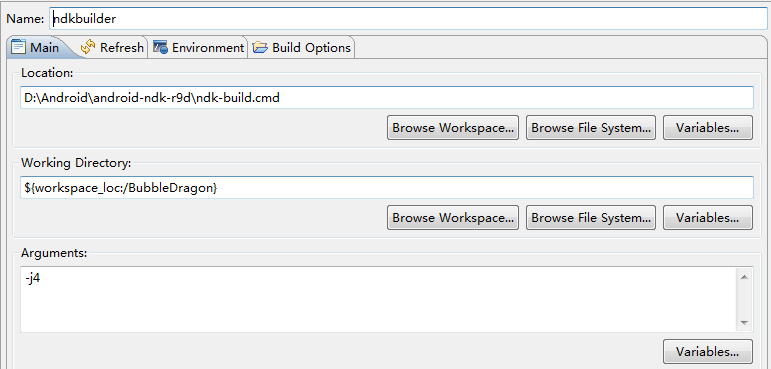


图1.1新建Build配置图

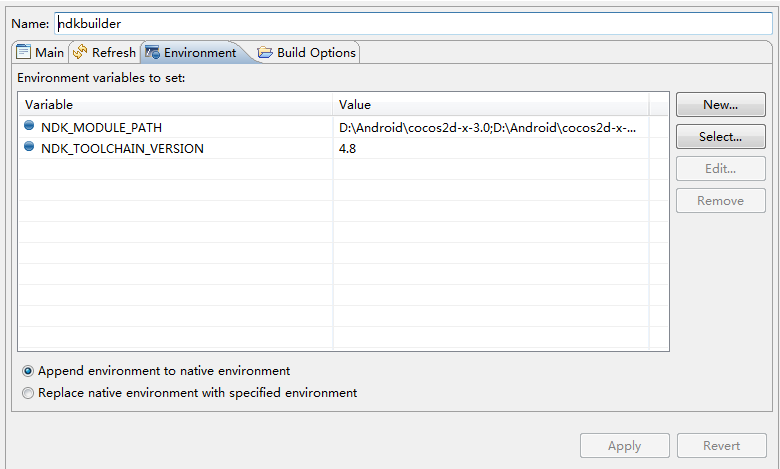


图1.2引擎开发模型路径配置图

其中NDK\_MODULE\_PATH为：

D:\Android\cocos2d-x-3.0;D:\Android\cocos2d-x-3.0\cocos;D:\Android\cocos2d-x-3.0\extensions;D:\Android\cocos2d-x-3.0\external;

（7）游戏开发完成的发布与宣传，后期的产品维护和更新。玩家的消息反馈等都要做统一的规划。

## 1.4开发工具及技术

（1）Cocos2d-x是MIT许可证下发布的开源跨平台游戏引擎，游戏开发快速、简易、功能强大。Cocos2d-x的核心优势是允许开发人员利用C++、Lua及Javascript来进行跨平台部署，覆盖平台包括iOS、Android、Windows Phone以及黑莓（BlackBerry）等[[[1]](#endnote-1)]。

（2）Microsoft Visual Studio 2012是微软公司在2012年推出的开发环境。其IDE（集成开发环境）的界面被重新组织与设计，使用起来更加简单明了。Microsoft Visual Studio 2012包括了如IDE（集成开发环境）、UML工具、代码管控工具这样的，在整个软件生命周期中所需要的大部分工具[[[2]](#endnote-2)]。

（3）Eclipse是一个基于Java的可扩展开发平台，最初是由IBM捐献给开源社区的，目前已经发展成为人气最旺的Java IDE[[[3]](#endnote-3)]。

（4）NDK， Android NDK 是在SDK前面又加上了“原生”二字，即Native Development Kit，因此又被Google称为“NDK”。Android程序运行在Dalvik虚拟机中，NDK允许用户使用类似C / C++之类的原生代码语言执行部分程序。

NDK包括了：

①.从C / C++生成原生代码库所需要的工具和build files。

②.将一致的原生库嵌入可以在Android设备上部署的应用程序包文件（.apk文件）中。

③.支持所有未来Android平台的一些列原生系统头文件和库[[[4]](#endnote-4)]。

（5）Python交叉编译，简单地说是把一种平台上的原生代码生成能运行在另一种平台上的代码 [[[5]](#endnote-5)]。

（6）Adobe Photoshop 简称“PS”，是由Adobe Systems开发和发行的图像处理软件，Photoshop主要处理以像素所构成的数字图像。使用其众多的编修与绘图工具，可以有效地进行图片编辑工作。ps有很多功能，在图像、图形、文字、视频、出版等各方面都有涉及[[[6]](#endnote-6)]。

## 1.5课题亮点

（1）本游戏玩法简单，操作易上手，老少皆宜，休闲益智。人们在闲暇之余或者压力太大时可以随时随地拿出来玩，界面设计温和美观，音乐动听，很容易让玩家放松。

（2）本游戏相对于Android原生游戏由于采用Cocos2d-x引擎，而编译采用的是C++，所以跨平台移植性较好，一份代码可以平台间互相使用。

## 1.6论文主要结构

第一章 绪论。首先概述课题研究的背景，从国际国内和游戏玩家与游戏开发商四个方面阐述了手机游戏具有广阔的发展潜力，简述了本游戏的发展历史，列举了七大开发中需要解决的问题，并简述了本课题具有的强有力的优势，详细分析了开发中要用到的开发工具，最后又表达了本课题的创新之处。

第二章 可行性分析。主要对技术可行性、经济可行性、操作可行性、法律可行性这几方面进行分析，最后得出是否值得开发的结论。

第三章 系统分析。首先对系统的功能进行需求分析，制定开发需求，大致开发思路以及主要开发哪些模块，其次通过画出系统流程图、用例图对系统进行更加细致的分析。将每个成员所担任的职责分析清楚，最后，对性能进行需求分析并画出快速原型图，并提供了相应的测试方案。

第四章 系统设计。这一部分是为了各个模块的编码做准备的，对系统的各个功能模块都会进行详细的设计。包括编码统一规范，对象之间的引用，对可以预见的一些问题提前进行处理，如系统优化设计。最后绘制出系统中类的设计和关键部分的顺序图及状态图。在开发中按照该设计图进行编码。

第五章 系统实现。这部分主要讲述怎样用代码实现系统设计阶段提出的各个功能模块。包括算法的分析、算法的建模和算法的实现。在此过程中还要对已知的bug进行修复和局部优化。

第六章 系统测试。这部分主要对已开发出的软件进行黑盒测试。并对代码进行优化和bug修复。

第七章 结论。这部分主要总结开发过程中遇到的问题、软件存在的不足之处、软件的创新之处以及总结出的经验。

# 2 泡泡龙游戏的可行性分析

## 2.1目标与方案

本游戏是以Cocos2d-x游戏引擎为框架开发的休闲射击类游戏，开发工具为Eclipse和Microsoft Visual Studio 2012，开发语言为C++。开发过程中借鉴了目前市场上已经发布的几款泡泡龙类型的手机游戏和网页版的游戏他们的共同点是，“3+消”的方式，也就是当发射的泡泡到达最终点位置时，若果有3个或者3个以上颜色相同的泡泡相连，则相连的泡泡将会消失，其不同在于玩法和游戏内容。本游戏去其糟粕，取其精华，再加入自己的创意。运用Cocos2d-x游戏引擎增加了酷炫的特效，提供了计时排行和倒计时两种玩法。最终的目标是开发出一款受众面广，内容丰富，易上手的能在移动设备上玩的休闲益智类游戏。

游戏中应该包括有主场景提供本游戏的操作菜单、选关场景提供游戏的不同玩法的进入入口、游戏场景提供游戏的主要操作界面、排行榜场景提供玩家玩游戏所用的时间和排行、帮助场景提供游戏的玩法介绍、关于场景提供开发者的介绍。

## 2.2可行性分析

### 2.2.1技术可行性

（1）开发者技术是否能达到：

本游戏采用Cocos2d-x游戏引擎为框架，在Eclipse（ADT）开发环境下编译执行，在Microsoft Visual Studio 2012编译环境中运用C++进行预先编译开发，最后Eclipse进行打包发布。由于出身于Android开发，所以能够完成对其的快速熟悉并上手。只是Cocos2d移植到Android上开发环境需要一系列配置，交叉编译也要配置无误，这些可以通过官方教程完成安装与配置。C++以前也接触过，对于其语法用法也有所了解，编写起来也容易上手。

（2）手机硬件水平是否能达到：

随着科学技术的发展，硬件更新换代速度非常快，目前大多数智能手机的硬件的最低配置都基本符合本游戏的配置，所以手机硬件能够达到所需要的水平。

（3）在规定的时间内，能否完成开发：

从开发人员自身以及所拥有的时间来看，能够按时完成任务。

### 2.2.2经济可行性分析

Cocos2d-x为开源的游戏引擎，所以不用为此花费资金。Eclipse也是开源工具，Microsoft Visual Studio 2012是Microsoft提供的开发者工具，是需要付费的，但是对于不是用于商业开发来说采用网上破解的即可，对开发不能造成影响。

所以在经济上完全可行。

### 2.2.3操作可行性

该游戏是通过手指触摸手机触摸屏来发射泡泡，达到射击效果的。目前，几乎所有的智能手机都为触屏手机，在操作上都能达到想要的效果。

### 2.2.4法律可行性

该游戏是自主研发设计的并且内容积极健康，不会造成侵权，不会违反相关法律。所以用法律的眼光看待此游戏的开发，是完全合法可行的。

### 2.2.5小结

根据上述对技术、经济、操作、法律的可行性分析。我认为开发者的能力能够应对此项目的难度，而且软件结构清晰、功能合理有亮点、成本不算高、收益空间大、发布方式多，所以本游戏项目可以执行。

# 3 泡泡龙游戏的系统分析

## 3.1功能需求分析

### 3.1.1系统基本功能

（1）加载动画场景

在手机上安装上后点击运行“泡泡龙”游戏，进入游戏场景，首先是加载游戏资源，将游戏的图片资源与音频资源预先加载到内存。玩家可以通过加载界面直观地看到加载进度，当加载100%时进入下一界面，即游戏菜单场景。

（2）游戏菜单场景

该场景是整个游戏操作的入口，是整个游戏的主场景，包括有“开始游戏”、“帮助”、“排行榜”、“音效”、“关于我们”和“退出”按钮。点击音效旁边的按钮可以控制在游戏的过程中是否播放音效和背景音乐。

（3）游戏选关场景

在游戏菜单场景中点击开始游戏会进入到本场景。里面包括三个五角星“★”选关按钮，和动画显示的“关卡选择”提示，并且可以点击返回键返回上一界面。当点击 m 每一个关卡按钮时都有对应的动画反馈给玩家，设计思想舒服贴切。

（4）游戏场景

在关卡选择按钮点击之后会进入到该场景，在前两关主要元素包括，“泡泡容器”、“发射泡泡指针”、“待发射泡泡容器”、“泡泡边界”、“计时提示”、“暂停按钮”，在第三关会有倒计时“支柱”，会根据时间的增长一点一点下落，直到到达发射泡泡的终点。当点击屏幕发射区域时，泡泡会和发射点构成“两点一线”发射出去到最终位置，若存在三个及以上相同颜色的泡泡就会掉落并伴随效果。

（5）帮助场景

在游戏菜单场景中点击“帮助”进入到“帮助场景”，本场景为玩家图文展现了该游戏的玩法，在该场景下方会有“上一页”和“下一页”菜单项，让玩家翻阅阅读两种不同玩法。

（6）排行榜场景

在游戏菜单场景中点击“排行榜”进入到“排行榜场景”，本场景主要显示了本游戏中所有关卡排行前5的游戏记录，可以通过下方“上一页”和“下一页”菜单项查看不同关卡的排行。玩家也可以通过点击“重置”按钮来清空数据。

（7）特效功能

在游戏菜单场景中，“泡泡龙”logo上会有粒子特效，在射击泡泡时，该泡泡到达最终位置会有粒子效果，当泡泡消除时会根据泡泡消除的情况给出彩蛋提示“Good”、“太棒了”、“不可思议”、“无懈可击”、“完美一击”和“独一无二”提示。在游戏过关和失败时也会有相应的粒子特效展示。

（8）关于场景

在游戏菜单场景中点击“关于我们”进入到“关于场景”，本场景显示开发者的信息。

（9）游戏暂停、重新开始、继续游戏场景

在游戏过程中当点击游戏暂停按钮时，会弹出一个菜单选择对话框，玩家可以选择“继续游戏”、“主菜单”和“重新开始”，点击会执行对应的操作。

（10）游戏过关和失败场景

如果玩家成功消除掉所有泡泡，将会看到游戏成功过关场景，其中显示了本次游戏用时和排行前五的游戏记录。如果游戏失败会进入失败场景，会有一个“LOSER”提示并伴随效果，该场景也会提示“点击重玩”，当点击屏幕时会重新开始本关游戏

### 3.1.2系统功能结构

本游戏的主要功能模块有游戏加载、关卡选择、发射泡泡、游戏帮助、碰撞检测、音乐控制、粒子特效、计时系统、暂停功能、泡泡消除等。其中关卡选择分为计时关卡和倒计时关卡、粒子特效分为实时特效和触发特效、计时系统分为时间读写和时间更新、音乐控制分为打开与关闭音乐、暂停功能包括继续游戏、返回主菜单和重新开始。

### 3.1.3系统用例图

该泡泡龙游戏的玩法主要分为两种类型，一个是计时模式，就是通过比较不同玩家或者同一玩家不同次游戏的用时来达到游戏目的；另外一种方式是倒计时模式，即在规定的时间内，在玩游戏的同时，倒计时柱会随着时间的增长而下降，在倒计时柱没有接触发射最底部或者泡泡没有接触最底部时都是有效的，直到失败或者全部消除，与此同时也会比较玩家的用时。在整个游戏中玩家可以通过控制是否开启音效来选择在游戏过程中是否播放音效和背景音乐。玩家在主菜单中可以查阅排行和本游戏的玩法，也可以退出游戏。

经过分析，泡泡龙用例图如图3.1所示：



图3.1系统用例图

## 3.2系统流程图

如图3.2所示，玩家启动游戏后首先是加载游戏的图片和声音资源，然后进入到游戏的主菜单，玩家可以选择三个关卡来游戏，这三个关卡前两个是计时模式，比较谁的用时最短，第三关是倒计时模式，在倒计时柱没有下落到最底部之前必须要成功过关否则失败，同时也会比较用时。进入到游戏界面玩家通过点击屏幕进行发射泡泡，当该泡泡的颜色和它周围的泡泡有超过3个及以上相同颜色的泡泡时就会消除，直到所有泡泡消除，在此期间如果泡泡没有完全被消除就到达底部则游戏失败，如果此时玩家选择了暂停游戏，则可以选择是否继续或者重新开始本关游戏再或者是返回到主菜单。在主菜单中玩家可以查看其它模块或者退出游戏。在排行榜模块中可以查看排名前五的游戏用时，关于、帮助模块可以查看相应的游戏说明，他们都可以通过返回到主菜单进行关闭游戏，在音效模块中，打开则整个游戏过程都会播放音效，如果关闭则不会播放音效。



图3.2系统流程图

## 3.3性能需求分析

### 3.3.1游戏性能分析

游戏应该本着正常玩不出现未响应，在画面上应该流畅不出现卡顿的现象，不会出现由于特效的加入影响手机性能的情况。具体体现在：

（1）时间特性

系统响应时间应在人的感觉和视觉范围内(<1s)，系统响应时间足够迅速(<5s)。

（2）画面刷新帧率

人的最低动态画面感是25帧/s，所以为了不影响游戏特效设置了游戏最低刷新帧率为30帧/s。

（3）适配性

由于市场上Android智能手机纷繁嘈杂，并没有统一的大小和像素，所以为了能够适应更多、更普遍的用户采用自适应的布局，应该将如标签根据屏幕分辨率按照比例放置。

（4）编译配置

由于交叉编译耗费的系统资源很大要求CPU频率大于等于800MHZ。内存大于2G。

### 3.3.2游戏运行平台需求

（1）智能手机或电脑。

（2）RAM大于等于512M。

（3）ROM大于等于1G。

（4）CPU频率大于等于800MHZ。

（5）操作系统为IOS/Android/Windows phone等支持 C++的平台。

## 3.4游戏快速原型图

### 3.4.1游戏主界面

游戏主界面是游戏的入口，包含了整个程序的所有的主要操作菜单，其中设计了进入游戏的入口按钮、查看排行榜、游戏帮助和关于菜单，当然还要提供给玩家退出游戏的入口。其快速设计图见下图3.3：



图3.3游戏主界面

### 3.4.2选关模块界面

游戏的选关界面是开始游戏的入口，设计的是三关，采用三个“★”按钮用户点击相应的按钮就可以进入对应的关卡。其快速设计图如下：



图3.4游戏选关界面

### 3.4.3排行榜界面

游戏排行榜界面主要显示最多前五名玩家的用时记录，他们按照用时由短至长排列如下效果图：



图3.5游戏排行榜界面

### 3.4.4关于界面

游戏关于界面只需要显示开发人员的信息即可：



图3.6游戏关于界面

### 3.4.5帮助界面

帮助界面应该采用图文的形式向玩家简单陈述游戏的玩法，设计脑图见下图：

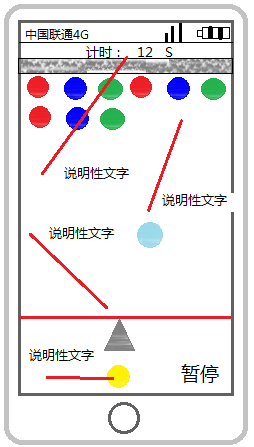


图3.7游戏帮助界面

### 3.4.6游戏界面

游戏主题界面设计，将手机屏幕最上端显示用时和泡泡容器上边界，左右边界是手机可显示的左右侧边框，最下边中间是发射泡泡的发射点，右下角设计了暂停游戏按钮可参考下图设计：

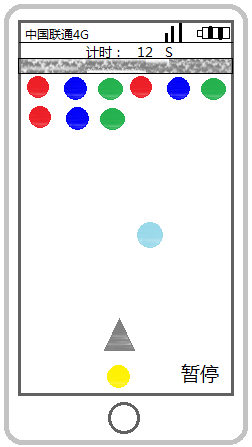


图3.8游戏界面

### 3.4.7暂停、过关和失败界面

游戏暂停时会弹出供用户选择的二级菜单对话框，其中应该包括“继续游戏”、“返回主菜单”和“重新开始”最基本的操作；过关界面跳转到类似排行榜界面，要显示本次的用时和过关提示信息；失败界面同样要提供失败提示信息，还要给玩家提供退出游戏和继续游戏的入口，其快速设计原型图大致设计为如下图：

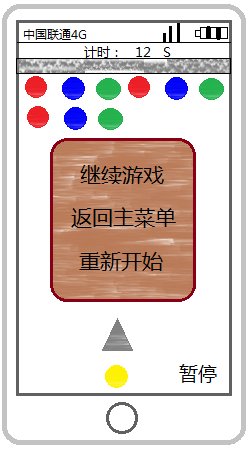
  

图3.9游戏暂停界面 图3.10游戏过关界面 图3.11游戏失败界面

# 4 泡泡龙游戏的系统设计

## 4.1系统基本思路

该游戏是触屏操作，就是通过手指点击屏幕来选择泡泡发射的方向，泡泡发射点和手指点击点会构成“两点一线”从而发射出去，当发射出去的泡泡到达最终位置时，则搜索附近的泡泡，如果有颜色相同则计数加1，再在下一个相同颜色递归寻找，如果计数大于等于3就当前这几个计数的颜色泡泡消除，如果其附着点的泡泡消失，则附着泡泡也要掉落，直到最底端的泡泡触及到发射点的平面；在第三关中，游戏计时柱会层层往下压低泡泡，使泡泡整体布局下压，直到最底端的泡泡触及到发射点的平面。

## 4.2软件设计目标

玩家通过点击屏幕发射泡泡，当发射的泡泡到达最终位置和相邻的泡泡有3个及以上颜色相同则消除，并伴随特效。在玩家玩的过程中玩家可以通过设置是否开启音效控制游戏音乐。最终过关的标准是，在最底端的泡泡未达到发射点平面完成消除所有泡泡。玩家也可以在游戏过程中选择暂停，来选择是否继续还是重新开始。

## 4.3系统功能模块设计

### 4.3.1游戏功能结构图

本泡泡龙游戏主要功能有发射泡泡、音效控制、粒子特效、游戏暂停、泡泡消除等，具体功能见下图4.1：

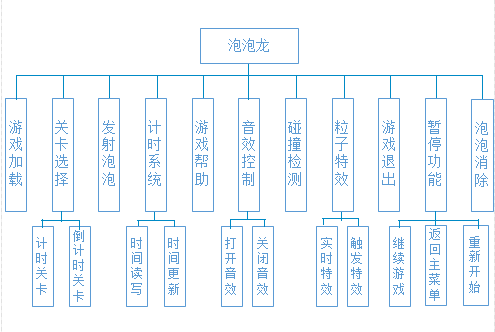


图4.1游戏功能结构图

### 4.3.2游戏执行模块设计

游戏在执行过程中会不断刷帧，不断更新泡泡位置和泡泡颜色以及计时器。在不断执行的过程中会做以下判断：

（1）泡泡是否可以消除。

（2）最底端的泡泡是否达到发射平面。

（3）第三关中计时是否达到计时柱要下落的时间点。

（4）泡泡是否全部消除。

（5）玩家有无点击暂停。

### 4.3.3玩家泡泡发射模块的设计

在发射平面以上到泡泡最后一排都是泡泡可达到区域，玩家通过手指触摸可点击区域发射泡泡，待发射泡泡会根据发射后的泡泡自动随机补上。

### 4.3.4泡泡模块的设计

泡泡总共有七种颜色，当玩家每发射一个泡泡时，它的周围会有粒子特效并持续一小段时间。

### 4.3.5彩蛋模块的设计

游戏会通过判断打落的连续泡泡个数给出玩家以视觉反馈和激励，比如消除三个是“Good”，消除四个是“太棒了”，凭借反射原理消除是“完美一击”，消除一片是“独一无二”等等。

### 4.3.6排行榜模块设计

在玩家通关一个关卡游戏时会将玩家用时记录下来存储，并展示给玩家；玩家可以通过查看排行榜查看用时，由于数据量有限，所以只显示前五名用时最短的记录。

### 4.3.7计时模块的设计

当游戏开始时就会启动计时，直到游戏过关或者失败，则计时暂停并存储该计时，如果在计时文件里超过五个就会比较该时间是否大于原来时间的最大值，如果大于，则舍弃该计时在排行榜中的存储，反之计入排行榜。

### 4.3.8设置模块的设计

玩家可以在主菜单中选择是否开启音效，如果游戏暂停，如果音效出于播放状态则暂停。

### 4.3.9界面特效模块的设计

界面特效包括两种，一个是游戏主菜单中logo的周围特效，另外一个是游戏过关或者失败的显示特效。

### 4.3.10碰撞检测模块的设计

由于泡泡是半径相同的圆形图片精灵，不考虑内切，如果两个泡泡碰撞，那么泡泡的圆心距肯定是两个泡泡的半径之和，也就是一个泡泡的直径。当然，还有内切的情况，它和第三个情况相交在游戏中是不会出现的，所以碰撞即d=2r，见下图4.2：



图4.2碰撞检测原理图

### 4.3.11音乐播放模块的设计

若玩家选择了打开背景音乐和游戏音效，当游戏开始时，音乐模块需要把音乐资源载入程序并且播放。玩家暂停或者游戏进入手机后台则音效和背景音乐将暂停播放。

### 4.3.12游戏关于模块的设计

显示开发者姓名版权等信息。

### 4.3.13游戏帮助模块的设计

显示游戏的为了帮助玩家上手的说明性引导内容。本游戏采用图文并茂的方式，简单易懂，玩家上手快，休闲又益智。

## 4.4系统优化设计

### 4.4.1加载动画的优化

由于加载动画的数字百分比和计时的数字是实时刷新而且频繁出现的，如果使用变量加以字体，就和游戏整体风格不符，所以采用了图片的处理方式。那么就会遇到不断地加载0~9的图片再不断消除它们，这给系统带来了负担，所以将它们这些数字做成了大小相同的图片位图，只需要加载一次图片资源，程序中不断根据每一张数字的大小切割图片就行了，这样就做到了资源的优化。

### 4.4.2纹理的优化

TexturePacker是一款纹理打包软件，可以把所有的纹理以最小的存储空间打包成一张图片并生成一个plist文件，而且Cocos2d-x为开发者提供了使用plist文件的函数降低了图片资源的加载耗时和占用过多内存。

### 4.4.3排行榜时间存储的优化

由于本游戏所需存储的数据并不多，只是存储游戏用时而且最多15个数据，所以我们不用诸如sql server、MySQL之类的数据库，否则它们会为我们游戏的开发带来不必要的消耗。因此，采用了Cocos2d-x提供的CCUserDefault类，CCUserDefault采用XML存储技术，通过shareUserDefault（）函数来获取实例。如此大大减少了在数据存储上所花费的时间。

### 4.4.4游戏界面刷新帧率的优化

正常情况下游戏的刷新帧率为60，这会使并不对刷新帧率要求太高的泡泡龙游戏开销太大，所以可以采用普通渲染机制使帧率在30左右，并不影响游戏的视觉效果和流畅度。Cocos2d-x游戏引擎中提供了批次渲染机制CCSpriteBatchNode，这种机制能使特效粒子只被渲染一次，大大提高了游戏的运行效果。

## 4.5系统中类的设计

### 4.5.1系统中类的说明

（1）游戏常量头文件——AppMacros

常量类中存储了游戏的一系列信息，包括各个节点的层次、泡泡的初始位置数组、泡泡发射速度等。

（2）游戏场景管理类——ShootBubbleSceneManager

该类包含了主菜单的部署方法和场景创建的方法。是整个游戏的核心类，其中还包括游戏中泡泡颜色的搜索，泡泡的消除、泡泡的掉落等核心方法，其它类基本都要引入或者定义该类的指针，在合适的场景中利用其中的创建场景的方法，或者抽取出的方法，比如初始化泡泡位置的方法。

（3）主场景布景类——StartLayer

该类主要处理主菜单中的回调方法和变量。其中包含了主菜单页面的布局，点击事件触发的回调事件。

（4）选关布景类——ChooseGKLayer

该类声明了一些列选关的布景数组和回调方法。其中包含了选关的布局和点击选关、返回触发的回调事件。

（5）帮助布景类——HelpLayer

该类声明了一系列帮助的布景成员变量和回调方法。内部包含了背景精灵展示的帮助内容、翻页按钮以及回调等。

（6）排行榜布景类——JFBLayer

该类声明了一系列排行榜的布景成员变量和回调方法。本类主要展现计时排行，内部封装了判断时间长短和保证不多于五个数据的方法。

（7）关于布景类——AboutUSLayer

此类声明返回键回调和布景的初始化方法。主要是布局了文本标签用于展示开发者信息。

（8）引擎入口类——AppDelegate

该类声明了整个程序在初始化、前台和后台要操作的方法回调。

### 4.5.2系统类图

限于空间，图4.3展现了几个重要的类之间的关系以及类中重要的成员函数，游戏中图片精灵均继承Sprite，菜单类均继承于MenuItemImage在图中不再做赘述。由于ShootBubbleLayer类中方法众多，主要方法如下表4.1：

表格4.1 ShootBubbleLayer类方法列表

|  |  |  |
| --- | --- | --- |
| 返回类型 | 函数名 | 备注 |
| void | initLocation() | 初始化屏幕中泡泡位置 |
| void | initIsEmpty() | 初始化关卡数组 |
| void | drawBall() | 绘制泡泡 |
| void | addBall() | 更新发射的泡泡 |
| void | puaseCallback() | 游戏暂停回调 |
| void | shootBall(Pointlocation) | 发射泡泡 |
| void | calPath | 泡泡是否碰撞 |
| bool | calEndBall | 泡泡运动路径 |
| void | search(int row,int col) | 递归搜索判断颜色是否相同 |
| bool | searchRoad(int row,int col) | 搜索所有泡泡周围是否有节点 |
| void | update() | 更新倒计时柱 |
| void | updateLocation() | 更新泡泡位置 |
| void | removeNodeGird() | 删除NodeGird对象 |

图4.3系统类图

## 4.6系统游戏框架图

游戏框架图如下图4.4所示，AppDelegate是cocos2d-x引擎入口。



图4.4游戏框架图

## 4.7系统中关键部分状态图

### 4.7.1状态图

待开发的泡泡龙游戏主要对象有泡泡二维数组对象、发射泡泡对象、待发射泡泡对象、游戏音效控制对象等，下面是一些核心对象的状态图：

（1）泡泡数组对象状态图，如图4.5所示：



图4.5泡泡数组对象状态图

（2）发射泡泡对象状态图，如图4.6所示：



图4.6发射泡泡对象状态图

（3）游戏音效控制对象状态图，如图4.7所示：

图4.7游戏音效控制对象状态图

# 5 泡泡龙游戏的实现

## 5.1安装开发环境

（1）JDK安装配置

（2）Microsoft Visual Studio 2012安装

（3）Cocos2d-x 3.0游戏引擎安装

（4）Python 2.7.9 安装配置

（5）NDK下载配置

（6）ant下载配置

（7）配置对应的环境变量

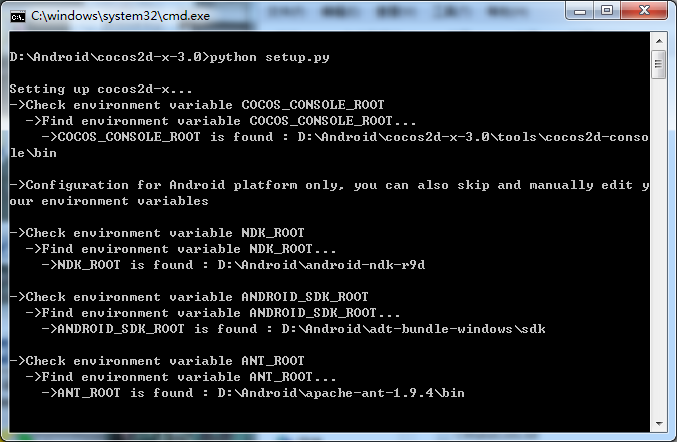
可以通过在Cocos2d-x安装目录下的setup.py文件采用命令行的方式配置对应环境的变量和新建项目，见下图：

图5.1开发环境配置图

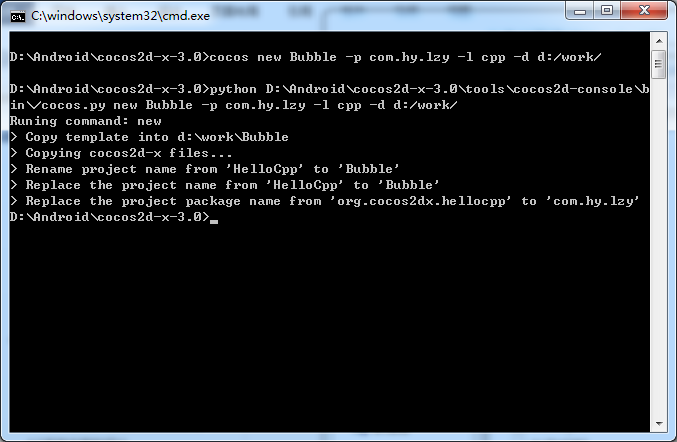


图5.2新建Cocos2d项目图

## 5.2菜单界面的实现

游戏启动后先加载资源动画，然后进入主菜单界面，其实现核心如下：

（1）创建加载动画精灵

通过导演类获取到导演实例Director::getInstance()，通过Layer类中的addChild()方法添加节点。

（2）加载动画初始化方法

|  |
| --- |
| for(int i=0;i<16;i++)  {  Director::getInstance()->getTextureCache()->addImageAsync( //加载图片资源  sa[i], //图片资源路径  CC\_CALLBACK\_1(StartLayer::loadingCallBack, this) //加载完成时被回调的方法  );  } |

（3）加载完成

|  |
| --- |
| if(jd==16)  {// 如果加载完成  lPros->setString("100%");  this->removeChild(lPros);  } |

（4）创建主菜单界面

|  |
| --- |
| Sprite\* mainbg = Sprite::create("mainpic/bg.png");//背景  Sprite\* sppl = Sprite::create("mainpic/menu\_title.png");//标题  ParticleSystemQuad\* psq = ParticleSystemQuad::create("lzxt/stars.plist");//粒子特效  //菜单创建  MenuItemImage\* puzzle = MenuItemImage::create  (  "mainpic/start.png", //正常图片  "mainpic/startp.png", //按下时图片  CC\_CALLBACK\_0(StartLayer::startCallback, this) //点击被回调掉的方法  ); |

（5）回调方法的创建

|  |
| --- |
| //创建帮助菜单项  MenuItemImage\* arcade = MenuItemImage::create(  "mainpic/help.png", //正常图片  "mainpic/helpp.png", //按下时的图片  CC\_CALLBACK\_0(StartLayer::helpCallback, this) //点击时被回调的方法  ); |

（6）回调方法的使用

|  |
| --- |
| void ShootBubbleSceneManager::gohelpScene()  {  // 设置深度检测  Director::getInstance()->setDepthTest(true);  // 创建一个场景对象  helpScene = Scene::create();  // 创建一个布景对象  HelpLayer\* hlayer = HelpLayer::create();  hlayer->sbsm=this;  // 将布景对象添加到场景中  helpScene->addChild(hlayer);  // 创建一个切换场景到特效  auto ss=TransitionProgressOutIn::create(1, helpScene);  // 切换到帮助场景  Director::getInstance()->replaceScene(ss);  } |

## 5.3泡泡射击的实现

（1）初始化泡泡

|  |
| --- |
| void ShootBubbleLayer::intiBall()  {  //产生一个1-8的随机数  srand((unsigned)time(NULL));  int i = rand()%8+1;  //创建一个精灵对象  ballNextStart=Sprite::createWithTexture(  Director::getInstance()->getTextureCache()  ->addImage("pic/ball\_"+StringUtils::format("%d",i)+".png"));  ballNextStart->setPosition(Point(visibleSize.width/2,50));  this->addChild(ballNextStart,GAME\_LEVEL);  //产生一个1-8的随机数  nextBallColor=i;  int j = rand()%8+1;  //创建一个精灵对象  ballStart = Sprite::createWithTexture(  Director::getInstance()->getTextureCache()  ->addImage("pic/ball\_"+StringUtils::format("%d",j)+".png"));  ballStart->setPosition(Point(visibleSize.width/2,140));  this->addChild(ballStart,GAME\_LEVEL);  // 将下一个将要发射的球的颜色记录为当前颜色  currBallColor=j;  } |

（2）泡泡的绘制

|  |
| --- |
| tempSprite=Sprite::createWithTexture(  Director::getInstance()->getTextureCache()  ->addImage("pic/"+s+"\_1.png")); |

（3）触摸屏幕

|  |
| --- |
| bool ShootBubbleLayer::onTouchBegan(Touch\* touch, Event\* event)  {  auto target = static\_cast<Sprite\*>(event->getCurrentTarget());  auto location = target->convertToNodeSpace(touch->getLocation());  shootBall(location);  return false;  } |

（4）射击泡泡

|  |
| --- |
| for(tempPoint = path->begin();tempPoint != path->end();tempPoint++)  {  currPoint=\*tempPoint;  float dx=currPoint.x-lastPoint.x;  float dy=currPoint.y-lastPoint.y;  float dis =(float)sqrt(dx\*dx+dy\*dy);  float moveT=0;  moveT = dis/SPEED/3;  if(countTemp==1)  {  MoveTo\* mtemp=MoveTo::create(moveT,\*tempPoint);  preAc=mtemp;  }  else if(countTemp>1)  {  MoveTo\* mtemp=MoveTo::create(moveT,\*tempPoint);  preAc=Sequence::createWithTwoActions(preAc,mtemp);  }  countTemp++;  lastPoint=currPoint;  } |

（5）更新位置

|  |
| --- |
| void ShootBubbleLayer::updateLocation()  {  for(int i=0;i<ARRAYSIZE;i++)  {  for(int j=0;j<ARRAYCOL;j++)  {  bLocation[i][j].x=bLocation[i][j].x;  bLocation[i][j].y=bLocation[i][j].y-R3S;  }  }  MAXHeight=MAXHeight-R3S;  } |

## 5.4游戏音乐及音效的实现

（1）音频背景音乐预处理

通过Cocos2d-x游戏的声音引擎获取实例，用到的类是SimpleAudioEnging对其预处理采用引擎中的方法preloadBackgroundMusic()。

（2）音频特效预处理

处理同背景音乐，采用的方法是preloadEffect()。

（3）音频的播放

采用的是引擎中的播放方法：pauseBackgroundMusic()和playEffect()。

（4）背景音乐继续、暂停与结束

采用的是引擎中的方法分别是：resumeBackgroundMusic()、 pauseBackgroundMusic() 和end()。

## 5.5游戏暂停的实现

（1）暂停对话框显示

|  |
| --- |
| // 创建一个背景  Sprite\* menuSprite= Sprite::create("pic/menu.png"); |

（2）暂停对话框回调

|  |
| --- |
| MenuItemImage\* jxyx = MenuItemImage::create(  "pic/jixup.png", //正常图片  "pic/jixu.png", //按下时的图片  CC\_CALLBACK\_0(ShootBubbleLayer::puaseCallback, this) //点击时被回调的方法  ); |

（3）游戏暂停

|  |
| --- |
| Director::getInstance()->pause(); |

## 5.6游戏核心的实现

### 5.6.1消除算法

（1）首先是泡泡位置的绘制，在AppMacros.h文件中定义了三关泡泡排列方式，在绘制过程中记录下泡泡的位置坐标。通过方法copyIsEmptyTemp()拷贝位置数组，通过drawBall()封装的图片精灵创建方法绘制泡泡。

（2）发射泡泡算法，将触摸点的横纵坐标记录在临时变量里，判断触摸点是否在发射位置之下，接着计算触点与发射位置纵坐标的增量从而计算出泡泡发射方向，以及计算出发射x，y的单位向量。

|  |
| --- |
| //获取触摸点位置  float touchX = location.x;  float touchY = location.y;  if(touchY<145){moveFlag=false;break;}  //获取当前球点位置  Point ballPosition = ballStart->getPosition();  float ballSX = ballPosition.x;  float ballSY = ballPosition.y;  //计算x,y方向的增量  float dx=touchX-ballSX;  float dy=touchY-ballSY;  //求出发射角度的弧度值  direction=(float)atan2(dy,dx);  //判断是否有枪膛，以便设置其方向  if(launcher!=NULL)  {  launcher->setRotation(90-direction/PI\*180);  }  //x,y方向的单位向量  vx=(float)(1\*sin(PI/2-direction));  vy=(float)(1\*cos(PI/2-direction)); |

（3）泡泡是否碰到其它泡泡，就是通过圆心距来判断，该方法将在下面的碰撞检测算法中介绍。

|  |
| --- |
| bool ShootBubbleLayer::calEndBall(); |

（4）计算泡泡运动路径path，取出其坐标，并利用其创建相应的移动动作对象，然后和上一动作两两组装，最后存储在一个动作上。其大致流程可以用图5.3来表示：



图5.3计算泡泡路径流程图

（5）通过递归计算发射的泡泡颜色与周围颜色相同的泡泡个数来消除。首先是搜索颜色相同的泡泡并记录个数，然后是遍历路径：

|  |
| --- |
| for(int i=0;i<ARRAYSIZE;i++)//遍历搜索路径的数组  {  for(int j=0;j<ARRAYCOL;j++)  {  Goneball[i][j]=0;  }  } |

最后消除泡泡以及没有附着点的泡泡

|  |
| --- |
| void ShootBubbleLayer::RMballDY3()  {  IsEmptyTemp[Row][Col]=0; //将泡泡运动到最终点的数组索引置为0  for(int i=0;i<ARRAYSIZE;i++)//遍历整个颜色索引数组  {  for(int j=0;j<ARRAYCOL;j++)  {  for(int i=0;i<ARRAYSIZE;i++)//遍历整个记录搜索路径的数组  {  for(int j=0;j<ARRAYCOL;j++)  {  Goneball[i][j]=0; //重置记录搜索路径  }  }  bool b=searchRoad(i,j); //搜索所有泡泡是否直接或间接连接到最上层泡泡  if(b)  {  continue;  }else  {  IsEmptyTemp[i][j]=0;  }  }  }  //遍历所有泡泡  for(int i=0;i<ARRAYSIZE;i++)  {  for(int j=0;j<ARRAYCOL;j++)  {  // 找出需要删除到球  if(IsEmptyTemp[i][j]!=IsEmpty[i][j])  {  //将其添加到删除球的向量中  RMball->push\_back(Point(i,j));  RMCountballs++;  }  }  }  RMball->push\_back(Point(Row,Col));  IsEmpty[Row][Col]=shootBallColor;  ballFalled();//消除的泡泡执行下落特效  copyIsEmptyTemp(); |

### 5.6.2碰撞检测

通过泡泡间的圆心距和泡泡直径比较得出是否碰撞。

|  |
| --- |
| bool ShootBubbleLayer::calEndBall()  {  int trow=-1;  int tcol=-1;  minJLBall=MAX\_VALUE; //初始化最小距离  //发射直线系数Ax+By+C=0,由参数方程x=cbx+vx\*t，y=cby+vy+t推导得出  float A=vy;  float B=-vx;  float C=vx\*cby-vy\*cbx;  for(int i=0;i<ARRAYSIZE;i++)  {  for(int j=0;j<ARRAYCOL;j++)//遍历地图泡泡中的所有泡泡，找碰撞最近的泡泡  {  if(IsEmptyTemp[i][j]==0)continue;  Point centerTemp=calCenter(i,j); //计算此列处圆心坐标  float dis= distance(A,B,C,centerTemp.x,centerTemp.y);  if(dis>=DR) //若圆心距距离大于直径则不碰撞  {  continue;  }  float dx=centerTemp.x-cbx; //求出发射泡泡与碰撞泡泡圆心坐标x之差  float dy=centerTemp.y-cby; //求出发射泡泡与碰撞泡泡圆心坐标y之差  //求出发射泡泡原始位置与被碰撞泡泡圆心距离  float L=(float)sqrt(dx\*dx+dy\*dy);  //求出发射泡泡原始位置圆心与被碰撞泡泡圆心到发射射线最近的距离  float cdis=(float)sqrt(L\*L-dis\*dis);  //求出被碰撞泡泡圆心距离发射直线最近点与碰撞时发射泡泡的圆心距离  float K=(float)sqrt(DR\*DR-dis\*dis);  //求出发射泡泡到碰撞时的运动距离  float jl=cdis-K;  if(jl<minJLBall)  {  trow=i;  tcol=j;  minJLBall=jl;  }  }  }  if(trow!=-1&&tcol!=-1)  {//计算碰撞时的圆心的横纵坐标  cbpx=cbx+vx\*minJLBall;  cbpy=cby+vy\*minJLBall;  path->push\_back(Point(cbpx,cbpy));  return true;  }  else  {  return false;  }  } |

### 5.6.3计时系统

游戏界面进入时开始计时通过 director->getScheduler()方法进行计时，获得到的时间显示在界面上通过FSLabel->setString(StringUtils::format("%d",timeScore));方式显示。

|  |
| --- |
| Director\* director = Director::getInstance();  auto sched = director->getScheduler(); |

# 6 泡泡龙游戏的测试

## 6.1测试方法介绍

游戏采用了黑盒测试，也称功能测试。它是通过测试来检测每个功能模块是否都能正常运行。测试中，在完全不考虑程序内部结构和内部特性的情况下，在程序接口进行测试，它只检查程序功能是否按照需求正常运行。黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试。

黑盒测试主要是以用户的角度，从输入与输出数据的对应关系进行测试的。测试可能会由于需求本身和测试外部主题设计的缺陷而造成测试不准确。[[[7]](#endnote-7)]

## 6.2测试项目

### 6.1.1菜单界面测试

表6.1是菜单界面的测试，主要是测试菜单项能否正确加载、点击菜单项后按钮是否会发生变化、点击菜单项后能否跳转到相应场景中。

表6.1 菜单界面测试

|  |
| --- |
| 测试编号：01 |
| 测试目的：  （1）测试菜单条目能否正确加载；  （2）测试点击是否有交互性变化；  （3）测试是否响应点击事件。 |
| 测试条件：运行平台为Android |
| 测试过程：游戏运行后，查看菜单界面并点击相应菜单项 |
| 期望结果：  （1）菜单项能正确加载；  （2）点击菜单项后按钮变为被点击状态；  （3）点击菜单项后能跳转到相应场景中。 |
| 测试结果：通过 |

### 6.1.2选关测试

表6.2 选关测试

|  |
| --- |
| 测试编号：02 |
| 测试目的：  （1）测试选关界面是否正常加载；  （2）测试点击是否有交互性变化；  （3）测试是否响应点击事件进入到对应的关卡。 |
| 测试条件：运行平台为Android |
| 测试过程：游戏运行后，查看选关界面并点击相应菜单项 |
| 期望结果：  （1）选关界面能正确加载；  （2）点击菜单项后按钮变为被点击状态；  （3）点击菜单项后能跳转到相应场景中。 |
| 测试结果：通过 |

### 6.1.3关卡测试

表6.3 关卡测试

|  |
| --- |
| 测试编号：03 |
| 测试目的：  （1）测试关卡界面是否正常加载；  （2）测试点击是否有交互性变化在开启音效与关闭情况下都进行测试；  （3）测试点击屏幕是否发射泡泡；  （4）测试泡泡是否更新；  （5）测试时间计时是否正常显示；  （6）测试倒计时柱和泡泡位置是否正常更新；  （7）测试泡泡是否正常碰撞和消除；  （8）测试暂停游戏时是否正常弹出暂停对话框，游戏界面是否停止，计时是否停止。 |
| 测试条件：运行平台为Android |
| 测试过程：游戏运行后，查看关卡界面并点击相应菜单项 |
| 期望结果：  （1）选关界面能正确加载；  （2）音效正常播放和停止；  （3）泡泡在指定区域发射不在发射平面下发射；  （4）发射后下一个泡泡正常随机生成；  （5）时间计时正常显示；  （6）倒计时柱和泡泡位置正常更新；  （7）泡泡正常碰撞和消除；  （8）对话框正常显示，游戏画面和计时正常停止。 |
| 测试结果：通过 |

### 6.1.4游戏设置测试

表6.4 音效设置测试

|  |
| --- |
| 测试编号：04 |
| 测试目的：  （1）测试点击是否有交互性变化；  （2）测试点击开关音效是否正常开启与关闭音效；  （3）测试在游戏中暂停后音效是否正常停止；  （4）测试游戏中继续游戏点击后是否继续播放音效；  （5）测试重新游戏和从其它界面进入主界面是否正常播放音效。 |
| 测试条件：运行平台为Android |
| 测试过程：游戏运行后，查看主界面并点击音效设置 |
| 期望结果：  （1）点击有交互性变化；  （2）点击开关音效正常开启与关闭音效；  （3）在游戏中暂停后音效正常停止；  （4）游戏中继续游戏点击后继续播放音效；  （5）重新游戏和从其它界面进入主界面正常播放音效。 |
| 测试结果：通过 |

### 6.1.5加载界面测试

表6.5 游戏加载界面测试

|  |
| --- |
| 测试编号：05 |
| 测试目的：  （1）测试游戏加载界面是否正常加载；  （2）测试加载完成后是否自动进入游戏菜单界面。 |
| 测试条件：运行平台为Android |
| 测试过程：游戏运行后，查看加载过程 |
| 期望结果：  （1）游戏加载界面正常加载；  （2）加载完成后自动进入游戏菜单界面。 |
| 测试结果：通过 |

### 6.1.6其它测试

表6.6 帮助界面翻页测试

|  |
| --- |
| 测试编号：06 |
| 测试目的：  （1）测试帮助界面是否正常加载；  （2）测试点击翻页按钮是否有对应交互性提示；  （3）测试点击翻页是否正常翻页。 |
| 测试条件：运行平台为Android |
| 测试过程：游戏运行后，点击主菜单帮助界面并操作翻页按钮 |
| 期望结果：  （1）帮助界面正常加载；  （2）点击翻页按钮有对应交互性提示；  （3）点击翻页正常翻页。 |
| 测试结果：通过 |

通过以上测试目前没有发现问题，达到了预期的测试效果。游戏显示运行正常，操作流畅。得出结论：测试合格。

## 6.3游戏帮助

### 6.2.1游戏背景

在蓝天、海洋和森林有一群阻挡你获取胜利的萌物泡泡，它们阻挡你前行，身为一个素质过硬的神枪手，你需要发射和他们同类的泡泡，但是有一点只有三个及以上相同颜色的泡泡才会解除咒语，打通通道。不要高兴太早，因为你的子弹是随机的哟，等待你的只有挑战。你，准备好了吗？

### 6.2.2游戏玩法

游戏共有三关，前两关是计时模式，玩家通过比较或者刷新游戏用时达到游戏目的；第三关是倒计时模式，在前两关的基础上加上了倒计时。是以时间压柱的形式体现的，在游戏过程中，不仅仅游戏计时在计时，游戏倒计时压柱还会随着时间的增长而往下移动，你要做的就是在压柱移动过程中，最下面的泡泡不碰及发射平面消除所有泡泡，才会成功过关。

# 7 结论

至此，已经基本完成了软件的开发。在此期间，遇到了形形色色的问题，有时为了解决一个问题，花费好几天也是常见。不过，经过自己的努力，基本上所发现的问题都得到了解决，虽然浪费了时间与精力，但收获也颇为丰富。在本部分中，将总结在开发过程中的心得体会和游戏的一些特色之处、克服的问题与不足之处以及可以更好优化的地方。

游戏特色：

本游戏有很多特色之处，如：

（1）为了节省内存运用纹理打包软件对纹理打包。

（2）为了减少泡泡渲染带来的消耗运用了批节点CCSpriteBatchNode。

（3）为了适配分辨率不用固定的参数而是运用Direct获取屏幕大小。

（4）为了减少数据库带来的开销运用引擎中CCUserDefault的XML文件存储游戏用时。

（5）游戏玩法多，分为计时模式与倒计时模式，简单易上手。

（6）游戏过程中增加了绚丽的特效，增加了游戏的趣味性。

克服的问题：

（1）Cocos2d-x游戏引擎在Eclipse下开发所需要搭建很复杂的运行环境。首先要配置JAVA和安卓的开发环境，由于采用的编程语言是C++，所以还要搭建交叉编译环境，在此期间遇到了C++中形如Code Analysis报错的问题，这是由于Eclipse代码格式和C++的不同所导致的，在Eclipse — “Window” — “preferences” — “C/C++” — “Code Analysis”中选择“Use Project Setting”选项，并将列出的错误选项取消选中，“Apply” —“OK”即可。此时在打开jni中的代码就不会报错了。

（2）游戏中要学会看Cocos2d-x的开发文档，我在Cocos2d官网参阅了官方的开发API并参考了相关开发Demo[[[8]](#endnote-8)]。

（3）刚开始遇到了游戏崩溃的问题，代码是没问题，最后发现是Cocos2d-x对内存的管理还不是很成熟，所以要我们自己调用retain和release函数对内存进行管理。例如在创建Cocos2d-x中的数组CCArray后，需要调用一次retain函数才能正确使用内存，不然程序会崩溃。

（4）运用Cocos2d-x游戏引擎进行开发就必须遵守游戏引擎的思想，比如导演、场景、层、精灵的概念，只有导演才能切换场景，层只能加到场景上，精灵只能加到层上。正因为这样，导致需要写很多类。刚开始的时候没太重视这个部分，没对游戏进行这些概念的区分，造成不能切换游戏场景等错误。之后对这些内容进行重定义才恢复正常。

不足和优化之处：

由于不是美工出身，游戏中素材是从网络上上架的App中提取出来的图片并作修改使用在本游戏中的。游戏关卡还很少，本游戏只是提供了经典玩法，耐玩性差，如果做成很多关并添加解锁功能，如果能够添加失败将会损失生命值，通过某个时间段“爱心”会自增，只有在有爱心的前提下才能进行闯关的话将会使游戏可玩性大大提高。

# 参考文献

# 英文原文

**Android Components and the Need for Multiprocessing**

Before we immerse ourselves in the world of threading, we will start with an introduction to the Android platform, the application architecture, and the application’s execution. This chapter provides a baseline of knowledge required for an effective discussion of threading in the rest of the book, but a complete information on the Android platform can be found in the official documentation or in most of the numerous Android programming books on the market.

**Android Software Stack**

Applications run on top of a software stack that is based on a Linux kernel, native C/C++ libraries, and a runtime that executes the application code (Figure 1-1).

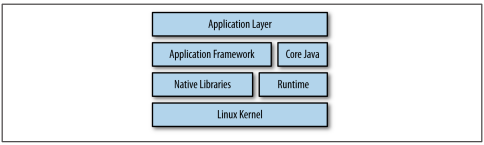


Figure 1-1. Android software stack

The major building blocks of the Android software stack are:

**Applications**

Android applications that are implemented in Java. They utilize both Java and Android framework libraries.

**Core Java**

The core Java libraries used by applications and the application framework. It is not a fully compliant Java SE or ME implementation, but a subset of the retired Apache Harmony implementation, based on Java 5. It provides the fundamental Java threading mechanisms: the java.lang.Thread class and java.util.concurrent package.

**Application framework**

The Android classes that handle the window system, UI toolkit, resources, and soon—basically everything that is required to write an Android application in Java. The framework defines and manages the lifecycles of the Android components and their intercommunication. Furthermore, it defines a set of Android-specific asynchronous mechanisms that applications can utilize to simplify the thread management: HandlerThread, AsyncTask, IntentService, AsyncQueryHandler, and Loaders. All these mechanisms will be described in this book.

**Native libraries**

C/C++ libraries that handle graphics, media, database, fonts, OpenGL, etc. Java applications normally don’t interact directly with the native libraries because the Application framework provides Java wrappers for the native code.

**Runtime**

Sandboxed runtime environment that executes compiled Android application code in a virtual machine, with an internal byte code representation. Every application executes in its own runtime, either Dalvik or ART (Android Runtime). The latter was added in KitKat (API level 19) as an optional runtime that can be enabled by the user, but Dalvik is the default runtime at the time of writing.

**Linux kernel**

Underlying operating system that allows applications to use the hardware functions of the device: sound, network, camera, etc. It also manages processes and threads. A process is started for every application, and every process holds a runtime with a running application. Within the process, multiple threads can execute the application code. The kernel splits the available CPU execution time for processes and their threads through scheduling

**Application Architecture**

The cornerstones of an application are the Application object and the Android components: Activity, Service, BroadcastReceiver, and ContentProvider.

**Application**

The representation of an executing application in Java is the android.app.Application object, which is instantiated upon application start and destroyed when the application stops (i.e., an instance of the Application class lasts for the lifetime of the Linux process of the application). When the process is terminated and restarted, a new Application instance is created.

**Components**

The fundamental pieces of an Android application are the components managed by the runtime: Activity, Service, BroadcastReceiver, and ContentProvider. The configuration and interaction of these components define the application’s behavior. These entities have different responsibilities and lifecycles, but they all represent application entry points, where the application can be started. Once a component is started, it can trigger another component, and so on, throughout the application’s lifecycle. A component is trigged to start with an Intent, either within the application or between applications. The Intent specifies actions for the receiver to act upon—for instance, sending an email or taking a photograph—and can also provide data from the sender to the receiver. An Intent can be explicit or implicit:

*Explicit Intent*

Defines the fully classified name of the component, which is known within the application at compile time.

*Implicit Intent*

A runtime binding to a component that has defined a set of characteristics in an IntentFilter. If the Intent matches the characteristics of a component’s Intent Filter, the component can be started.

Components and their lifecycles are Android-specific terminologies, and they are not directly matched by the underlying Java objects. A Java object can outlive its component, and the runtime can contain multiple Java objects related to the same live component. This is a source of confusion, and as we will see in Chapter 6, it poses a risk for memory leaks.

An application implements a component by subclassing it, and all components in an application must be registered in the AndroidManifest.xml file.

**Activity**

An Activity is a screen—almost always taking up the device’s full screen—shown to the user. It displays information, handles user input, and so on. It contains the UI components—buttons, texts, images, and so forth—shown on the screen and holds an object reference to the view hierarchy with all the View instances. Hence, the memory footprint of an Activity can grow large.

When the user navigates between screens, Activity instances form a stack. Navigation to a new screen pushes an Activity to the stack, whereas backward navigation causes a corresponding pop.

In Figure 1-2, the user has started an initial Activity A and navigated to B while A was finished, then on to C and D. A, B, and C are full-screen, but D covers only a part of the display. Thus, A is destroyed, B is totally obscured, C is partly shown, and D is fully shown at the top of the stack. Hence, D has focus and receives user input. The position in the stack determines the state of each Activity:

• Active in the foreground: D

• Paused and partly visible: C

• Stopped and invisible: B

• Inactive and destroyed: A

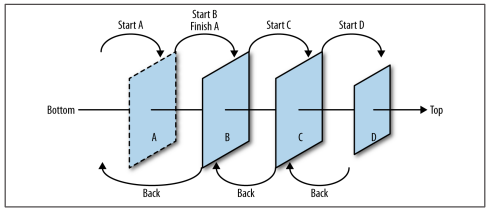


Figure 1-2. Activity stack

The state of an application’s topmost Activity has an impact on the application’s system priority—also known as process rank—which in turn affects both the chances of terminating an application (“Application termination” on page 7) and the scheduled execution time of the application threads (Chapter 3).

An Activity lifecycle ends either when the user navigates back—for example, presses the back button—or when the Activity explicitly calls finish(). **Service**

A Service can execute invisibly in the background without direct user interaction. It is typically used to offload execution from other components, when the operations can outlive their lifetime. A Service can be executed in either a started or a bound mode:

*Started Service*

The Service is started with a call to Context.startService(Intent) with an explicit or implicit Intent. It terminates when Context.stopService(Intent) is called.

*Bound Service*

Multiple components can bind to a Service through Context.bindService(Intent, ServiceConnection, int) with explicit or implicit Intent parameters. After the binding, a component can interact with the Service through the Service Connection interface, and it unbinds from the Service through Context.unbindService(ServiceConnection). When the last component unbinds from the Service, it is destroyed.

**ContentProvider**

An application that wants to share substantial amounts of data within or between applications can utilize a ContentProvider. It can provide access to any data source, but it is most commonly used in collaboration with SQLite databases, which are always private to an application. With the help of a ContentProvider, an application can publish that data to applications that execute in remote processes.

**BroadcastReceiver**

This component has a very restricted function: it listens for intents sent from within the application, remote applications, or the platform. It filters incoming intents to determine which ones are sent to the BroadcastReceiver. A BroadcastReceiver should be registered dynamically when you want to start listening for intents, and unregistered when it stops listening. If it is statically registered in the AndroidManifest, it listens for intents while the application is installed. Thus, the BroadcastReceiver can start its associated application if an Intent matches the filter

**Application Execution**

Android is a multiuser, multitasking system that can run multiple applications at the same time and let the user switch between applications without noticing a significant delay. The Linux kernel handles the multitasking, and application execution is based on Linux processes.

**Linux processes**

Linux assigns every user a unique user ID, basically a number tracked by the OS to keep the users apart. Every user has access to private resources protected by permissions, and no user (except root, the super user, which does not concern us here) can access another user’s private resources. Thus, sandboxes are created to isolate users. In Android, every application package has a unique user ID; for example, an application in Android corresponds to a unique user in Linux and cannot access other applications’ resources. What Android adds to each process is a runtime execution environment, such as the Dalvik virtual machine, for each instance of an application. Figure 1-3 shows the relationship between the Linux process model, the VM, and the application.

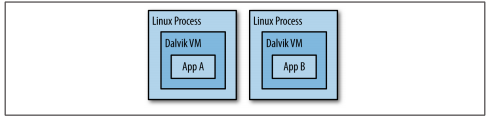


Figure 1-3. Applications execute in different processes and VMs

By default, applications and processes have a one-to-one relationship, but if required, it is possible for an application to run in several processes, or for several applications to run in the same process.

**Lifecycle**

The application lifecycle is encapsulated within its Linux process, which, in Java, maps to the android.app.Application class. The Application object for each app starts when the runtime calls its onCreate() method. Ideally, the app terminates with a call by the runtime to its onTerminate(), but an application cannot rely upon this. The underlying Linux process may have been killed before the runtime had a chance to call onTerminate(). The Application object is the first component to be instantiated in a process and the last to be destroyed.

**Application start**

An application is started when one of its components is initiated for execution. Any component can be the entry point for the application, and once the first component is triggered to start, a Linux process is started—unless it is already running—leading to the following startup sequence:

①. Start Linux process.②. Create runtime.

③. Create Application instance.

④. Create the entry point component for the application.

Setting up a new Linux process and the runtime is not an instantaneous operation. It can degrade performance and have a noticeable impact on the user experience. Thus, the system tries to shorten the startup time for Android applications by starting a special process called Zygote on system boot. Zygote has the entire set of core libraries preloaded. New application processes are forked from the Zygote process without copying the core libraries, which are shared across all applications.

**Application termination**

A process is created at the start of the application and finishes when the system wants to free up resources. Because a user may request an application at any later time, the runtime avoids destroying all its resources until the number of live applications leads to an actual shortage of resources across the system. Hence, an application isn’t automatically terminated even when all of its components have been destroyed. When the system is low on resources, it’s up to the runtime to decide which process should be killed. To make this decision, the system imposes a ranking on each process depending on the application’s visibility and the components that are currently executing. In the following ranking, the bottom-ranked processes are forced to quit before the higher-ranked ones. With the highest first, the process ranks are:

*Foreground*

Application has a visible component in front, Service is bound to an Activity in front in a remote process, or BroadcastReceiver is running.

*Visible*

Application has a visible component but is partly obscured.

*Service*

Service is executing in the background and is not tied to a visible component.

*Background*

A nonvisible Activity. This is the process level that contains most applications.

*Empty*

A process without active components. Empty processes are kept around to improve startup times, but they are the first to be terminated when the system reclaims resources.

In practice, the ranking system ensures that no visible applications will be terminated by the platform when it runs out of resources.

|  |
| --- |
| **Lifecycles of Two Interacting Applications**  This example illustrates the lifecycles of two processes, P1 and P2, that interact in a typical way (Figure 1-4). P1 is a client application that invokes a Service in P2, a server application. The client process, P1, starts when it is triggered by a broadcasted Intent. At startup, the process starts both a BroadcastReceiver and the Application instance. After a while, an Activity is started, and during all of this time, P1 has the highest possible process rank: Foreground.    Figure 1-4. Client application starts Service in other process  The Activity offloads work to a Service that runs in process P2, which starts the Service and the associated Application instance. Therefore, the application has split the work into two different processes. The P1 Activity can terminate while the P2 Service keeps running.  Once all components have finished—the user has navigated back from the Activity in P1, and the Service in P2 is asked by some other process or the runtime to stop—both processes are ranked as empty, making them plausible candidates for termination by the system when it requires resources.  A detailed list of the process ranks during the execution appears in Table 1-1. |

It should be noted that there is a difference between the actual application lifecycle— defined by the Linux process—and the perceived application lifecycle. The system can have multiple application processes running even while the user perceives them as terminated. The empty processes are lingering—if system resources permit it—to shorten the startup time on restarts.

**Structuring Applications for Performance**

Android devices are multiprocessor systems that can run multiple operations simultaneously, but it is up to each application to ensure that operations can be partitioned and executed concurrently to optimize application performance. If the application doesn’t enable partitioned operations but prefers to run everything as one long operation, it can exploit only one CPU, leading to suboptimal performance. Unpartitioned operations must run synchronously, whereas partitioned operations can run asynchronously. With asynchronous operations, the system can share the execution among multiple CPUs and therefore increase throughput.

An application with multiple independent tasks should be structured to utilize asynchronous execution. One approach is to split application execution into several processes, because those can run concurrently. However, every process allocates memory for its own substantial resources, so the execution of an application in multiple processes will use more memory than an application in one process. Furthermore, starting and communicating between processes is slow, and not an efficient way of achieving asynchronous execution. Multiple processes may still be a valid design, but that decision should be independent of performance. To achieve higher throughput and better performance, an application should utilize multiple threads within each process.

**Creating Responsive Applications Through Threads**

An application can utilize asynchronous execution on multiple CPU’s with high throughput, but that doesn’t guarantee a responsive application. Responsiveness is the way the user perceives the application during interaction: that the UI responds quickly to button clicks, smooth animations, etc. Basically, performance from the perspective of the user experienced is determined by how fast the application can update the UI components. The responsibility for updating the UI components lies with the UI thread, which is the only thread the system allows to update UI components.

To make the application responsive, it should ensure that no long-running tasks are executed on the UI thread. If they do, all the other execution on that thread will be delayed. Typically, the first symptom of executing long-running tasks on the UI thread is that the UI becomes unresponsive because it is not allowed to update the screen or accept user button presses properly. If the application delays the UI thread too long, typically 5-10 seconds, the runtime displays an “Application Not Responding” (ANR) dialog to the user, giving her an option to close the application. Clearly, you want to avoid this. In fact, the runtime prohibits certain time-consuming operations, such as network downloads, from running on the UI thread.

So, long operations should be handled on a background thread. Long-running tasks typically include:

• Network communication

• Reading or writing to a file

• Creating, deleting, and updating elements in databases

• Reading or writing to SharedPreferences

• Image processing

• Text parsing

|  |
| --- |
| **What Is a Long Task?**  There is no fixed definition of a long task or a clear indication when a task should execute on a background thread, but as soon as a user perceives a lagging UI—for example, slow button feedback and stuttering animations—it is a signal that the task is too long to run on the UI thread. Typically, animations are a lot more sensitive to competing tasks on the UI thread than button clicks, because the human brain is a bit vague about when a screen touch actually happened. Hence, let us do some coarse reasoning with animations as the most demanding use case.  Animations are updated in an event loop where every event updates the animation with one frame, i.e., one drawing cycle. The more drawing cycles that can be executed pertime frame, the better the animation is perceived. If the goal is to do 60 drawing cycles per second—a.k.a. frames per second (fps)—every frame has to render within 16 ms. If another task is running on the UI thread simultaneously, both the drawing cycle and the secondary task have to finish within 16 ms to avoid a stuttering animation. Consequently, a task may require less than 16 ms execution time and still be considered long.  The example and calculations are coarse and meant as an indication of how an application’s responsiveness can be affected not only by network connections that last for several seconds, but also tasks that at first glance look harmless. Bottlenecks in your application can hide anywhere. |

Threads in Android applications are as fundamental as any of the component building blocks. All Android components and system callbacks—unless denoted otherwise—run on the UI thread and should use background threads when executing longer tasks.

**Summary**

An Android application runs on top of a Linux OS in a Dalvik runtime, which is contained in a Linux process. Android applies a process-ranking system that priorities the importance of each running application to ensure that it is only the least prioritized applications that are terminated. To increase performance, an application should split operations among several threads so that the code is executed concurrently. Every Linux process contains a specific thread that is responsible for updating the UI. All long operations should be kept off the UI thread and executed on other threads.

# 中文译文

**Android组件和多进程处理的需要**

在我们开始讲述Android进程之前先来了解一下Android的应用架构和应用程序的执行过程。本章介绍的线程以本书所涉及的知识为基准，您可以通过Android平台上的官方文档或大多数市场上Android众多的编程书籍中找到完整的信息。

Android架构

应用程序上运行的软件，是基于Linux内核，原生的C / C + +库，和一个运行时执行应用程序代码（图1.1）



图1.1 Android架构

Android软件架构的主要模块有：

应用层

Android应用程序是由Java实现的，它们都使用Java库和Android库。

Java核心

应用程序和应用程序框架都使用Java核心库。它不是完全兼容的Java SE或ME的，但一部分是Apache实现的子集，是基于Java 5实现的。它提供了Java的基础线程机制java.lang.thread和 java.util.concurrent类包。

应用框架

Android的处理窗口系统、UI工具包、资源等基本上都是要求用Java写在Android应用层的。该框架定义和管理了Android组件的生命周期和它们之间的联系。此外，它定义了一套Android特有的异步机制，应用程序可以利用简化线程管理：handlerthread，AsyncTask，intentservice，asyncqueryhandler和装载机。所有这些机制将在本书中一一描述。

本地库

C / C + +库包含了图形处理、媒体、数据库、字体、OpenGL等。Java应用程序通常不直接与本地库相互作用，因为应用程序框架提供了Java包装的原生代码。

运行时

执行编译Android应用程序代码的是在沙箱运行环境虚拟机中进行的，虚拟机有一个内部的字节码。无论是Dalvik或ART模式（Android Runtime），每一个应用程序都执行自己的运行时，ART模式是后来在KitKat（API Level 19）作为一个用户可选的运行时，但Dalvik是在写入的时候默认的运行时。

Linux内核

底层的操作系统。它允许应用程序使用的硬件功能有：声音、网络、摄像头等，它也管理应用程序的进程和线程。一个过程的开始为每一个应用程序，每个进程分配一个运行时库。一个正在运行的应用程序，在这个过程中，多个线程可以执行应用程序代码。内核将可用的CPU执行时间对进程和线程进行分配和调度。

应用程序体系结构

一个应用程序的基石是应用对象和Android四大组件：Activity、Service、ContentProvider和BroadcastReceiver。

应用对象

一个正在执行的应用程序在Java中的表示是android.app.application对象被实例化，应用程序启动和破坏时，应用程序停止（即，应用程序的类的一个实例持续应用的Linux进程的寿命）。当进程终止并重启，即创造了一个新的应用实例。

组件

一个Android应用程序的基本的组件有Activity、Service、ContentProvider和BroadcastReceiver。这些组件的配置和交互定义了应用程序的行为。它们的实体有不同的职责和生命周期，但他们都表示应用的入口点，其中应用程序可以开始。一旦一个部件开始，它可以触发另一个组件等等，在整个应用程序的生命周期。一个元件被触发启动的意图，无论是在应用程序或应用程序之间。意图指定接收者行事，比如行动、发送电子邮件或拍照，可以从发送者到服务提供数据。

对intent，一个intent可以分为显式或隐式意图：

显式意图：

通过指定Intent组件名称来实现的，它一般用在知道目标组件名称的前提下，一般是在相同的应用程序内部实现的

隐式意图：

运行时绑定到一个组件，定义了一套特征在IntentFilter（意图过滤器）。如果意图匹配组件的意图的特性符合IntentFilter，该组件就可以启动。

组件和它们的生命周期是Android特有的术语，他们不直接匹配底层的Java对象，一个Java对象的分量和运行库可以包含多个Java对象，这和相同的组件相关。我们将在6章看到，它对内存的泄漏有相应描述。

应用程序所有组件通过继承它的父组件实现，应用程序必须在AndroidManifest.xml文件中注册。

Activity

一个Activity就是一个屏幕——几乎占用设备的全屏。它显示用户的信息、处理用户的输入等等，诸如此类。它包含的UI组件——按钮、文本、图像等等，显示在屏幕上，并持有一个对象的应用架构优先视图层次的所有视图的实例。因此，一个活动的内存占用会变得很大。

当用户界面之间导航，活动实例形成堆栈。导航到一个新的Activity时该Activity到活动栈，在它下面的栈将会弹出。

在图1.2中，用户已经开始的初始活性Activity A导航到B而被完成，然后从C到D，A，B，和C是全屏幕，但D只覆盖了他们的一部分。因此，在堆栈中A被销毁，B是完全遮蔽，C被部分示出， D是在堆栈的顶部完全显示。因此，D接收用户输入焦点。在堆栈中的位置决定了每个活动的状态。

前台：D

暂停，部分可见：C

停止，看不见的：B

非活动，销毁的：A



图1.2 Activity状态图

应用程序的最顶层的活动状态受应用程序的系统优先级的影响，也就是过程秩进，它受两方面因素的影响：终止应用程序（第7页的“应用程序终止”）和应用程序的安排执行时间结束（第3章）。

一个Activity生命周期的结束不但由于用户的返回，比如当用户点击返回按钮。或者Activity明确调用了finish()方法。

Service

一个Service 是一段长生命周期的，没有用户界面的程序，可以用来开发如监控类程序。一个服务可以在显式模式下或绑定模式下执行：

显示模式

服务是以调用Context.startService(Intent)显式或隐式调用开始的，终止是调用Context.stopService(Intent)。

绑定模式

使用bindService()启动的服务与调用者绑定,只要调用者关闭服务就终止,使用此方法启动时,服务首次启动系统先调用服务的onCreate()-->onBind(),如果服务已经启动再次调用不会再触发这2个方法,调用者退出时系统会调用服务的onUnbind()-->onDestory(),想主动解除绑定可使用Contex.unbindService(),系统依次调用onUnbind()-->onDestory();

ContentProvider

一个应用程序想要分享内容或应用程序间的数据大量交互可以利用一个Content Provider。它可以提供访问任何数据源，但这是最常用的与SQLite数据库合作，总是私人申请。在Content Provider的帮助下，一个应用可以发布数据，执行远程过程的应用。

BroadcastReceiver

该组件具有非常有限的功能：它侦听来自内部的意图应用、远程应用程序或平台。它过滤来电的意图来确定哪些是发送到广播。一个BroadcastReceiver应该动态注册，当你想开始监听意图时监听。如果是在AndroidManifest.xml静态注册，它在安装应用程序时监听意图。因此，BroadcastReceiver可以开始相关如果意图相匹配的过滤器可以匹配的应用程序。

应用程序执行

Android是一个多用户，多任务系统，可以同时运行多个应用程序，同时，让应用程序之间切换的用户没有注意到一个显著的延迟。Linux内核的处理多任务，并执行应用程序是基于Linux进程。

Linux内核

Linux分配每个用户唯一的用户ID，基本上都是一些由OS跟踪，保持了用户的独立。每个用户都有访问由权限保护的私有资源，没有用户（除了根，超级用户，其它）可以访问其他用户的私人资源。因此，在Android中创建沙箱隔离用户，每个应用程序包都有一个唯一的用户ID;例如，在Android的应用程序对应于Linux的一个唯一的用户而不能访问其他应用程序的资源。

Android增加了每个进程运行时的执行环境，如Dalvik虚拟机，为每个应用程序实例。图1.3显示了Linux进程之间的关系模型、虚拟机、以及应用。



图1.3应用程序在不同进程和虚拟机中

默认情况下，应用程序和进程有一个一对一的关系，但如果需要的话，一个应用程序运行在几个过程或几个应用程序在同一个进程中运行是可能的。

生命周期

应用程序生命周期封装在Linux进程之中，而在Java中，映射到android.app.Application类封装。每个应用程序的应用对象开始时调用它的onCreate()方法。理想情况下，应用程序将终止与呼叫运行时其onTerminate()，但应用程序不能依靠这一点。该底层Linux进程可能在打电话给onTerminate()之前已被杀死运行时，最后一个被销毁。

程序启动

它的组件之一是开始执行时这个应用程序就启动了。任何组件可以被应用为切入点，一旦第一个组件触发Linux进程就开始触发，除非它已经导致下面的启动顺序：

1. Linux进程启动
2. 创建运行时
3. 创建应用实例
4. 创建应用程序组件

建立一个新的Linux进程和运行不是瞬时的操作。它会降低性能，会影响用户体验。因此，该系统试图通过启动一个名为合子的系统启动特殊进程，为Android应用缩短启动时间。合子有一整套核心库的预装，新的应用程序进程是从合子的过程分叉而不复制的核心库，这是在所有应用程序所共享的。

应用程序终止

在应用和结束时，系统要释放资源。因为用户可以在任何的时间请求一个应用程序，运行时避免破坏它的所有资源，直到活动应用程序的数量消耗掉整个系统的资源。因此，应用程序不会自动终止，即使所有的组件都被摧毁终止。

当系统资源不足时，就由运行时来决定哪个进程应该被杀死。做出这个决定时，这取决于系统规定的应用程序的可见性，每个进程和当前正在执行的组件优先级。凭借优先级，系统进程栈是：

前台

应用程序作为一个可视化组件已经在前面，服务绑定到一个活动前面远程进程，或BroadcastReceiver的运行。

可见

应用程序有一个可见的组成部分，但部分遮蔽

服务

执行服务时在后台，并且不依赖于有形成分

后台

一个不可见的活动，这是包含大多数应用进程级。

空

一个进程没有任何组件，空的进程被提升，当系统回收资源时首先将被销毁。

在实践中，优先级系统确保没有可见的应用程序将由平台耗尽资源时终止。

**两种交互应用的生命周期**

这个例子说明了两个过程的生命周期，P1和P2，在一个典型的方式（图1-4）进行交互。P1是在P2中调用一个服务的客户端应用，客户端进程，P1开始时它是由一广播意图启动触发，在启动时，该过程开始有一个BroadcastReceiver和应用实例，过了一会儿，一个活动的开始，在这段时间，P1有最高排名：前台进程

图1.4在其他进程中客户端程序启动服务

在运行过程中的P2，活动卸载到一个服务，开始的服务及相关的应用实例。因此，应用程序工作拆分成两个不同的过程， P2在服务运行过程中P1的Activity可以终止。

一旦所有组件已经完成，在P1 Activity用户导航返回同时P2服务的有些进程和运行时被终止，这两个过程都被列为空，使他们被系统终止时，它需要资源的合理候选。

在执行过程中出现的优先级清单如下表1-1：

|  |  |  |
| --- | --- | --- |
| 程序状态 | P1进程级 | P2进程级 |
| P1从广播启动 | 前台 | N/A |
| P1启动Activity | 前台 | N/A |
| P1在P2服务下启动 | 前台 | 前台 |
| P1 Activity销毁 | 空 | 服务 |
| P2 服务停止 | 空 | 空 |

应当指出的是，由Linux流程中定义的实际应用生命周期和感知应用程序生命周期之间的差。该系统可在运行即使用户认为他们作为终止多个应用程序进程。空过程将持续保留，如果系统资源允许它缩短了启动时间上重新启动。

通过线程创建响应应用程序

应用程序可以异步执行使用多个CPU上高吞吐量的数据,但这并不能保证及时响应应用程序。响应性是用户感知应用程序在交互方式:用户界面响应迅速单击按钮,平滑的动画,等等。基本上,从性能构建应用程序的性能。也被称为主线程,但是在本书中我们坚持称之“UI的主线程”。用户体验是应用程序可以更新UI组件的速度决定的。更新UI组件的责任在于UI线程,这是唯一允许更新UI组件的线程系统。

为了使应用程序响应，应该确保没有长时间运行的任务在UI线程上执行。如果都在一条线程上，则在该线程的其他执行就会被推迟。通常情况下，UI线程上执行时间长的任务表现是UI变得反应迟钝，因为它不允许正常更新屏幕或接受用户按键。如果应用程序延迟UI线程太长，一般5-10秒，运行时将显示一个“应用程序没有响应”（ANR）对话框给用户，给她选项关闭应用程序。显然，要避免这种情况。事实上，运行时禁止某些耗时的操作，如网络下载，从UI线程上运行。

因此，耗时操作应在后台线程来处理。长时间运行的任务通常包括：

* 网络通讯
* 读写操作
* 删除和更新数据库中的元素
* 创建、读或写SharedPreferences
* 影像处理
* 文本解析

**什么是一个长期的任务？**

即当任务应该在后台线程执行，作为用户感知滞后的用户界面。例如，缓慢的按键反馈执行，但只要清楚地表明没有固定的定义动画，即任务是太长UI线程上运行的就是长期任务。通常情况下，动画相比按钮点击UI线程任务的竞争更加敏感，当触摸屏幕时，人的大脑是有点含糊。因此，我们可以做一些粗糙动画去延迟操作。

动画是在一个事件循环中每一帧的更新，也就是说，一个图形循环更新。如果目标是做每秒60帧的绘图，则每一帧必须在16毫秒渲染出来。另一个任务是在UI线程上同时运行，同时绘制周期和二级任务必须完成在16毫秒以避免口吃动画。因此，一个任务可能需要不到16毫秒的执行时间，仍然被认为是长时间。

任何组件块在Android应用程序线程是最基础的。所有的Android组件和系统的回调，在表示UI线程上、运行和执行长任务时，应使用后台线程。

总结

Android应用程序在Linux操作系统上的Dalvik运行时，即被包含在Linux进程中运行。Android运行在一个进程优先级系统之上。为了提高性能，应用程序应当被拆分为使代码能够被同时执行在多个线程之间的操作。每个Linux过程包含一个特定的线程，负责更新UI。所有耗时操作应在UI线程中不能影响其他线程的执行。

# 致 谢

经过了几个月的努力，毕业设计已经完成，这也意味着在未来不久的日子里将会为我的大学本科生活画上一个句号。转瞬间，大学四年转眼即逝，老师们的谆谆教诲，父母的支持，学长学姐的帮助，同学们的一直陪伴，不仅让我的知识得到了丰富，而且也让我慢慢走向了成熟。

首先，我要感谢我的导师谢红侠老师。在实习的这段时间里，谢红侠老师为我提供了相关的资料，敦促我完成相关任务，耐心的解释和审阅毕业设计论文的每一个细节，提出很多可以更好在开发过程中解决问题的方案，使我准时完成了任务书、开题报告和论文的撰写。在此我非常感谢谢红侠老师无私地抽出时间对我进行细心指导，帮助我进行论文的修改和完善。再次对谢红侠老师表示诚挚的感谢！

其次，我要感谢大学期间所有的老师，所有的同学以及指导员。我的大学生活因你们而充实美好。

最后，我要感谢我的父母，在背后是你们辛苦的教育与无私的支持，让我有机会完成大学本科的学业，成为对社会有用的人。

1. [] 马朔, 尹航. Cocos2d-x高级开发教程:制作自己的捕鱼达人[M]. 北京:北京邮电出版社, 2013 [↑](#endnote-ref-1)
2. [] 高璐, 马玉志. 浅谈Microsoft Visual Studio 2010新特性[J]. 黑龙江科技信息, 2010(32):96-96 [↑](#endnote-ref-2)
3. [] 刘洪星, 谢玉山. Eclipse开发平台及其应用[N]. 武汉理工大学学报(信息与管理工程版), 2005(02) [↑](#endnote-ref-3)
4. [] Ducrohet, Xavier. Android 2.2 and developers goodies.. Android Developers Blog. Google. [2010-05-20] [↑](#endnote-ref-4)
5. [] Perchat J, Desertot M, Lecomte S. Component Based Framework to Create Mobile Cross-platform Applications[J]. procedia computer science, 2013(19):1004-1011 [↑](#endnote-ref-5)
6. []http://baike.baidu.com/redirect/0a4amZ8XtzVq6ZkCDH-OCODRADnslxF3F5clZiNUpC9OnisE1lcYhAViGsDEeqXajCEooDmzzXizSwf8PwazFCc. Photoshop的由来 江苏省邗江中学[2015-05-21] [↑](#endnote-ref-6)
7. [7] 万年红,李翔.软件黑盒测试的方法与实践：《计算机工程》,2000年12期 [↑](#endnote-ref-7)
8. []http://www.cocos2d-x.org/wiki/Cocos2d-x. Cocos2d-x 官方API

   [9] 吴亚峰,闫敏,于复兴.《Cocos2d-x 3.0 游戏开发实战详解》 2014. 人民邮电出版社

   [10] 钟迪龙. 《Cocos2d-x 3.x 游戏开发之旅》 2014. 电子工业出版社

   [11] 满硕泉. 《Cocos2d-x 权威指南》 2014. 机械工程出版社

   [12] 爱范儿.《Android和IOS市场占有率》 2014. 新浪手机网

   [13] 史济民,顾春华,郑红 《软件工程——原理、方法与应用》 2009. 高等教育出版社

   [14] （美）AlanDennis,BarbarraHaleyWixom,RobertaM. 《系统分析与设计》 2010. 人民邮电出版社

   [15] 魏延伟 黑白盒测试系统的设计与研究. 学位论文 2013

   [16] 孙勇 《现代软件工程》 2011. 北京希望电子出版社

   [17] 梁春燕 《Visual Studios 2010并行编程从入门到精通》 2012. 清华大学出版社

   [18] 童天 软件可维护性的方法和作用 2012. 大观周刊

   [19] cstriker1407 Cocos2dx-3.x版本Director学习笔记 2014. 博客

   [20] http://blog.csdn.net/hewei0241/article/details/24882503 2009. TexturePacker非常棒的游戏资源图像处理工具. Vigor. 2014 博客 [↑](#endnote-ref-8)