FLASH 抛投类手机游戏的设计与实现

摘 要

随着 ADOBE AIR 跨平台运行时的日趋流行，应用 ActionScript3.0 开发的 FLASH APP 在各类手持设备上得到了飞速的发展。而 BOX2D 作为一款性能优越的 C++开源物理引擎，也移植到了 FLASH 平台上。在 FLASH 开发的众多手机游戏门类中，带有物理引擎的抛投类游戏正在逐渐成为一个闪亮的分支，流行于 Android、iOS 等主流移动操作系统的手持设备上。然而直接应用 FLASH BOX2D 物理引擎开发抛投类手机游戏，在物理抛投模拟，物体碰撞破碎，游戏场景编辑等方面都存在不小的开发难度，而且在开发的时间成本，游戏的运行效率等方面都遇到了不小的挑战。基于开发效率与游戏稳定性，在游戏开发中就很有必要进行总结归纳，进而创新出一套轻量级的，易于扩展的，简单易用的 FLASH 抛投类手机游戏开发架构与设计。

FLASH 抛投类手机游戏的基础思路是利用 BOX2D 物理引擎的强大功能，实现游戏中相应物体的多边形碰撞盒绘制。同时在普通抛投功能的基础上，应用物理几何理论知识，对抛投轨迹的多样性，趣味性进行设计与实现。对于碰撞破碎策略，则根据游戏实际应用的需求，综合考虑性能与开发复杂度的基础上给出合理的可选择方案。另外也充分考虑了游戏调试的便捷性与操控的多样性，对抛投类手机游戏的场景编辑器作了详细的设计与实现。而整体的框架设计则充分运用组件封装的概念与设计模式的思想，力求对 FLASH BOX2D 进行进一步的封装，以达到简单易用而又的易于扩展目的。

论文通过对 FLASH 抛投类手机游戏的详细分析，归纳了基于 FLASH BOX2D 物理引擎进行手机游戏开发的优势，并指出了开发过程中存在的各种问题。在对 FLASH BOX2D 技术进行的全面阐述的基础上，深入分析了物理游戏特殊效果的开发，并且针对抛投类手机游戏开发的主要技术与难点，比如多边形碰撞盒绘制，物理抛投，碰撞破碎，游戏场景编辑等一系列问题，提出了一套整合优化的开发方法。在此基础上设计与实现了一套快速，灵活，高效的基于 FLASH BOX2D 的抛投类手机游戏开发框架。

最后通过“攻城对决”实例游戏的开发，在时间成本、游戏运行的稳定性等方面，都取得的预期的效果，进一步验证了基于 FLASH BOX2D 物理引擎的抛投类手机游戏开发框架的实际应用价值。同时也为相关手机游戏的开发积累了重要的经验。

关键词 FLASH，BOX2D，抛投，物理碰撞，手机游戏

DESIGN AND IMPLEMENTATION OF CASTING MABILE GAMES

ABSTRACT

With the increasingly popular ADOBE AIR cross-platform runtime, application development FLASH APP ActionScript3.0 in various handheld devices has been rapid development. The BOX2D as a superior performance C + + open source physics engine, FLASH also ported to the platform. FLASH development in many categories of mobile games, tossing games with physics engine is gradually becoming a shining branch, popular on Android, iOS mobile operating system and other mainstream handheld devices. However, the direct application of FLASH BOX2D dumped class physics engine developed mobile games, dumped in physics simulation, collision broken objects, game scene editor, etc. There are more difficulty of development, but also in the time cost of development, operational efficiency and other aspects of the game have encountered no small challenge. Based on game development efficiency and stability, in game development it is very necessary to summarize, and then a set of innovative lightweight, easy to expand, easy-to-use FLASH dumped class mobile game development architecture and design.

FLASH-based mobile games category dumped the idea is to use BOX2D powerful physics engine to achieve the object of the game corresponding polygon collision box drawn. At the same function on the basis of common dumped on the application of theoretical knowledge of the physical geometry of the diversity dumped trajectory design and implementation be fun. For collision crushed strategy game based on the needs of practical application, considering alternative solutions given a reasonable basis for the performance and development of the complexity. There is also full account of the diversity of the convenience and control of the game debugging, dumped class of mobile gaming scene editor made the detailed design and implementation. The overall design of the full use of the ideological framework of component packaging concepts and design patterns, and strive to

FLASH BOX2D further package, yet easy to use in order to achieve the purpose of easy expansion.

This article by a detailed analysis of the class mobile games, mobile games are summarized be developed based on FLASH BOX2D physics engine advantage, and pointed out the problems that exist in the development process. On the basis of a comprehensive technology FLASH BOX2D elaborated on in-depth analysis of the development of a physical game special effects, and the main technical and difficult for the dumped class mobile game development, such as drawing a polygon collision box, dumped physics, collision broken, game scene editing a series of problems, a set of optimized integrated development approach. On this basis, the design and implementation of a set of fast, flexible, and efficient class-based FLASH BOX2D dumped mobile game development framework.

Finally, the "siege duel" game development instance, in the time cost, stability and other aspects of the game to run, have achieved the desired results, and further validate the practical application of class-based mobile game development framework dumped FLASH BOX2D the physics engine value. At same time, this paper also for the development of mobile games related accumulated significant experience.

Keywords FLASH； BOX2D; Casting； Physical collision； Mobile Game

目 录

1 绪 论

1.1 FLASH 手机游戏开发的背景

早期的 Flash 应用，因为得不到 GPU 硬件的加速，所有的图片渲染工作都由 CPU 来完成，这很大程度上限制复杂图形的应用。而且当时的网络下载速度也比较慢，对于资源占用量较大的游戏动画特效，也只能是弃之不用，这直接影响了 FLASH 游戏的体验性。因此当时主要是以应用矢量图为主，位图为辅的策略来开发游戏。随着 Adobe 开发的新技术 stage3D 的诞生，极大的提升了 FLASH Player 和 Adobe AIR 的位图渲染能力，使得基于桌面浏览器和 IOS、Android 等移动设备的应用程序都能够使用 GPU 的硬件加速能力。而应用 BOX2D 物理引擎开发的 FLASH 游戏，更因为其真实的操控体验，而受到了越来越多游戏体验者的肯定，《愤怒的小鸟》、《水果忍者》等都是比较成功的例子。

1.1.1 手机游戏平台发展现状

目前的手机游戏主要运行在智能手机上，智能手机与比传统手机相比，拥有更强的运算能力，特别是在显卡加速方面有了很大的提高，保障了游戏绚丽画面的流畅输出。手机所装载的操作系统主要包括 Symbian 系统、Android 系统、iOS 系统、Windows Phone 系统和 BlackBerry 系统[1]。

Symbian 作为智能手机的先驱，在 2005 年至 2010 年曾一度引领风潮。随着手机互联网的迅猛发展，由于其对社交网络支持的不足，Symbian 系统的市场份额逐日下滑，到 2010 年底，Android 操作系统的市场份额就超过了 Symbian 系统。

Android 系统作为一款开源并且免费的手机操作系统，由于其开放、包容的特点赢得了全球数以万计程序员的青睐，这在很大程序上确保了它的活跃性与创新性。目前 Android 系统已经成为了市场上主流的智能手机操作系统。

iOS 是苹果公司为其移动设备 iPhone、iPad 所设计的操作系统，随着 iPhone 以及 iPad 的成功推广，目前 iOS 系统已经成为了手持设备上最流行的操作系统之一。

Windows Phone 是微软的手机系统，它起步最早，在 2004 年就已经推出，但一直没有得到市场的广泛认可，可以说是发展缓慢，直到看到 iOS 和 Android 的迅猛发展之后，微软才重新组织力量进行研发。到了 2011 年，Windows Phone7 开始应用于 NOKIA 的智能手机上。2014 年 4 月 Build2014 开发者大会发布 Windows Phone8.1。Windows Phone 起步早，发展慢，相比 Android 和 iOS 还有很长的路要走。

Blackberry是加拿大Research In Motion公司推出的一款注重于邮件商务应用的智能手机系统，和其他智能手机系统显著的不同点在于它的安全性能更强。另外它的系统稳定性也非常的优秀，因此得到了众多商务人士的青睐。也正是因为它的独特定位，在大众市场上很难得到广泛的认可。

1.1.2 FLASH 手机游戏发展现状

近年来，在 IPhone 和 Android 等智能终端推出之后，手机游戏发展迅速。通过移动网络为用户提供单机游戏、网络游戏的业态也呈显出越来越热的趋势。据不完全统计，自 2007 年到 2012 年间，中国的手机游戏市场发展迅猛，其规模的复合增长率达到了45.4%，而且增长率还曾现出了不断扩大的趋势。在市场体量上，2013 年的规模就超过50 亿元[2]。在手机游戏活跃用户的统计数据来看，2012 年达到了 1 亿，预计在 2014 年将达到 4 亿[3] 其增长率也是相当的可观。

目前手机游戏中，利用物理引擎开发的游戏种类比较多，用户粘度比较高，而且用户评价也比较好，分析一下主要有以下几个大类：

1、切割类游戏

这类游戏的玩法主要是通过触摸屏划动拉出线条来切割不同的物体，使得切割后的形成的物体碎片并且获得物理属性，然后根据相应的物理属性来更新物体的状态，以达到游戏所设定的目标。这类游戏代表作品主要有《智慧之刃 Splitter》,《冰破维京船 Ice Breaker》等。

2、反弹类游戏

反弹类游戏主要是利用物体相互碰撞后的自然折返为基础，通过加入各种趣味性的游戏关卡设计而成的一类手机游戏，比较简单的有台球物理。其他的还有《点火 Pyroll》,《炮打笑面人 Roly-Poly Cannon》,《魔法水滴 Enigmo》等。

3、平衡类游戏

平衡类游戏主要是通过判断不同材质的物体所具有的不同密度，根据体积的不同，形成不同的质量，然后通过物体的堆积达到游戏所设定的平衡状态。这类的游戏代表主要有《完美平衡 Perfect Balance》,《巨塔 Huje Tower》,《图腾破坏神 Totom Destroyer》等。

4、绘制类游戏

绘制类游戏通常会让游戏体验者控制画笔，根据游戏体验者选择的材质，以及绘制的不同形状，形成不同的具有物理属性的物体，去完成游戏所设定的不同的目标。比如，画一个圆，就形成一个球，球会在重力作用下下落，形成一定冲力。这类游戏主要有《神奇画笔 Magic Pen》,《魔法画笔 Draw Play》等。

5、道具摆放类游戏

道具摆放类游戏类似于绘制类游戏，所不同的是游戏会预先给予游戏体验者各种不

同的道具，而不是由游戏体验者去绘制来形成物体。游戏体验者可以将不同的道具摆放在各种可能的位置上，点击完成后赋予物理属性，来完成游戏所设定的目标。这类游戏主要有《多米诺传动系统 Dynamic Systems》,《疯狂机器》,《粒子对撞 Collider》等。

6、抛投类游戏

通过炮塔、弹弓一类的抛投设备，将炮弹等抛投物体按照相应的角度与力量投射出去，给目标物体以撞击等损害，这是其基本原理。在此基础上通常可以发展出具有不同风格，不同轨迹的抛投对象。这类游戏的典型代表就是《愤怒的小鸟》。

1.1.3 FLASH 游戏技术现状

Flash 一开始是以使用矢量技术制作动画来达到发布文件体积较小，网络传播快而著称的[4]。基于此，Flash 游戏可以让用户省去客户端游戏下载的漫长过程，而直接在网页上进行游戏的操作。较大型的 Flash 网页游戏通常都允许用户在下载了基本的模块之后，就可以进入游戏了，而其它暂时不相关的模块则是在后台预下载或者直到用户进入相应模块再下载。这也就是在玩 Flash 网页游戏时，经常可以看到打开某页面的时候，会先 loading，然后画面再显示出来，这样实现了边下载内容边使用相应模块的功能，这是 Flash 最大的技术特点所在。但也是因为这种即时下载，Flash 网页游戏的玩法很多都是回合制，很少即时类。这一技术特点决定了 Flash 网页游戏的前端表现、后端数据是在不断的传输中的，如果是即时战斗玩法则会导致延迟，但在回合制游戏里这一弱点便会削弱，这是回合制的优势，随着网速的提升，延迟问题也会一定程序上得以改善。

3D 是 Flash 将来的发展方向。虽然做 3D 游戏的成本会比较高，但对于游戏公司来说[5]，3D 产品不容易被同质化。3D 游戏的下载量大，下载时间长可能会是个问题，但将来随着国内网络速度的进一步提升都会得到解决。

Starling 是一个基于 Stage3D（这是 Flash Player11 及 Adobe AIR 3 中新增的为 3D 加速功能所提供的 API）所开发的一个能够使用 GPU 来加速的 2D Flash 应用程序的 ActionScript3 框架[6]。Starling 主要是为游戏开发而设计的，但是它的用途不仅限于此。Starling 最大的好处在于你可以很快地写出使用 GPU 加速的应用程序而不必接触那些复杂的底层 Stage3D API[6]。

图 1-1 是 Starling 与 Stage3D 结构关系图，大多数 Flash 开发人员只要使用 Starling 框架就可以轻松的获得 GPU 的硬件加速能力，从而极大的提高 Flash 手机游戏的性能与用户体验。

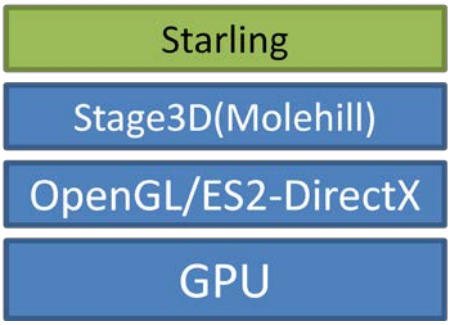

图 1-1 Starling 与 Stage3D 结构关系图

Fig.1-1 Starling and Stage3D structure

1.2 研究的目的和意义

截止到 2013 年 2 月，作为真正跨平台、跨浏览器并在各种浏览器版本中提供一致网络体验的 Web 互动媒体标准，Flash Player 在全球电脑中的装机量为 13 亿台，是 PS、Wii 和 XBOX 系列游戏主机销量总和的 5 倍，Flash Player 新版本发布一周内浏览器更新量为 4 亿+。庞大的用户群体是商业开发 Flash 游戏的厚实基础。

Flash 游戏应用通过 Adobe AIR 打包，可以轻松的发布成 iOS 和 Android 等移动平台所使用的原生 App。截止到 2013，可开启自身硬件加速的移动设备数为 5 亿台，已经发布的 Flash 移动游戏 App 超过了 2.8 万个，Facebook 上 96%的游戏是用 Flash 开发。在中国，每月四千万人玩的“开心农场 2”采用的也是 Flash 新技术 Stage3D，排名前九位的 Flash 网页游戏带来了超过 7 千万美元的月收入。

Flash 手机游戏开发的便捷性，在一定程序上也降低了开发商进入手机游戏行业的门槛。开发商们普遍认为 Adobe Air 是把他们的游戏转向移动市场最好的工具。

因此，选择 FLASH 手机游戏开发作为本课题的内容进行深入研究，具有以下意义：

1.对从 Flash 网页游戏开发转向 Flash 手机游戏开发的人员来说，全面了解 Flash 手机游戏开发的现状，并在此基础上提高对 Flash Box2D 开发的认识。

2.通过对 Flash 手机游戏的各个功能模块进行细致的分析，为实际开发提高开发的效率，保证开发的质量。

3.通过实例的分析与实现，巩固对 Flash Box2D 手机游戏开发的认识，并找到不足之处，为将来的进一步提升打下坚实的基础。

1.3 论文的主要研究内容及结构安排

在开发带物理引擎的 Flash 游戏中，Flash 常用的引擎主要是 Box2D。应用 Flash Box2D 构建物理世界的过程是比较繁琐，相当枯燥的，需要程序员与设计师的长时间沟通与协作，在不断磨合中，才有可能达到精确、合理的实现游戏物理世界的目标。如何简化物理游戏开发，优化碰撞盒数据的自动采集，实现物理与可见 UI 之间的精确绑定，

是物理游戏快速成功构建的关键。

本文将通过对 Flash Box2D 核心概念的阐述，进而分析了物理基本形状的构建，并且逐步延伸到对不规则物体的模拟，最终实现复杂物体物理碰撞的搭建。在此基础上，分析设计、实现物理碰撞盒的图形化绘制。最后应用面向对象的设计方法与相关设计模式，对 Flash Box2D 物理引擎进行二次封装，实现物理碰撞盒与 UI 的快速绑定，最终整合成一套快速开发 Box2D 物理游戏的框架。

论文的重点是通过对 Flash Box2D 的技术模块应用的分析与实现，实现在 FLASH 手机游戏开发中的实例应用，因此论文将以此为主线，分析 FALSH 手机游戏的发展现状，开发技术，以及 FLASH BOX2D 的各种场景的应用，最终形成比较完整的 FLASH 手机游戏开发框架。论方共分 7 个章节，各章节安排如下：

第一章绪论，介绍了 FLASH 手机游戏发展的现状，以及 FLASH BOX2D 手机游戏的发展，并论述的项目的目的与意义，以及论文主要的内容。

第二章简要叙述了 FLASH 手机游戏的开发现状，分析了 FLASH 抛投类手机游戏的特点以及 Flash Box2D 物理引擎的应用优势，随后对 Flash Box2D 在手机游戏开发中存在的问题进行了论述，以使读者对 FLASH BOX2D 有了基本的认识。

第三章介绍了 FLASH BOX2D 的基础技术，对基本的 FLASH 物理游戏进行了深入浅出的分析，并在此基础上对 Box2D 特殊效果的实现进行了详细介绍，以使读者对 FLASH BOX2D 有了全面深入的认识。

第四章主要是基于 FLASH BOX2D 物理引擎，对物理引擎进行进一步封装，设计 EASY BOX2D 框架结构，并根据抛投类游戏主要功能进行分类设计，最后对游戏编辑器的结构进行了设计，为之后的实现打下坚实的基础。

第五章在之前 EASY BOX2D 框架结构的基础上，实现了 EasyBox2D 的游戏框架以及抛投类游戏的主要功能，最后对游戏编辑器进行了逻辑实现。

第六章是基于 EASY BOX2D 框架，进行抛投类手机游戏实例开发的介绍，论证了游戏框架设计的现实意义。

最后一章是对论文的总结各展望，对论文特色进行阐述，同时分析设计方法存在的不足，并对各模块的应用在将来的工作中的进一步探索进行了展望。

2 FLASH 抛投类手机游戏的需求分析

2.1 FLASH 开发手机游戏概述

随着 Adobe 公司于 2007 年发布 AIR 之后，Flash 通过 AIR 运行时实现在其在桌面系统上的部署，摆脱了 FLASH 只能运行于浏览器的单一局面。通过 AIR 开发的 FLASH 手机游戏应用，不仅具有原来 FLASH 的几乎所有功能，而且还可以应用原生的操作系统所提供的功能，像本地文件的读写、系统菜单的调用以及系统特定事件监听与派发[7]。

2.1.1 Android 与 iOS 手机应用概述

对于 ActionScript3.0 的开发者来讲，开发基于 Android 和 iOS 系统的游戏应用和开发桌面 AIR 应用其实没有多大的区别，所不同之处仅在于发布过程中，Android 手机应用程序是发布成.apk 文件[8]，而 iOS 则发布成.ipa 文件。在开发手机应用时，既可以访问 Flash Player 的 API，还可以调用一些增强功能，像基于矢量的绘图，摄像头等多媒体支持等等[9]。

总来说 AIR 对于 Android 和 iOS 应用的支持，主要集中在多点触摸功能、手势识别功能、加速度计功能、持久性文件读写功能和数据库存储等方面[10]。对于不些不被支持的功能与 API 下面将详细说明。

2.1.2 FLASH 对设备的支持与 API 的限制

FLASH 对于 Android 和 iOS API 的支持总体来讲是比较类似的，对于摄像头和麦克风等输入设备的支持有一些区别，表 2-1 列出了具体情况 [9]。

表 2-1 FLASH 对硬件设备的支持表

Table2-1 FLASH support hardware table

功能模块	Android	IOS
触摸操作与手势识别	√	√
地址位置传感器	√	√
加速度计	√	√
麦克风	√	
摄像头	√	
Stage Web View	√	

续表 2－1

功能模块	Android	IOS
摄像头媒体库/照片库	√	仅支持添加到库
SQLite 数据库	√	√
文件读写操作	√	√
通过 URL 协议加载移动设备的能力，例如电话、电子邮件和短信	√	√

除了上面表格描述的功能之外，还有一些不支持的核心移动功能，比如对蓝牙的控制，联系人列表的读写操作，原生手机 UI 的控制，音乐播放器以及音乐库的一系列功能的控制，以及参数设置，日历设置等等。

总的来说，AIR for Android 和 Packager for iPhone 在与相应的设备功能配合运作方面，都要弱于原生的 SDK 开发的应用，因此在创建 FLASH 手机应用程序时，充分考虑到这些局限是很有必要的。

另外在 Android 和 iOS 上开发 FLASH 游戏应用时，还要考虑到某些 FLASH 的 API 以及 AIR 的功能是不被支持的。表 2-2 列举了一些不受支持的 AS3 API 对象 [9]：

表 2-2 不受支持的 FLASH API 对象列表

Table2-2 FLASH API unsupported objects list

Accessibility
DRMManager
DNSResolver
DockIcon
EncryptedLocalStore
HTMLLoader
LocalConnection
NativeApplication 主要有 exit(), isSetAsDefaultApplication(), menu,和 startAtLogin 方法
NetworkInfo
NativeProcess
NativeWindow 和 NativeWindow.notifyUser()
NativeMenu
PDF 支持主要是读取
PrintJob 主要是应用设备打印的能力
Socket 支持(DatagramSocket, SecureSocket, ServerSocket)
Socket.bind()
Shader 和 ShaderFilter 应用于 UI 表现层的修饰
StorageVolumeInfo
XMLSignatureValidator

作为 Flash Builder 开发的另外一大分支，Flex MXML 并不支持创建 AIR for Android

应用程序，与 iOS 也是不兼容的。

2.2 FLASH 抛投类手机游戏的特点

2.2.1 FLASH 抛投概述

游戏中任何将物体扔，抛，撞，弹到空中的运动，都可以理解为抛投运动[11]。在 Flash 游戏中通常是首先选择一个对象，并对其进行拖拽，沿某个方向移动一段距离，然后松开对象，对象沿着拖拽的方向继续移动。

Flash 手机游戏中的抛投，可以完全遵循物理规律来模拟。应用几何抛物线公式，计算出抛物轨迹，然后加入重力，风，摩擦力等物理因素的影响，达到真实的物理世界体验[12]。也可以是在遵循一定物理规律的基础上，进行适当违反客观规则的创作，以达到抛投曲线的多样性，趣味性。这也是很多市场认可度高的游戏中经常采用的设计方法。

2.2.2 FLASH 物理碰撞分析

Flash 关于物体碰撞的检测方法有很多，而这也是抛投类游戏必不可少的，必须要思考解决的问题。通常来讲，Flash 原生的碰撞检测 API 函数有 hitTestObjet()和 hitTestPoint()[17]，但是有很多抛投类游戏都不止于使用这些原生的方法。为了达到完美的游戏效果，通常还有几种更精确的方法，下面进行具体分析。

物体经过抛投之后，就要进入碰撞检测阶段了，这也是 Flash 抛投类游戏必不可少的阶段。

碰撞检测的思想非常简单，只要知道两个物体是否有在同一时间内某个部分处在了同一位置上。当然，也许物体不止两个，这就需要知道其中一个是否和共它的物体发生了碰撞。

Flash 碰撞检测的方法有很多，可以是判断两个显示对象是否重叠，也可以对物体实际像素进行检测，还可以根据距离来判断，当然如果使用 Flash Box2D 物理引擎的话，则可以使用 Box2D 提供的碰撞检测。各种碰撞检测各有特点，具体分析如下：

（1）hitTestObjet()主要用于判断两个显示对象间是否发生碰撞，这个最简单，也是最高效的碰撞检测方法。但是这种方法不够精确，因为它是判断两个显示对象的矩形边界是否相交。图 2-1 为 Flash 碰撞检测示意图，图中只有正方形是发了了碰撞，但由于实际上 Flash 检测的是矩形边界，因此这三组对象都被检测发生了碰撞[14]。

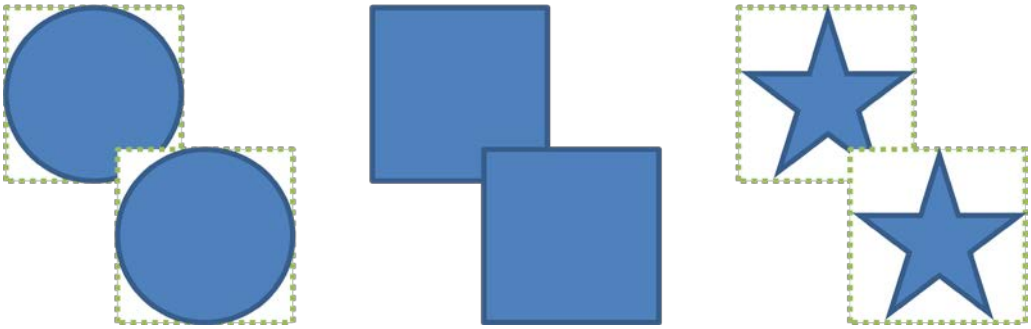

图 2-1 Flash hitTestObjet 碰撞检测

Fig.2-1 Flash hitTestObjet hit test

因此可以认为，hitTestObjet()只能进行初略的碰撞检测，或者是矩形的碰撞检测。

（2）hitTestPoint()方法用于判断某个点与显示对象间是否发生了碰撞。默认情况下，这种方法也会遇到点与显示对象矩形边界发生碰撞的问题，但是它有一个参数 shapeFlag 可以控制在碰撞中是检测显示对象的可见图形还是矩形边界。这种方法比较适用于非常小的显示对象与其他显示对象之间的碰撞。

（3）基于距离的碰撞检测，它的主要思想是先确定两个物体分开的最小距离，再检测当前它们之间的距离，然后比较这两个距离的大小。如果当前距离小于最小距离，那么就断定两个物体发生了碰撞。从上面的描述不难发现，这种碰撞检测比较适合于圆形物体之间。

（4）不规则形状对象的碰撞检测主要是基于像素的检测，通过比较两个 BitmapData 对象，检测它们是否有像素的重叠来判断是不发生了碰撞。

这种方法虽然可以适用于任何形状，比较精确，但是对于每个显示对象，尤其是矢量对象，都要准备一份 BitmapData 来用于碰撞检测，这对内存的消耗以及 CPU 的消耗比较大，对于 Flash 抛投类手机游戏的开发，必须时间注意其对性能的影响。

（5）Flash Box2D 物理引擎来碰撞检测。

Flash Box2D 是采用包围盒的方法进行碰撞检测的，它的主要思想是建立不同精度的几何模型来代替碰撞物体，首先在较低精度的包围盒之间进行碰撞检测，以较快速度来排除不可能相交的几何元素对,从而有效的减少相交测试的次数，然后进一步测试较高精度模型来达到一定的碰撞检测真实性。

2.2.3 FLASH 物体破碎策略分析

前面章节分析了很多碰撞的方案，碰撞之后将要考虑则是物体的破碎了，物体爆炸，改变颜色或者只是简单的消失，这些多是通常 Flash 游戏可以考虑的方式，但是对于抛投类游戏而言，应用最多的还是对象的破碎。

如果从快速开发的角度来看，那么预先渲染帧动画来模拟物体的破碎不失为一个好的策略。但是如果要获得真实的破碎效果，大部分游戏还是通过大量物理模拟计算，来重新生成新的破碎对象。从这个角度考虑，选择 Flash Box2D 物理引擎来帮助计算破碎

对象，进而模拟物体破碎，也不失为一个好的选择。

2.2.4 FLASH 游戏关卡设计

游戏关卡主要是通过设计不同难度的游戏场景，让游戏操作者在某些条件未满足时，不能任意前往下一关[13]。而不同关卡具有不同的游戏道具，这些道具具有不同的作用与生命值，如何有机搭配组合出符合游戏总体进程的关卡，往往是决定游戏能否成功的关键。

游戏关卡设计通常会经过策划，场景搭建，测试关卡几个部分，如果完全手工的进行 Flash 抛投类游戏的场景搭建，关卡测试，则既费时费力，又容易出错。而除了手工进行关卡设计之外，通常还可以借助 Flash CS 设计工具，先将游戏道具整理到资源库中，然后根据需要拖放不同的道具到舞台上，最后使用 JSFL 对整个场景进行遍历，记录下各种道具的类型，位置信息，最近通过 XML 等文本形式保存关卡的信息。

关卡测试则需要另外打开游戏，读取场景搭建数据后进行测试，整个流程虽然比完全手工运作来的简便，但还是需要在不同工具间进行切换，在关卡搭建的过程中不能实时预览真实的道具搭建效果。

2.3 Flash Box2D 物理引擎的应用优势

2.3.1 Flash 物理引擎现状

Flash 物理引擎是一组计算机应用程序，用来模拟牛顿物理模型，通过对物体赋予质量，速度，作用力，摩擦力，弹性等物理变量，来模拟计算在不同情况下的物体应力效果。它主要应用于模拟物理状态的互动游戏中。

目前的 FLASH 物理引擎主要有 FLASH BOX2D, APE,FLADE,FOAM,FISIX 等，用的比较多的主要是 FLASH BOX2D 和 APE。

Box2D 最初是一个用于模拟 2D 刚体的 C++引擎，因为其出色的性能和丰富的功能，后来被开发者移植到了 FLASH 平台，便有了 Flash Box2D 物理引擎。

Flash Box2D 是模拟刚体的物理引擎，物理世界的刚体具有不同的形状，比如圆形，矩形，三角形，凸多边形，复杂多边形等等。这些刚体在物理世界中创建后受到重力，摩擦力等物理作用力的影响。同时在运行过程中，还可以根据游戏设计的要求，施加各种作用力在刚体上，从而实现各种动作需求。

APE(Actionscript Physics Engine)是一个 ActionScript3 写成的物理引擎，用于模拟现实中物体发生的运动和碰撞。它是免费、开源的，遵循 MIT 协议。但其成熟度不够好，相对于 Flash Box2D 来讲功能更为单一。

FLASH BOX2D 与 APE 的功能比较可以参见表 2-3。

表 2-3 FLASH BOX2D 与 APE 功能对比表

Table2-3 FLASH BOX2D and APE function table

	Flash Box2D	APE
多场景	支持	不支持
模型	模型多，功能强大	有基本模型
渲染	无渲染模型	有渲染模型
碰撞精确度	比较精确	有时存在视觉 bug
单位	MKS(Meter-Kilogram-Second()	Pixel

2.3.2 Flash Box2D 应用优势

利用物理引擎制作的游戏，因为手感真实，通关方法多变而越来越受游戏体验者欢迎。而 box2d 引擎由于上手简单、性能良好、效果逼真等优点，成为越来越多物理游戏制作者的首选。在国内第一个推出客户端游戏开发平台和 SDK 的网易 iTownSDK 游戏开发者计划，在上周更新的 SDK 版本中，也趁热打铁地整合了 box2d 引擎接口，开放 2D 物理引擎。2D 物理引擎的开放使得开发者使用 iTownSDK 所能制作的游戏有了更多的选择。物理游戏因为其来自于人们对现 实的认识，操作简单，玩法容易理解，一直受到游戏体验者的喜爱。而物理游戏的题材、玩法设计还有很多可挖掘之处。

Flash Box2D 具体应用优势有以下几点：

（1）对于需要进行各种物体碰撞检测的，建议不要自己去写一堆方法，而用 Box2D 去进行模拟，这样便于后续的应用扩充。应用扩充起来的方便程度远远大于你自己写函数进行判断。

（2）对物理世界中的物体进行添加和删除，一定要在模拟结束后，可能有人问什么时候算模拟结束，这个点也就是在 step 执行后。如果不是这么删除 body，那么很可能引起程序崩溃，你可以想象，一个物体正在运动，某位大仙突然让它穿越了，物理世界很茫然，当数到这个物体时，应该进行它的动作，但发 现他不在了，于是想疯了，然后崩溃了。碰撞检测中需要物体消失也一样。

（3）物理世界的 step 模拟时间为 0 时，意味着世界禁止，也就是停止了，所以暂停和开始的功能很容易。

Flash Box2D 已被用于蜡笔物理学、愤怒的小鸟、Rolando、Fantastic Contraption、Incredibots、Tiny Wings、Transformice、Happy Wheels 等游戏的开发。“愤怒的小鸟”需要控制好小鸟弹出的力度以及方向来击中目标，而“Tiny Wings”则需要掌握好滑坡物理加速度以及风的力量以让小鸟尽可能飞得高飞的远。而 box2d 是一个功能丰富的 2D 物理引擎。使用 Box2D 物理引擎，开发人员可以方便的赋予运动物体相应的物理属性，像碰撞、自由落体、惯性等等。在开发游戏过程中，利用此物理引擎开发会比较简单，

学会了创建物理世界后，加上相应的物体就可以实现物理模仿。

2.4 Flash Box2D 在手机游戏开发中存在的问题

2.4.1 Flash Box2D 游戏开发的效率

应用 Flash Box2D 来进行手机游戏开发，尤其是抛投类手机游戏的开发是个不错的选择，但是开发的效率还是存在不少问题。

总体上来讲，对 Flash 开发人员来说，主要是前端开发，习惯了对显示对象的控制，然而 Flash Box2D 主要是对物理世界数据的模拟，并没有直接的显示对象，虽然可以通过调试模式看到模拟对象的线框表示，但对于注重数据模拟的编程习惯来说，还是有一定的学习成本曲线。

对于物理形状的构建，Flash Box2D 只支持矩形，圆形，及凸多边形的构建。然而在游戏中往往还会有很多特殊形状物体的创建，比如任意多形边，这些物理刚体的创建如果都用硬编码的形式为做的话，效率非常低。举个例子，如果一个游戏场景有 20 个凸多边形的话，如果一个多边形从顶点的计算，位置的摆放，角度的旋转一系列功能的调整来计算，差不多会要 10 分钟的工作量，那么一个游戏场景就要花 3 小时 20 分钟，这还只是按 Flash Box2D 支持的凸多边形来计算，如果要涉及到凹多边形的话，工作量会更高，单单一个场景数据的调配有时差不多要到一天的工作量。在 Flash Box2D 抛投类手机游戏开发过程中，一个游戏几十个，上百个游戏场景的情况是很普遍的，如果直接用 Flash Box2D 来开发，效率是比较低的。

另外，为了游戏的效果，通常还会有一些特殊效果的考虑，比如浮力效果，流体效果等等，这些都会比较花时间。

2.4.2 Flash Box2D 游戏开发的难度

Flash Box2D 游戏开发除了效率方面的问题，还存在着不少难点，以下是在实际开发中经常碰到的一些难点的归纳：

（1）抛投

开发 Flash 抛投游戏，第一个想到的应该是如何进行抛投。抛投按视角来分可以分为第一视角和第三视角的抛投。

所谓第一视角的抛投，就是游戏操控者从屏幕的底部向顶部抛投，也可以说是向前抛投。如果应用 3D 游戏引擎或者算法，做成 3D 效果的第一视角抛投游戏是最具有代入感的，但通常 Flash 开发的第一视角抛投游戏还是以 2D 效果来开发的居多，其算法主要是利用物体的缩放来达到前后抛投的效果。在抛投的过程中游戏操控者通常看不到游戏的主角。

根据对 Flash 抛投类游戏的统计分析，受市场欢迎的还主要是以第三视角开发的。第三视角的抛投算法主要依据是抛物线，游戏操控者一般从屏幕的左边向右边抛投物体。在抛投的过程中游戏操控者可以看到游戏的主角。

下面主要探讨一个第三视角抛投的难点。如果只是抛投圆形的物体，比如炮弹，球，那么只要计算出抛物线的路径基本上就可以解决问题了，不论是自己开发抛物线公式，还是应用向量分量的计算方式，也或者直接使用 Flash Box2D 物理引擎。但是如果游戏要抛投的是矩形的物体，比如弓箭，那么问题就复杂的多了。

如果只计算抛物线，使用 Flash Box2D 来实现抛投的一般步骤是这样的。

1、获取游戏操控者操作弓箭的发射力度以及角度。

2、把角度值赋值给弓箭的 angle 属性。

3、把力度值赋值给弓箭的 linearVelocity 属性。

4、设置线性推力，以使弓箭在重力作用下按抛物线飞出。

效果就会像图 2-2 所示的样子。

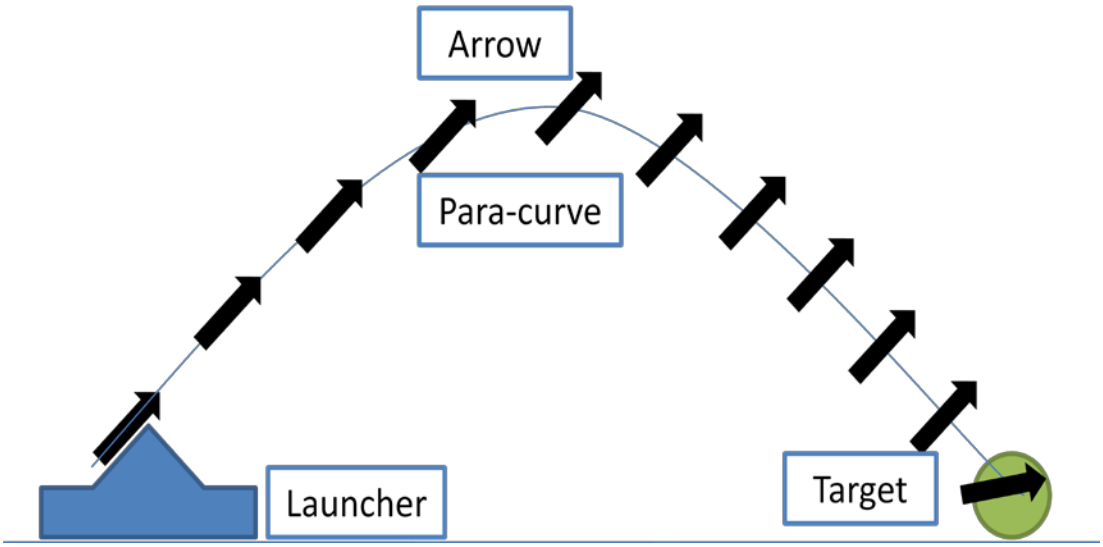

图 2-2 弓箭飞行轨道

Fig.2-2 Arrow flight path

这显然不是游戏要的效果。也不是正确的物理抛投现象，真实的情景应该是让发射出去的弓箭，能随着“弹道”发生相应的角度变化，就像图 2-3 一样。

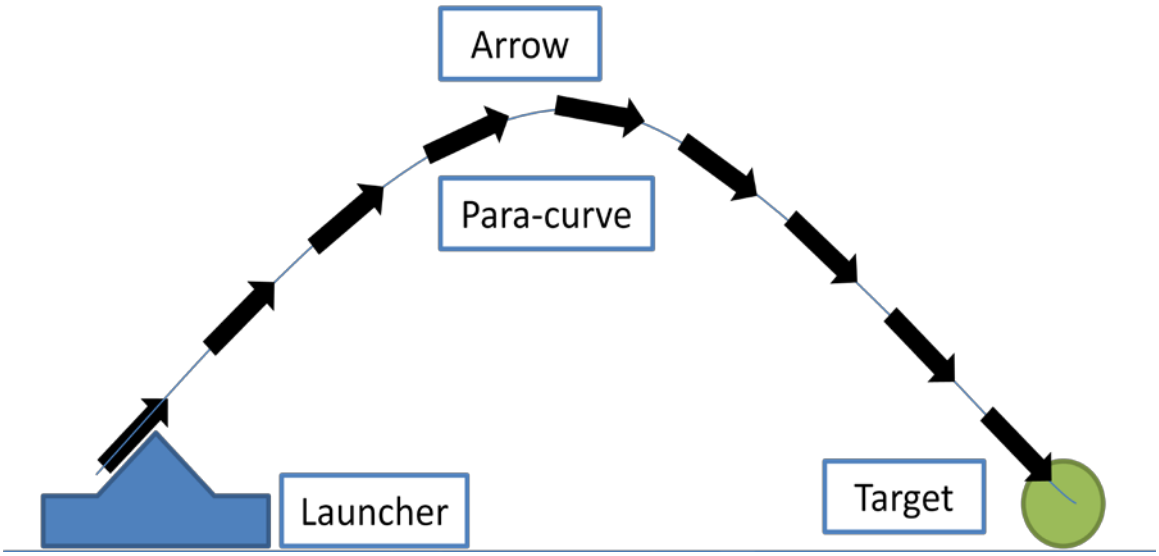

图 2-3 弓箭飞行实际轨道

Fig.2-3 Arrow flight real path

（2）破碎

在 Flash 抛投游戏中，物体由于碰撞而产生破碎是另一个要探讨的难点。位图序列破碎动画是比较常用的破碎策略。因为位图序列动画一般是由动画效果工程师在先期就渲染制作好的，程序员只要将渲染好的。位图动画加载到游戏中，在适当的时候控制它的播放就好了，无论从效果还是实现难度都是比较好的，但是因为其是预先渲染好的位图序列，这限制了它的可变化性，所有的碰撞所产生的破碎，不论力度，角度，破碎的效果都一样，而且破碎物不论大小，通常都不再具备物理性。

如何解决物体碰撞所产生的破碎的多样性与物理性就成了是 Flash 抛投类游戏的难点。破碎的多样性，可以考虑预先渲染多套位图序列动画，然后根据具体的碰撞力度与角度来分类播放不同的位图序列。这在一定程序上能解决这个问题，但是这样一样则势必会增加内存的使用，对手 Flash 抛投类手机游戏，内存的合理使用也是一个必须的考量。

破碎的物理性，显然，在碰撞之前，物体是有质量，弹性，形状等一系列物理属性的。在碰撞之后，如果应用位图序列动画来表现物体的破碎的话，为了在视觉上完美，就必须让位图序列动画代替原来的物体，也就是说必须把原来的物体从世界中删除，而只显示破碎动画。破碎动画只是位图动画，其本实不属于物理世界，所以也就不可能与生俱来物理属性。

如果要让每个破碎的独立物体具有物理属性，则必须重新构建所有的破碎物体的刚体，基本的步骤如下：

1、根据物体的破碎程度，筛选出比较大的破碎物，也就是说剔除掉相对较小的颗粒物。

2、扫描破碎物的位图形状，如果是类似凸多边形的，计算出各个破碎物的多边形

顶点。

3、通过 Flash Box2D 的一般流程创建出各个破碎物刚体。

4、如果是类似凹多边形的形象，则先切割成多个凸多边形，然后再分别计算出各个凸多边形顶点。

5、通过 Flash Box2D 的一般流程分别创建出各个凸多边形。

6、最后通过组合法将各个凸多边形组合成原凹多边形，以形成破碎物的物理属性的赋予。

这些步骤每一步多困难重重，对位图图形的分析涉及到了渲染显示的底层算法，其复杂程序可想而知。

2.4.3 FLASH BOX2D 手机游戏的优化

在 FLASH 开发 WEB 应用和桌面应用时，通常不需要太多关注 CPU 和 Memory 的使用限制，因为现在的电脑通常性能都是比较优越的，而对于移动设备来说则要注意这些资源的使用，虽然最新的 Android/iPhone 的处理器性能和内存容量有了很大的提高，但跟电脑应用相比，还是小了很多，因此，在创建 FLASH 手机应用时要始终关注性能，并对发布的版本进行必要的优化

如何使 FLASH 手机应用所使用的 CPU 和内存最小，是游戏开发的优化方向，但是这两者之间时常是相互影响的，当采取措施减少内存的使用时，往往会使用更多的矢量图，以致 CPU 资源的使用上升了，反之，如果为了减少 CPU 的使用，而用位图来代替矢量效量的绘制时，就会增加内存的使用。

因此，在优化的同时要考虑一个问题，当前更重要的是内存还是 CPU 资源，并尽量能使他们的使用达到一个平衡。

2.5 本章小结

本章主要对 FLASH 手机游戏展开了概述。分析了 FLASH 抛投类手机游戏的特点，包括抛投，物理碰撞与物体破碎策略的分析；概述了 FLASH 物理引擎的现状，并分析了 FLASH BOX2D 物理引擎的优势；在此基础上深入分析了目前 FLASH 手机游戏开发的效率、难度与优化问题。本章是本次课题研究的现实需求，为全文的展开提供现实意义的支撑。

3 FLASH 抛投类手机游戏的技术基础与关键

3.1 Flash Box2D 技术基础

3.1.1 Flash Box2D 核心概念

（1）世界(world)：物理世界就是模拟物理刚体的一个集合，物理刚体由世界创建，在这个集合中的物理刚体受到共同的物理特性的作用，比如重力。理论上来讲，在游戏中可以创建多个世界，以便模拟多个物理作用集合，但通常是不需要的。

（2）刚体(rigid body)：刚体可以理解为一个十分坚硬的物体，其中的任何两点之间的相对距离都是完全不变的，刚体的对立面就是柔体，像水。柔体的任意两点间的相对距离会随着流动而改变。Flash Box2D 世界中任何对象都是刚体，也就是说 Flash Box2D 只支持对刚体的模拟。

（3）形状(shape)：形状是各种二维几何体，例如圆形和多边形。在几何学中，多边形通常可以分为简单多边形和复杂多边形。简单多边形是边不相交的多边。简单多边形按凸性区分，可以分为凸多边形和凹多边形，凸多边形是指它的内角都不大于 180°。Flash Box2D 只支持凸多边形。

（4）夹具(fixture)：其概念源于机械工程，在机械制造过程中用来固定加工对象，使之占有正确的位置，以接受施工或检测的装置。Flash Box2D 中的夹具是用来将形状绑定到物体上，并添加密度(density)、摩擦(friction)和弹力(restitution)等物理属性的对象。

（5）关节（joint）：Flash Box2D 允许通过关节（joint）来创建刚体之间的约束。常用的关节有鼠标关节，距离关切，旋转关节等。通过鼠标关节，在游戏中可以实现对物品的拾取、拖拽以及抛掷；通过距离关节可以设置两个刚体之间的固定距离；使用旋转关节可以使刚体围绕中心点旋转。

以上是 Flash Box2D 的一些核心概念，下面将通过具体的代码，来详细说明 Flash Box2D 在游戏中是如何起作用的。

3.1.2 在 Box2DFlash 中定义世界

基于 Flash Box2D 的游戏开发，第一步通常就是对物理世界的创建。与现实世界一样，Flash Box2D 的世界（World）也是有重力（gravity）的，所以先定义物理世界重力（world gravity）[16]。

```
var gravity:b2Vec2=new b2Vec2(0,9.8);
```

b2Vec2 是一个 2D 的向量数据类型，它将储存 x 和 y 矢量分量。b2Vec2 的构造函数

有两个参数，都是数值，代表了 x 和 y 分量。通过这种方法定义 gravity 变量作为一个矢量。 x=0 说明水平方法上没有重力。如果要模拟一个恒定的风力作用，可以在这里定义一个数值来模拟。y=9.8 说明这个 Flash Box2D 模拟了一个近似于地球重力的重力。在物理世界中创建的所有刚体都会受到这个重力的影响，这意味着如果一个刚体被创建在空中，则它将在重力的作用下自由下落。

对物理世界中的所有刚体，Box2D 物理引擎在 Flash 播放的每一帧多需要去计算其物理的位置，这在刚体较多的情况下是比较耗费内存的。而 Flash Box2D 提供了一个对外接口，当在物理世界中的刚体静止时，Box2D 可以允许他们进入睡眠状态，这样它们将不受作用力的影响。

一个睡眠的刚体就不需要进行物理模拟计算，睡眠的刚体只是表示自己的存在，并静止在它的位置上，不会对世界中的任何事物产生影响，Flash Box2D 物理引擎将会忽略它，从而极大的提升处理速度。为了让 Flash Box2D 知道是否可以让刚体睡眠，通常可以定义一个变量来指示。

```
var canSleep:Boolean=true;
```

最后，要定义的世界通过下面这句话来创建：

```
var world:b2World = new b2World(gravity, canSleep);
```

这样物理世界就创建好了。为了便于在游戏中的任何模块都能访问到这个世界，通常会将其保存在一个全局的变量中，这点在后面的章节中将会细述。

3.2 FLASH BOX2D 刚体创建基本步骤概述

Flash Box2D 的世界已经创建了，接下来将深入的阐述世界中刚体的创建。

通常使用以下步骤将刚体添加到世界中[15]：

1、创建一个刚体定义，它将持有刚体的信息，例如刚体的位置信息等。

2、创建一个形状，它将决定刚体的显示形状。

3、创建一个夹具，将形状附加到刚体定义上。

4、创建刚体在世界中的实体，使用世界来具体创建刚体。

3.2.1 创建刚体的定义

无论是创建圆形还是多边形，第一步都是创建一个刚体的定义：

```
var bodyDef:b2BodyDef=new b2BodyDef();
```

b2BodyDef 类是一个刚体的定义类，它将持有创建刚体所需要的所有数据。比如刚体的位置、角度、是否活动状态等。

下面是具体定义刚体位置的语句

```
bodyDef.position.Set(x,y);
```

通过 position 属性显示的设置了刚体在世界中的位置。有一个需要注意的地方是，这里的单位是米。虽然 Flash 是以像素（pixels）为度量单位，但是在 Box2D 中尝试模

拟真实的世界并采用米（meters）作为度量单位。对于米（meters）和像素（pixels）之间的转换没有通用的标准，但是可以采用下面的转换标准，这是根据经验得出的，通常可以有很好的运行效果：

1meter= 30Pixels

在实际的编程过程中，可以定义一个变量来帮助将米（meters）转换成像素（pixels），比如 var pixelToMeter:int = 30; ，这样便可以在 Box2D 世界（world）中进行操作时使用像素（pixels）而不用使用米（meters）来作为度量单位，只需要在数值后面除上这个比例值。这将制作人员在制作 Flash 游戏时，可以更加直观使用像素来思考。

3.2.2 创建刚体的形状

形状（shape）是一个 2D 几何对象，例如一个圆形或者多边形，在这里先只讨论凸多边形（每一个内角小于 180 度）[15]。因为 Flash Box2D 原生的 API 只能处理凸多边形。

接下来以创建一个圆形为例，来详细说明刚体形状的创建：

```
var circleShape:b2CircleShape;
circleShape=new b2CircleShape(25/ pixelToMeter);
```

b2CircleShape 是用来创建圆形形状的 API，它的构造函数只需要一个半径（radius）作为参数。上面的代码中，创建了一个圆形，它的半径为 25 像素（pixels）。这里可以使用像素来思考是因为在像素数值后面除以了用刚刚创建的米/像素比例值。

也可以设置一个全局常量 worldScale 来保存这个比例，这样便可以在其他任何需要的地方方便的访问这个比例值了。以后每次想要使用像素进行操作时，只要将它们除以 worldScale 即可。当然也可以定义一个方法名为 pixelsToMeters 的方法，在每次你需要将像素（pixels）转换成米（meters）时调用。

当有了刚体定义和形状时，就可以使用夹具（fixture）来将它们粘合起来。接下来具体看看刚体夹具的创建。

3.2.3 创建刚体的夹具

夹具（fixture）用于将形状绑定到刚体上，同时也可以定义刚体的材质，比如设置密度（density），摩擦系数（friction）以及恢复系数（restitution）等等一系列物理属性[15]。

首先，创建夹具（fixture）：

```
var fixtureDef:b2FixtureDef = new b2FixtureDef();
```

b2FixtureDef 是创建夹具的定义类，很简单，可以不传任何参数。定义了夹具，就可以把刚体的形状赋值给夹具了：

```
fixtureDef.shape=circleShape;
```

最后，通过世界来创建具体的刚体：

var theBall:b2Body=world.CreateBody(bodyDef);

这样创建的刚体球就算加入到 Flash Box2D 的世界中了。然后通过刚体实例来创建刚才定义好的夹具：

theBall.CreateFixture(fixtureDef);

这样就创建完了有血有肉的刚体了。当然还有很多其他的具体属性，在这里不一一列举了，详细的应用可以参考 Flash Box2D 的 API 文档。

3.3 常用物理刚体的创建

通常创建物理游戏都是要创建一个可见对象，然后为这个可见对象创建一个与之相匹配的物理对象。而创建物理对象又需要经历一套复杂的流程：

1、创建 Body definition,

2、创建形状

3、创建 Fixture definition

4、通过世界创建 Body

5、通过 Body 创建 Fixture

通过普通的流程来创建物理游戏会遇到以下几类问题，下面将一一细述。

3.3.1 简单多边形的创建

简单多边形是指矩形，圆形，凸多边形等 Box2d 原生支持的形状。这类多边形通常可以通过 create body by type 和 create body by vectories[16]来创建。

在 FLASH 开发物理游戏的时，要明确二个基本的概念，刚体与显示对象 DisplayObject。你可以理解刚体就是 BOX2D 里的 b2Body，而显示对象则可以先简单的理解为 Sprite，他们是有着本质的区别的。

刚体是 Box2D 中的一个物理模拟对象，不是可显示对象，但是可以通过 b2DebugDraw 来进行调试模拟，这时刚体会显示成线框。刚体在运动过程中是以米为计量单位的，它通过 b2World.createBody()来创建。

显示对象 Sprite 很从字面上理解它就是显示对象，通过 addChild 添加到舞台就可以被显示，它在运动过程中是以像素为计量单位的，它可以通过 new 来创建，然后 addChild 添加后直接显示在舞台上。

Box2D 主要是对刚体的物理性模拟，为了让刚体的运行可视化，会根据刚体的状态改变来逐帧更新显示对象的状态，这样就产生了显示对象具有物理属性的感觉。

刚体创建的过程，与 AS 中 new 一个对象，addChild 到舞台中显示比起来要复杂很多，刚体的创建可以看做是一个工厂模式，根据你的需求(b2BodyDef)生产指定的刚体。图 3-1 是刚体工厂模式的示意图。

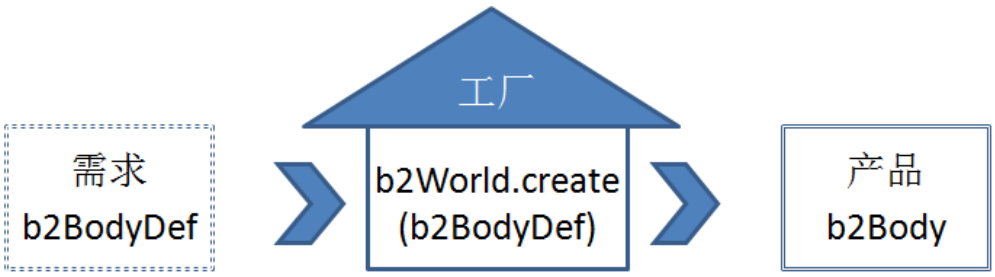

图 3-1 Flash Box2D 刚体工厂模式

Fig. 3-1 Flash Box2D rigid body factory model

刚体的创建过程可以分为以下几个步骤：

1. 创建刚体需求 b2BodyDef。

var bodyRequest:b2BodyDef = new b2BodyDef();

在这个需求中，主要包括下面的内容[16]：

a) position：刚体的坐标位置，单位是米 m。

b) angle：刚体的角度，单位是弧度。

c) userdata：用来存储应用中所特需的信息。

d).....

2. b2World 工厂用 createBody 方法创建刚体产品，并自动将其添加到 Box2D 世界中，然后返回该刚体。

body=world.CreateBody(bodyRequest);

3. 定义刚体物理特性。

虽然已经成功添加了一个刚体。但是这个刚体密度有多大，具体是什么形状还没有定义，刚体的物理属性不是与生俱来的，而刚体的形状则决定了刚体碰撞的边界，因此需要用一个 b2FixtureDef 的对象来指定刚体的物理属性。b2Fixture 的创建过程同样也是一个工厂模式。图 3-2 为夹具工厂模式的简单表示。

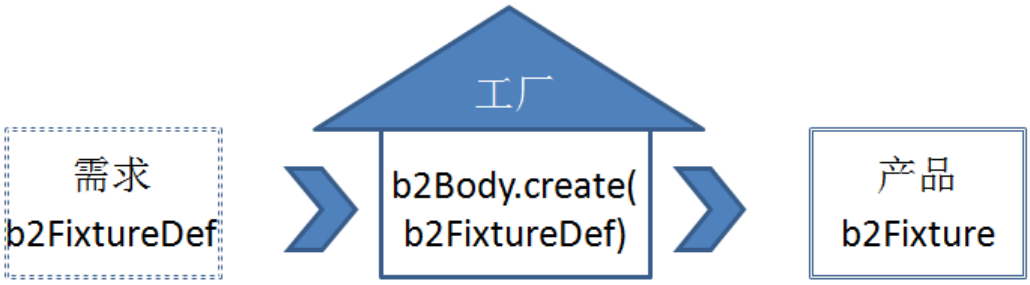

图 3-2 Flash Box2D 夹具工厂模式

Fig. 3-2 Flash Box2D fixture factory model

这个创建过程与 b2Body 的创建过程是一样的。形状需求 b2FixtureDef 可以定义以下内容[16]：

a) density：质量。

b) friction：表面摩擦力。

c) restitution：表面张力，这个值越大，刚体越硬。

d) shape：设置刚体为形状。

e) .....

4. 定义刚体的形态。

前面已经用到了 shape 来指定刚体的形状，但是这个 shap 也是需要义和创建的。通常游戏中用的是 b2shape 的子类，主要有 b2PolygonShape、b2CircleShape，下面以 b2PolygonShape 为例说明

```
var boxShape:b2PolygonShape = new b2PolygonShape();
boxShape.SetAsBox(width/2, height/2);
```

常用的除了 SetAsBox，还有 SetAsVector()。定义好之后传递给 b2FixtureDef 就可以了。

从上面的创建过程来看还是比较麻烦的，不见得每次创建一个刚体都要手动去设置各值，以后的开发可以考虑比较方便的方案。

3.3.2 简单多边形的快速创建

开发物理游戏需要创建的物理对象通常比较多，如果通过普通流程逐一进行物体创建，比较的费时费力，而且维护更新也比较困难。

如果可以先实现静态物体的创建，然后在需要物理属性的物体上注入 Flash Box2D 的物理特性，就可以达到点石成金的快速效果。

Flash Box2D 模拟现实世界的计算过程其实跟 Flash 显示层是没有关系的，它只是运行在显示层背后的一种完全数字逻辑的运算，物体的显示层就好比一个人的外形躯体，而 Box2D 则是外形躯体的内在精神，是支配外形去实现物理运动内在力量，因此，完全可以考虑将物理特性注入表现层的 UI，来实现整个物理对象的创建。

3.3.3 创建线条刚体

BOX2D 的作为刚体的物理引擎，通常都是用来模拟块状物体，但是在实际的游戏开发中又经常会碰到线条类的物体，比如绳子，树枝等等。对于这些物体，基本的思路其实就是用多个线段组合起来模拟具有一定柔性的线条。

从理论上讲，线条是由无数个点组成的，可以创建圆来代替点，进而形成线，但是从性能效率方面来考虑，实际开发中其实只要用多个线段组合成线就可以了。

图 3-3 为线条在创建刚体时的组合模拟，可以假设每个线断长度为 segmentLength，线段的长度越短，线段的数量越多，那么线条模拟的就越逼真。

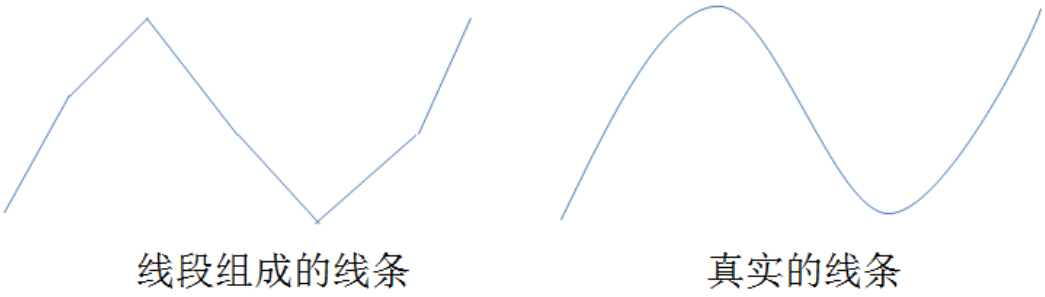

图 3-3 线条模拟

Fig. 3-3 Analog lines

创建线条的算法如下：

1）先定义线段的长度 segmentLength

2）在鼠标点下后，开始绘制线条，并记录之前的鼠标坐标位置为 prePoint

3）在线条绘制过程中，持续检测鼠标坐标 curPoint 与前一个点 prePoint 之间的距离为 distance，当 distance 大于线段长度时，创建一个新的线段，将线段添加到 segmentList 中，并将 curPoint 赋值给 prePoint，这样循环往复，直到线条绘制完成。

4）鼠标弹起后，遍历所有的线段，并利用多边形组合法，创建所有线段刚体，然后组合成一个完整的线条。

3.4 Box2D 特殊效果的实现

通过上面对 Flash Box2D 的基础技术分析，可以看到，在每次开发一款游戏的时候，都会需要对物理世界进行初始化，而这一过程在很大程序上是相似的，为了避免不必要的重复劳动，通常可以将物理世界的初始化抽离出来，设计在游戏框架中，这样就便于每次开发时应用了。

另外也可以根据不同的物理环境，比如，水体，风洞等特殊场景，设置不同的物理参数，然后通过继承，合成的编程模式加以扩展。

下面具体分析一下特殊场景的液体浮力效果。

3.4.1 浮力效果

通常不会是整个场景都是在液体里，而是一部分在自然空间，一部分需要模拟液体浮力。虽然 Flash Box2D 提供了浮力参数，但它是对整个世界起作用的，这显然不能满足游戏开发的需求，因此如果能够定义一个区域来实现浮力效果，那么就可以实现类似池塘的场景了。

通过多次的假设验证，最后发现可以通过创建一个跟设想的池塘一样大小的传感器，然后将将浮力控制附加在这个传感物体上，只要其他物体掉落到传感区域上，就模拟浮

力效果，图 3-4 就是浮力效果的模拟。

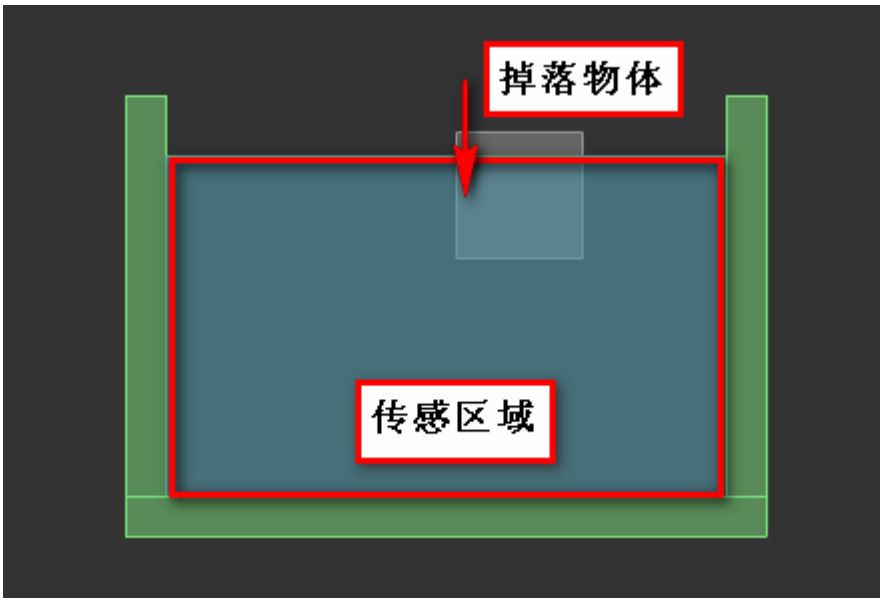

图 3-4 浮力效果

Fig. 3-4 Buoyancy effect

3.4.2 流体效果

在探讨物理效果时，通常将液体和气体统称为流体。流体通常是由无数个粒子组合而成的，因此粒子系统几乎是每个游戏引擎中必不可少的部分。在计算机上的生成与显示流体效果，经常是使用粒子系统模拟的，主要包括火、爆炸、烟、水流、火花、落叶、云、雾、雪、尘、流星尾迹或者象发光轨迹这样的抽象视觉效果等等。

实现粒子系统，首先需要对粒子系统进行一个简单的划分，它可以简单的分为两种：点粒子和四边形粒子。点粒子就是说粒子系统中的所有粒子都是由一个点构成的，因此这个点通常就需要不同的颜色来展示，四边形粒子实际上就是将一个粒子纹理给映射到一个四边形上，作为一个粒子对象，当然可以看出点粒子的实现肯定比四边形粒子的实现简单很多，下面分析这两种类型的粒子的实现。

虽然 Flash Box2D 只能模拟刚体，但是可以用一系列小的刚体的流动来实现流体的效果，图 3-5 很好的表示了用粒子表示液体的情形。

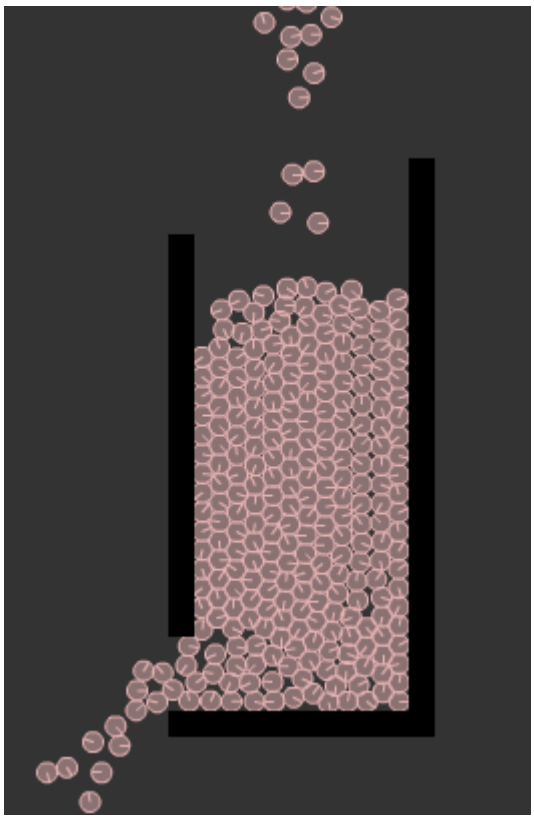

图 3-5 液体粒子效果

Fig. 3-5Liquid particle effects

为了让它具有真实的质感，可以更改贴图，图 3-6 就是增加白色贴图后的效果。

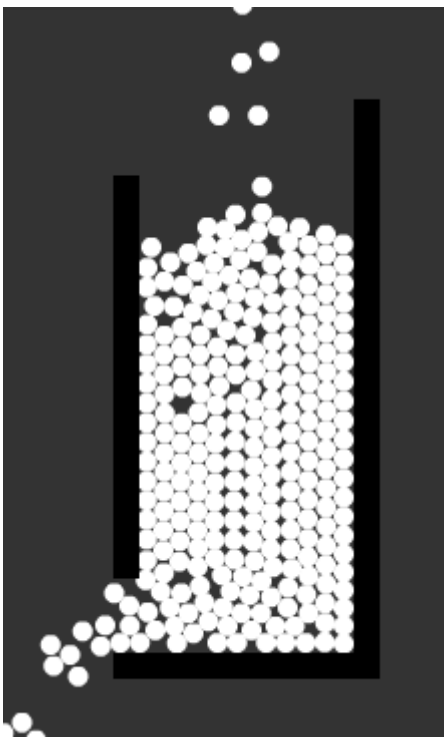

图 3-6 粒子贴图后效果

Fig. 3-6 The effect of particle after mapping

为了让流体更真实，可以加入一些模糊，图 3-7 就是加了模糊后的粒子效果。

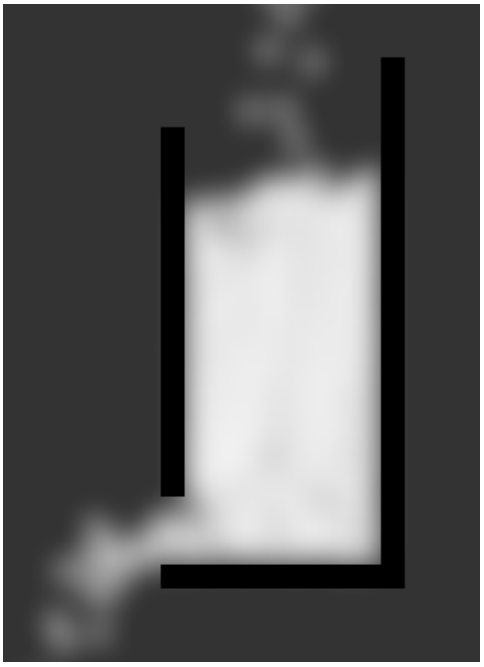

图 3-7 粒子模糊后效果

Fig. 3-7 The effect of particle after blur

现在有点像雾的感觉，如果再加入一个阀值来改变颜色，并且更新一个更好的背景则会更加具有真实感，图 3-8 就是改变颜色和背景之后获得的效果。

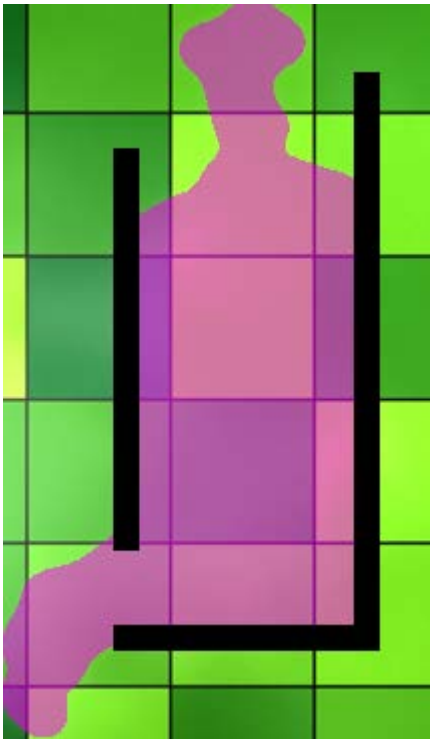

图 3-8 真实液体效果

Fig. 3-8 The real effect of liquid

3.4.3 刚体的缩放效果

因为 Flash Box2D 模拟的是物理场景及运动的数据，它们不是显示对象，因此也就无法使用 width、height 或 scaleX、scaleY 来缩放刚体的大小了。而且 Flash Box2D 物理引擎也没有提供相应的 API 调用，因此给相应需求的开发带来了不小的麻烦。

通过研究 Flash Box2D API 发现，b2Body 类并没有提供直接调整刚体尺寸的属性或方法，而是要通过 b2Shape 类的一些方法来实现，所以首先要用 b2Body.GetFixtureList().GetShape()获取刚体的图形，然后根据图形的类型进行不同的设置。

b2Shape 的子类有两种，一个是 b2CircleShape 类，这个图形的尺寸调整起来比较简单，直接调用 b2CircleShape.setRadius()就可以重新设置新的半径。参数可以设置为 b2CircleShape.getRadius()*scale 来进行比例缩放。另一个是 b2PolygonShape 类，因为多边形的顶点数量和位置不同，形成的图形也不同，所以没有类似于 SetRadius()的 SetWidth()或 SetHeight()方法。实际上，可以把多边形分解成多个顶点，多边形等比例缩放时，顶点到中心点的距离也是等比例缩放的，图 3-9 就是等比缩放的示意，所以只需要用顶点向量的 Multiply()方法乘以指定的缩放比例即可。

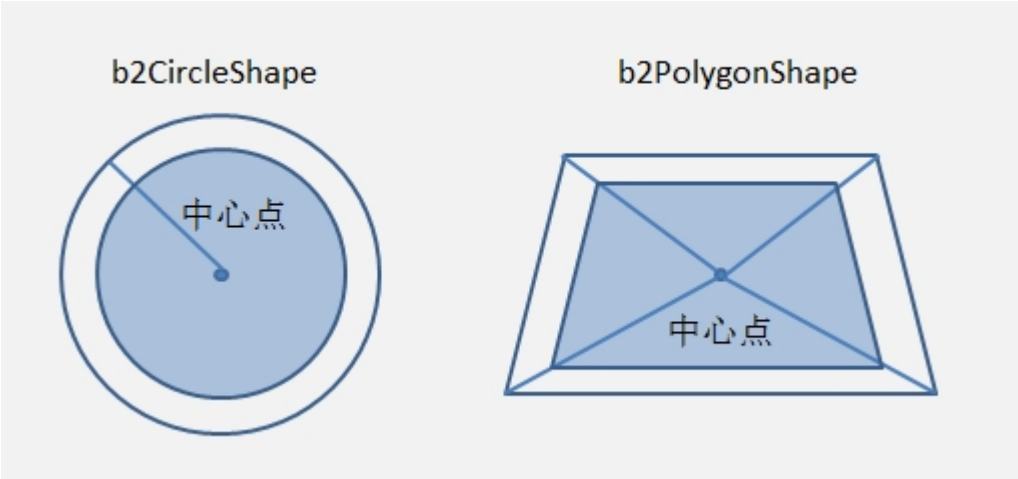

图 3-9 等比例缩放

Fig. 3-9 Proportional scaling

3.5 本章小结

通过对 Flash Box2D 的核心概念的了解，以及物理世界的初始化，物理刚体的创建，可以说大体上能够创建基本的基于 Box2D 引擎的游戏了。但是也不难发现，基本上每次开发一个物理游戏，都要重复物理世界的初始化。另外对于游戏中的每一个物理刚体，如果都按照上面的步骤去创建的话，也是相当繁琐的过程。因此，下面就需要去具体分析一下，看看游戏框架怎样设计，才能够使游戏框架在开发物理手机游戏时既方便实用，又性能卓越。

4 EasyBox2D 游戏框架的设计

游戏框架设计目标是设计一个基于 Starling 框架，支持 Feather 扩展 UI 组件类库，对 FLASH BOX2D 进行进一步封装的轻量级的，易于扩展的，简单易用的 FLASH 手机游戏框架。因此将这个游戏框架命名为 EasyBox2D，以下描述中 EasyBox2D 就是框架的名字。EasyBox2D 能够显著简化物理刚体的初始化，并且提供了极为简单，具有多种选择性的刚体皮肤绑定方案。

4.1 EasyBox2D 游戏框架总体结构

在 FLASH 推出硬件加速渲染的 Starling 框架之前，FLASH 都是应用 CPU 来运算的，当时的 FLASH 平台主要有 FLASH PLAYER 和 FLASH AIR，而 BOX2D 作为时下最流行的物理引擎，也推出了 FLASH AS3.0 版本，随着硬件技术水平的发展，越来越多的应用、游戏都提出了硬件加速位图显示的解决方案，ADOBE FLASH 也与时俱进的推出了基于 Stage3D 的渲染引擎 Starling 框架，并且集成到了随后更新发布的 FLASH PLAYER 和 FLASH AIR 这二个运行平台中。而 Feather UI Lib 作为 Starling 的扩展 UI 库，通过 Starling 框架，利用 GPU 的强大能力渲染组件，实现了更加平滑和友好的用户界面体验。

EasyBox2D 以组合[15]的形式将 Starling、Feather 包含其中，并在 FlashBox2D 的基础上进一步封装了 BOX2D 的功能，以实现对物理刚体的初始化简化，提高游戏开发的效率，降低了开发的难度。图 4-1 就是 EasyBox2D 游戏框架总体结构。

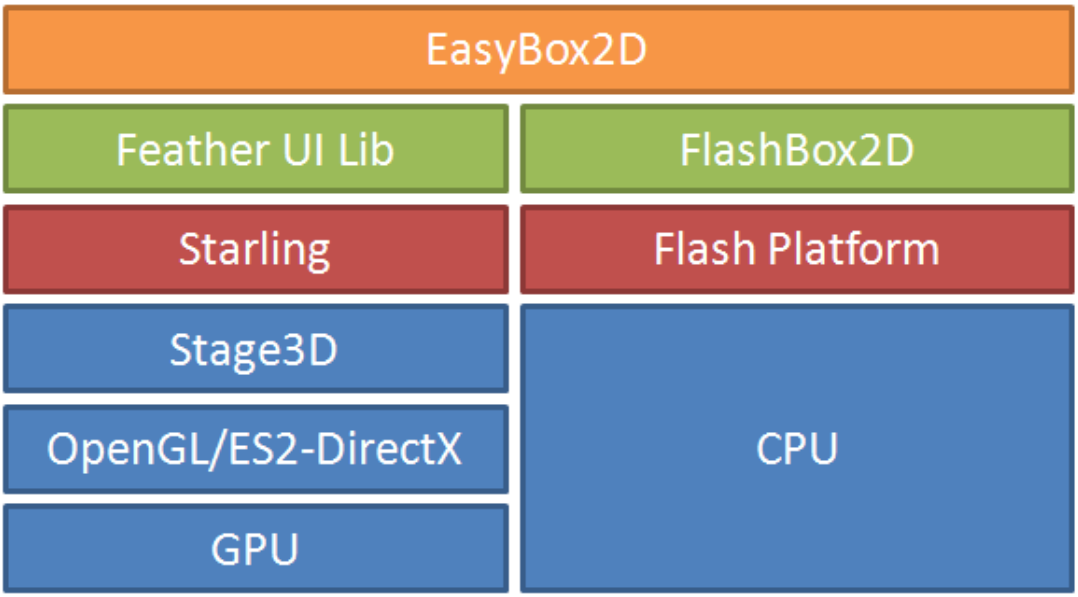

图 4-1 EasyBox2D 游戏框架总体结构

Fig. 4-1 The overall structure of EasyBox2D game framework

4.2 EasyBox2D 结构设计

4.2.1 EasyBox2D 总体结构

EasyBox2D 总体结构主要分二大层，界面层主要提供相应的调用接口，以方便游戏业务逻辑获得物理引擎的支持，在界面层下面的则是具体提供各种功能的功能层。功能层又分为 5 个大的模块，由 FLASH 原生的显示模块，刚体创建工厂模块，Contact 模块，数据模块以及提供数学几何运算的工具类模块。基本的结构设计可以从下面的图 4-2 中得到初步的了解

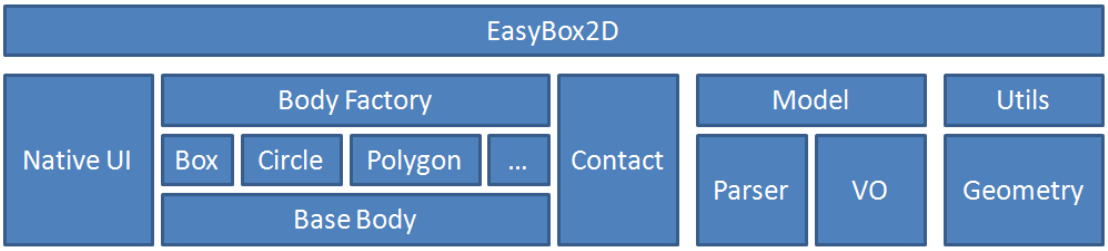

图 4-2 EasyBox2D 总体结构

Fig. 4-2 EasyBox2D overall structure

4.2.2 EasyBox2D 管理类

为了便于游戏开发中应用 FLASH BOX2D 来进行物理物件的创建与使用，EasyBox2D 将提供唯一的入口来完成基本上所有的物理特性的创建与管理。对 BOX2D 的初始化，Debug 模式的切换，边界的设定等常用的功能都会在统一的入口文件中提供，并且用户如果觉的预定义的方法不能完全满足游戏开发需求的话，也可以通过重写来实现特定的功能。

结构图 4-2 中最上面的模块就是框架提供的唯一入口，它相当于一个管理器，负责 BOX2D 的初始化，刚体的创建，管理等等。

4.2.3 FLASH 原生显示列表

FLASH 原生显示的支持主要是为了在 FLASH BOX2D 调试模式中提供各种调试数据的绘制与显示，这是由 FLASH BOX2D 物理引擎所定义的调试模式所决定的。

当然除此之外，因为是基于 Starling 开发的手机游戏框架，而 Starling 所有的显示内容都不属于 FLASH 原生显示的范畴，而是位于在 FLASH 原生显示列的下面一层，如果有些功能 Starling 不支持，或者通过 GPU 运算效率比较低的显示内容，也可以考虑

使用 FLASH 原生显示列表。这一模块就是为了方便对 FLASH 原生显示列表的调用，使得 Starling GPU 显示与原生 FLASH CPU 显示能够融合在一起。

4.2.4 Body 工厂

通过 FLASH BOX2D 技术基础部分的描述，相信大家对刚体的生成有了较深入的了解，那么在游戏开发中如果每次都重复构建 b2BodyDef，b2FixtureDef，b2Shap，然后再由 b2World 创建的话就效率比较低了，所以考虑将基本的刚体创建封装在框架中，以提高游戏开发的速度。

刚体工厂会提供包括方形，圆形，凸多边形等基本形状的创建，如果这些不能满足开发需求的，也可以通过继承 BaseBody 来扩展。

所有刚体的创建都会按照统一的格式来定义，用户只要提供刚体的类型和相应参数就可以完成创建，比如创建一个 BOX 的 API 调用 easyB2D.addBox(property)。

4.2.5 Contact

Contact 模块主要是提供对碰撞的个性化处理。

因为碰撞的处理具有很多的不确定性，对 hp 的伤害规则通常也比较复杂，所以框架只是提供了碰撞的基本信息，并以事件的形式向外抛出，在界面层去做个性化的处理。

4.2.6 数据模块

数据模块主要用来处理与物理引擎相关的数据，其中 B2UserData 是对各种类型物体的物理特性的数据描述，当然为了方便相关数据的存取，也可以加入表示物体生命值的变量 hp， 以及可见对象的 Reference 引用等信息。

4.2.7 工具类模块

工具类模块主要提供数学几何相关的计算支持。

因为在 Box2d 只支持矩形，圆形和凸多边形的创建，而且对于凸多边形的顶点数据也有顺序的要求，必须是顺时针的顶点数组，这给非基本型物理形状的创建带来了很大的不便。为了对任意多边形向凸多边形的转化，就需要进行很多数学、几何运算，这些计算公式的 API 支持都会由工具类模块提供。

4.3 抛投类游戏主要功能设计

4.3.1 抛投功能设计

从市场比较受欢迎的 Flash 抛投类游戏的分析中可以看出，计算抛物线的方法有多种，下面将主要的几种计算方法进行分析：

（1）二维空间中计算抛物线最简单的方法就是将向量拆分成竖直分量和水平分量，然后分别计算。由于分量之间是完全独立的，所以可以分别对其进行计算。在 Flash 游戏中可以，可以更直接的理解为显示对像每帧向左移到 5 个像素，也就是说该物体在 X 轴上的速度向量为 Vx=5。

抛投的关键是如何确定速度向量的大小，而在模拟物理运动过程时，所应用的公式是：旧的位置+速度向量=新的位置，那么计算速度向量就可以由前面的公式变化而来：速度向量=新的位置-旧的位置，在拖拽的过程中，会在每一帧上产生新的位置，用当前帧的位置减去前一帧的位置，就可以行道这一帧所移动的距离，也就是物体的速度向量。当然在实际开发中还要考虑重力作用和摩擦力，这样就能使物体运动更加真实可靠。

（2）先根据条件计算出抛物线公式，在抛投类游戏中通常还会考虑到的几何学知识就是抛物线，抛物线是轴对称的曲线，其开口可以是任意方向。

抛物线表达式通常有标准式和顶点式

标准式：$y = ax^2 + bx + c$，a，b，c 为常数 （4-1）

顶点式：$y = a(x - h)^2 + k$，顶点为（h，k），对称轴为 x=h （4-2）

在理想状态下，可以认为抛物线的形状是由三个因素决定的，第一个是顶点，也就是抛物线的顶端，第二个是对称轴，它经过顶点，并将抛物线对称分割，第三个是 a 常数，常数 a 代表了抛物线的开口方向和开口大小，如果 A 是正数，则抛物线开口向上，如果 A 是负数，则抛物线的开口向下。A 的绝对值越大，抛物线的开口越小。

（3）应用成熟的游戏引擎，这也是比较高效的选择，可以选择 Box2D 物理引擎来模拟物体的抛投。Flash Box2D 提供了 API，可以直接设置物体在水平方向各垂直方向上的线性速度。

4.3.2 碰撞功能设计

基于对 Flash 抛投类游戏的分析以及 Flash Box2D 物理引擎的了解，游戏碰撞功能的设计将以 Flash Box2D 物理引擎为基础。

Flash Box2D 物理引擎的碰撞检测分为两个检测阶段：粗略检测阶段主要是利用物体的包围盒来进行粗略的层次碰撞检测，目的是快速过滤没有发生碰撞的物体。如果物体的包围盒发生了碰撞，那么就进入到精细检测阶段进行更精细的判断。

精细检测阶段也可以继续包围盒的思路，把物体细分，对于物体的每个部件确定包围盒，然后再继续进行碰撞检测，直到系统规定的，可以容忍的误差范围。

4.3.3 破碎功能设计

为了实现真实、细致的物体破碎效果，破碎功能舍弃了开发复杂度较低的预渲染帧策略，采用了 Flash Box2D 物理引擎的超强计算能力，来实现物体破碎效果。

Flash Box2D 要实现物体破碎效果，通常会经过以下步骤：

1、b2world.RayCast(callback:Function, point1:b2Vec2, point2:b2Vec2)切割

利用光线投射物理世界，找出所有在光线路径中的夹具，你可以在回调函数中控制是否获得了最接近光线发射源的刚体。光线投射会忽略包含出发点的形状。图 4-3 是光线投射的 3 种可能情况的示意。

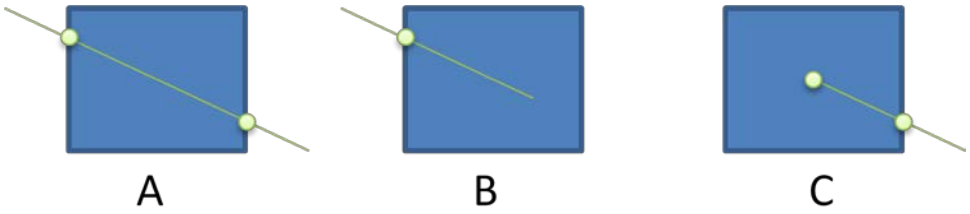

图 4-3 光线投射

Fig. 4-3 Light projection

图 4-3 中，只有 A 图会触发回调函数，而 B、C 则是会被忽略的。

回调函数会有四个参数 function Callback(fixture:b2Fixture, point:b2Vec2, normal:b2Vec2, fraction:Number):Number

fixture：被投射线射中的刚体的夹具

point：初始交点，也就是射线进入刚体的入点

normal：初始交点处的法向量

fraction：从射线的起点到初始交点的长度占整个射线的比例

两点决定一个直线，在数学上知道了两点，再定义直线上的其它点，常使用参数方程。也就是定义 P(fraction) = p1 + fraction * (p2 - p1)。当 fraction 为 0 时，就代表 p1，当 fraction=1 时，就代表 p2。这样的定义下，两点之间的线段就是参数从 0 到 1 之间变化。参数小于 0，表示反向的点，大于 1 就表示正向超出线段的点。maxFraction 就表示要测试的点对应的参数是在[0, maxFraction]内。数学上很喜欢将一些变量归结成 0 到 1 之间变化，这叫做规范化。处理问题的常用手段是用某个变换(这里说的变换是广义的)将变量归结成 0 到 1 之间，再在规范化之下计算，之后再用个反变换得到原问题的答案。上面说的直线参数化，可以看成规范化的一种。

因为每次射线切割对象时都会调用 laserFired 方法，而且切割到的对象不止一个，所以如果你不希望射线继续切割，返回 0，那么射线会停止继续检测。返回 1，射线会

继续切割其他的对象。

2、切割完之后会形成二个点，一个是射线的入点，一个是出点。将位于射线上方的多边形顶点和射线的入点，出点共同组成一个新刚体的顶点，位于射线下方的多边形顶点和射线的入点，出点组成另一个新刚体的顶点

3、因为 Flash Box2D 在创建多边形时只支持顺时针排序的顶点序列，所以要对新的刚体的顶点做顺时针排序。

4、根据多边形面积筛选多边形。先将多边形按顶点顺序切割成多个三角形，然后根据 3×3 矩阵的行列式与三角形面积的关系式，依次计算出三角形的面积并累加，计算出多边形的面积，根据多边形面积的大小，剔除掉面积较小的多边形，以提高游戏运行效率。

5、最后根据筛选出来的多边形的顶点创建出新的多边形刚体。

4.4 游戏编辑器的结构设计

游戏编辑器主要为用户提供一个能通过图形化的界面，对新建的或者已有的关卡进行编辑，设计并保存关卡数据的平台[19]。游戏编辑器不仅能让游戏体验者能玩自己设计的关卡，更可以分享给其他游戏体验者，当然游戏开发者也能通过编辑器快速搭建游戏场景，以便在开发中不断测试各种环境下的游戏性能。具体来说，EasyBox2D 游戏编辑器包括以下几个方面：

1）场景数据格式的设计与存储。

2）导入设计好的场景数据，在此基础上对场景数据进行编辑[19]。

3）根据预先设计好的元素来新建场景数据，用户可以通过拖拽的方式进行搭建。

4）根据编辑器提供的画图功能来进行场景的绘制。

5）编辑完成后对场景数据进行预览与保存。

图 4-4 可以看作是 EasyBox2D 编辑器的总体结构

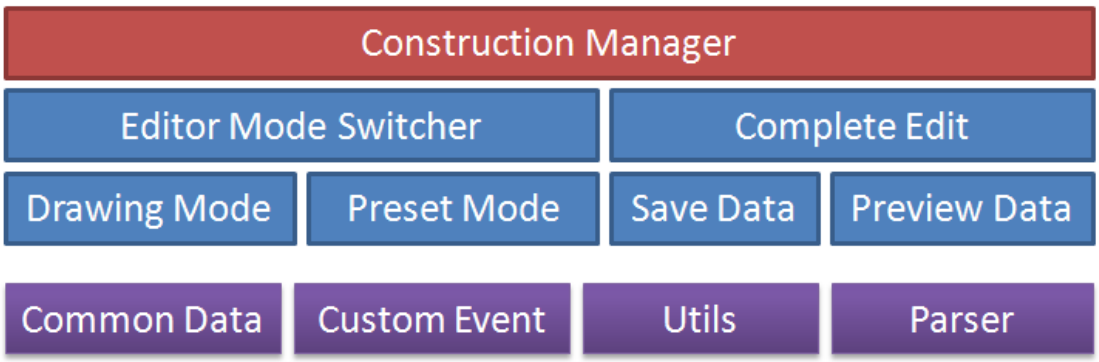

图 4-4 EasyBox2D 编辑器总体结构

Fig. 4-4 EasyBox2D editor structure

Construction Manager 作为编辑器的操控中心，负责各模块间的互动合作。下面具体说明各功能模块的功能设计。

4.4.1 场景数据格式的设计

为了便于场景数据在 FLASH EasyBox2D 环境下的使用，场景数据将使用 JSON(JavaScript Object Notation)的数据格式来保存。JSON 作为轻量级的数据交换格式，比较容易阅读和编写，对于计算机的解析和生成支持也比较友好[20]。JSON 采用完全独立于编程语言的文本格式，主要表现为以下两种结构：

（1）“名称/值”对的集合（A collection of name/value pairs）。在不同的程序语言中，它被关联于不同的描述，主要有纪录（record），结构（struct），对象（object），字典（dictionary），哈希表（hash table），有键列表（keyed list）等等。

（2）值的有序列表（An ordered list of values）。这主要体现在一系列数据的有序组合，在很多编程语言中，作为数组（array）来使用。

因为这些数据结构都是比较常用的，所以绝大多数计算机编程语言都会以某种形式的对象来使用它们。这也使得 JSON 数据可以在大部分编程语言之间进行交换。EasyBox2D 采用“名称/值”对的集合结构来组织数据。

4.4.2 场景数据的搭建模式

场景数据的搭建模式主要是提供预先设计好的构建部件，通过选取相应部件，然后以拖拽的方式在场景是进行搭建，总体结构如图 4-5：

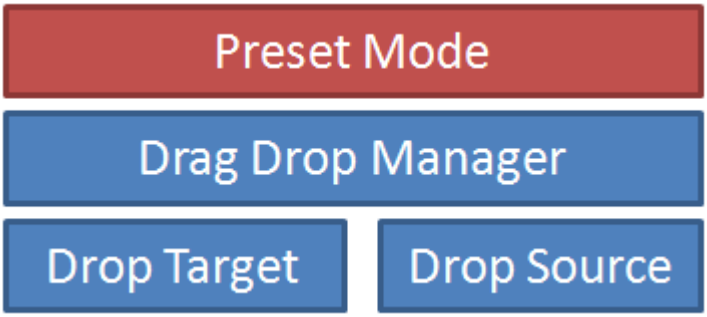

图 4-5 搭建模式结构图

Fig. 4-5 The structure diagram of building model

4.4.3 场景数据的绘制模式

如果游戏场景只需要 Box2D 原生形状来进行搭建，只是对各形状位置，旋转等有要求的，可以绘制模式来搭建场景。

绘制模式是通过选取适合的形状，通过在屏幕上拖拽来实现形状，大小，位置控制的场景搭建模式，主要提供方形，圆形，凸多边形等 Box2D 支持的原生形状。在绘制过程中会有绘制模式，拖拽移动模式，旋转模式，强制正方形模式，强制贴合整数位置

模式等的切换功能，图 4-6 为绘制模式的总体结构图。

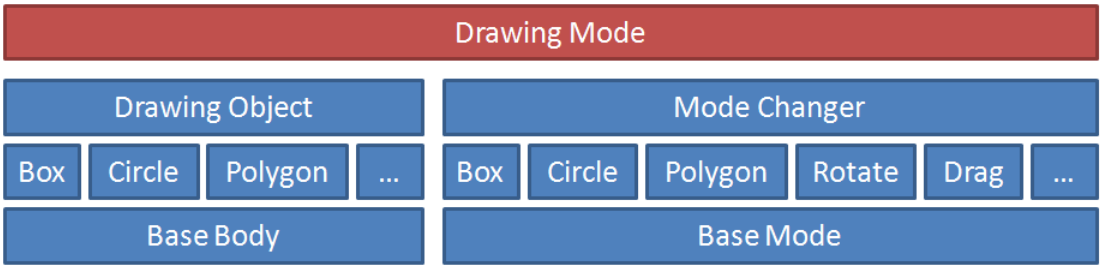

图 4-6 绘制模式结构图

Fig. 4-6 The structure diagram of drawing model

不论是绘制形状，还是绘制模式，都是允许在具体游戏开发中根据实际需求进行扩展的。

物理游戏不仅是要创建矩形，圆形，凸多边形等 Box2d 原生支持的形状，还会有很多不规则物体，比如凹多边形，复杂图形等等。但是 Flash Box2D 原生的 API 是不支持凹多边形等复杂图形的创建的，这主要是受限于 Box 2D 的自身结构设计。然而，这在现实项目中是不可避免的。受到其他一些第三方类库的启发，解决思路基本是这样的：先将复杂的凹多边形切割成一系列的凸多边形，然后各个创建，最后将这一系列凸多边形组合成原来的凹多边形。

4.4.4 特殊图形的扫描模式

如果已经有了具体的图形，那么可以考虑通过手工的方式，预先设置刚体数据，然后应用搭建模式来创建场景。也可以将这个图形作为参考，通过绘制模块中绘制多边形顶点的方法来绘制场景数据。但是如果考虑更自动化的方式，就可以考虑使用图形扫描的方式来获到多边形的顶点数据，然后应用上面提到方法，根据多边形顶点来创建具体的图形刚体数据了。

4.4.5 场景数据的预览与保存

场景数据的预览主要是为了便于在搭建过程中对最终生成游戏场景的预览，预览过程是调用 EasyBox2D 进场景数据进行解析，生成的过程。

场景数据的保存可以选择数据库存储，也可以选择本地文件存储，具体可以根据游戏开发实际需求选择。

4.5 本章小结

（1）EasyBox2D 实现了对物理世界的封装，减少了每次开发游戏的重复对物理世界的初始化操作。同时 EasyBox2D 还提供了通过显示对象快速查找物理对象的公开方

法，方便了对物理对象的控制。

（2）用 Factory 实现所有 Body 的创建，只需要准备好相应物理属性，就可以非常方便的实现基本刚体的创建。对复杂凹多边形刚体的创建则提出了分割成凸多边形最后组合的解决方案。当然使用图形扫描也是更自动化的方向。

（3）游戏编辑器设计 EasyBox2D 游戏编辑器实现基本的游戏场景编辑需求，可以在绘制模式和搭建模式间进行切换，并且支持预览与保存。对于绘制模式和搭建模式的混合编辑，则是后续扩展的方向。

5 EasyBox2D 游戏框架的实现

5.1 EasyBox2D 游戏框架的逻辑实现

5.1.1 世界封装类

世界封装类是进入物理世界的入口， 主要是为了简化游戏开展中每次都要手动去初始化物理世界的过程， GameWorld 类具体实现了对物理世界的封装。

GameWorld 类是继承 b2World 的，并且是单例类，这样能保证了游戏中始终只有一个世界。因此在游戏开发中就不需要去手动初始化世界了，只要在需要使用 b2world 的地方，通过 GameWorld.getInstance() 方法。

图 5-1 是 GameWorld 的类图。

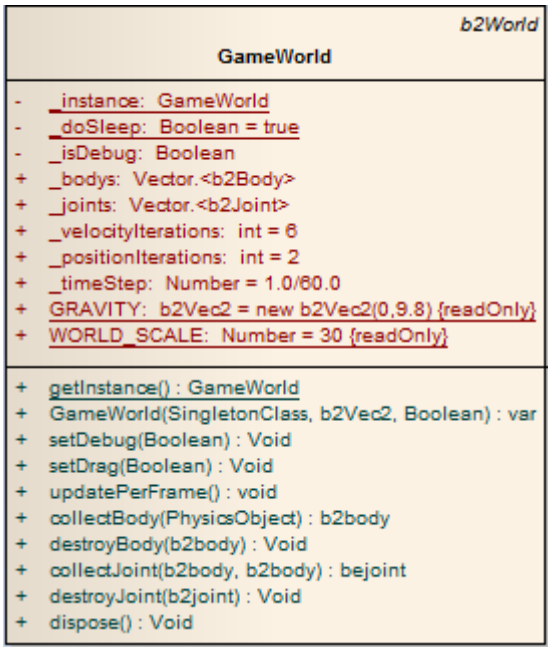

图 5-1 GameWorld 类

Fig. 5-1 The class of GameWorld

GameWorld 类初始化了世界，提供了相关属性的设置，比如 Gravity 等。并且提供了通过显示对象来快速获取物理对象的公开方法 getBodyByDO(displayObject:*):b2body，这极大的方便了对物理对像的调用。

5.1.2 EasyBox2D 工厂类

通过 GameWorld 类来实现对物理世界的封装，这只是游戏框架的开始，下面要来实现对物理刚体创建的工厂类。

在游戏框架中主要是应用工厂模式[18]来实现对物理刚体的创建，model 包主要是物理数据的格式，manager 包提供了相对独立功能的管理类，math 包将独立处理一些几何图形相关的数据，contact 和 event 包为以后实现自定义的碰撞方案预留了接口，utils 包主要是提供一些静态的工具类方法，以方便在开发中使用。

Factory 是方便创建物理对象的工厂，主要包括基本形的创建和任意多边形的创建，为了在将来的开发中能更好的扩展， 这里采用了工厂模式的设计方法，如果以后有一些特殊类型的物件，也能很方便的通过继承 BaseBodyFactory 类来扩展，只要复写 initShap()方法来提供特殊的形状。图 5-2 为刚体工厂模式的类图。

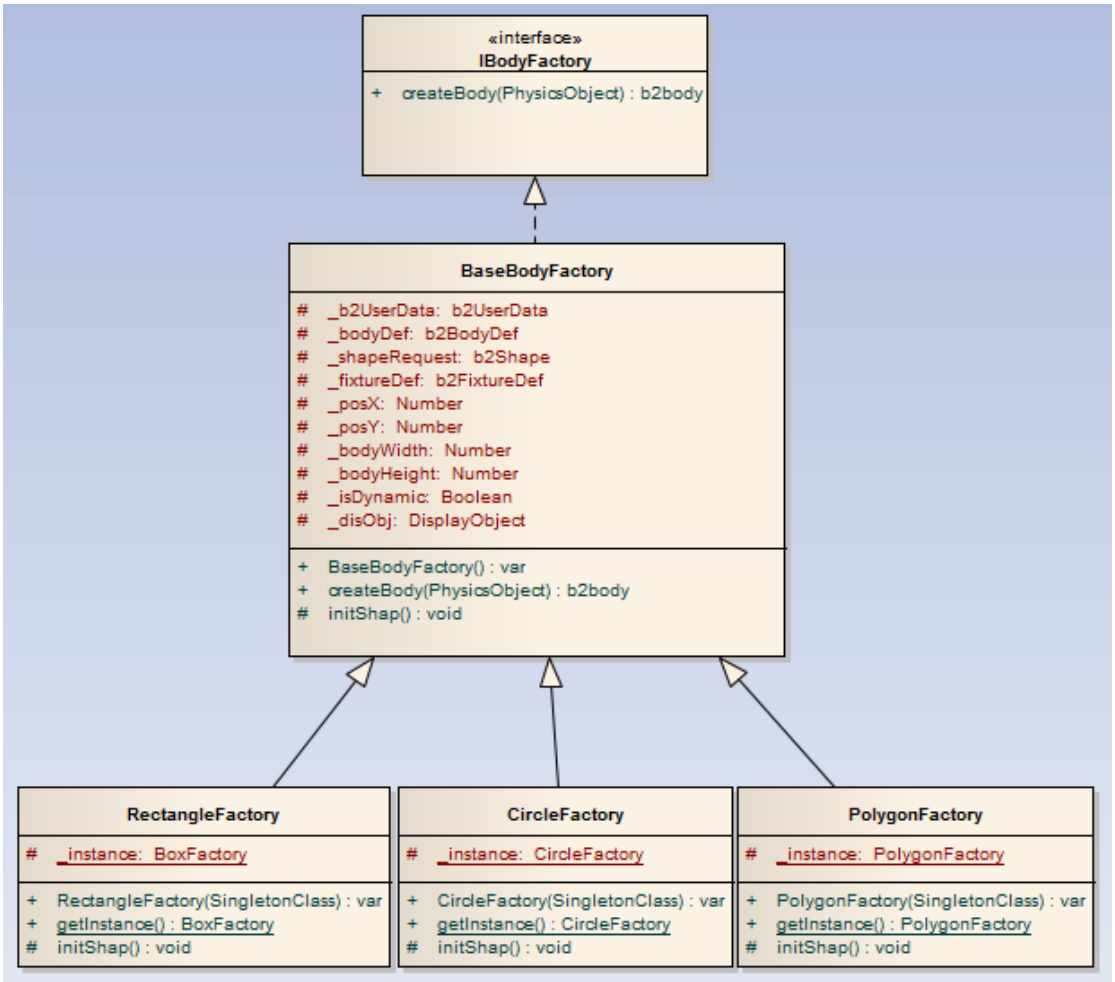

图 5-2 刚体的工厂模式

Fig. 5-2 Factory modal of rigid body

5.1.3 model 包

Model 包内主要是实现了数据的预处理以及对数据的保存，图 5-3 为 Model 数据类图。

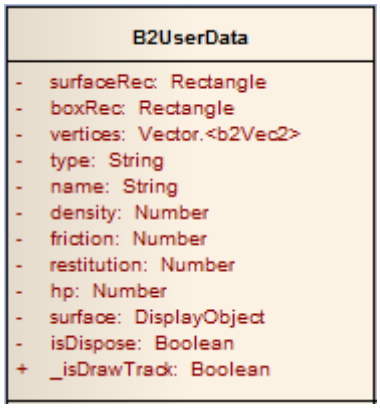

图 5-3 Model 数据

Fig. 5-3 Modal data

5.1.4 debug manage 包

debug manage 包主要提供对物理对象调试的管理

因为 Box2D 的刚体是不可见的，这对于调试是个问题，为了能够使刚体可视化，在开发过程中可以将世界设置成 Debug 模式，Box2D 在 Debug 模式下会绘制线框图来表示刚体的形状和位置，具体有 6 种显示选择：

e_shapeBit　绘制形状

e_jointBit　绘制关节连接

e_aabbBit　绘制刚体的边界框线

e_pairBit

e_centerOfMassBit

e_controllerBit

为了便于对调试模式的管理，可以设计 DebugDrawManager 类，用来控制调试模式的开启与关闭，能及对各种调试数据的输出选择，这有助于开发人员方便的管理 Box2D 刚体的调试。

在物理游戏中鼠标拖拽功能也是非常基本，常用的功能，因此对这个功能进行了封装。DragManager 类提供了对物理对象的鼠标响应， 当鼠标点击的时候，自动获取在鼠标下的物体，并提供了拖拽的功能。

5.1.5 多边形绘制辅助工具包

多边形绘制辅助工具包主要提供数学几何相关的计算支持。

在项目中如果是凸多边形的创建，可以使用 Flash CS 工具，先将可见图形放在一个图层中作为参考对象，然后使用钢笔等绘画工具绘制多边形。使用钢笔工具的好处是可以方便的调整各个顶点的位置。

第二个可以选择的方法是使用 JSFL(Flash 提供的角本语言) 对图形顶点进行扫描并记录 X 与 Y 坐标，并输出相应格式的数据。这时我们可以使用凸多边形的顶点数据来绘制物理对象，但有个问题是使用 JSFL 输出的顶点数组顺序是乱的，这就带来了一个问题：直接使用 Box2d 原生方法不能正常绘制图形。

当然开发人员可以手动的来记录这些凸多边形的顶点数据，从而形成相应顺序的顶点数据，但这将是一项非常耗时的工作，基于此，多边形绘制辅助工具包还增加了 PointSorter 类来处理顶点排序的问题，这将大大提供工作效率。图 5-4 为 Model 数据类图。

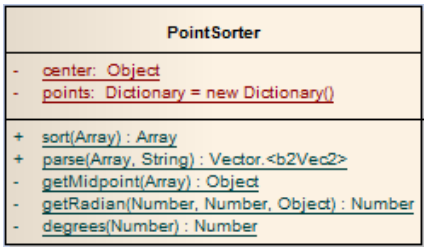

图 5-4 Model 数据

Fig. 5-4 Modal data

那么对于凹多边形来说，因为凹多边形的顶点排序有着不唯一性，因此就不能使用上面的方法来处理，经过多方搜寻，终于得到了一个变相的解决方案，原理如下：将凹多边形切割成多个三角形或四边形，然后组合成凹多边形。

创建顶点还是跟上面一样使用 Flash CS 工具，首先导入参考图，然后实现鼠标移动记录顶点功能，进行多边形顶点采集。这个过程是根据相对时间来读到鼠标所在位子的坐标信息的。最后结束绘制时按先后顺序保存顶点数据到一个数组，以便程序使用。

从开源社区获得一个工具类 Separator，采用这个工具类可以帮助快速处理这个问题。图 5-5 为工具类 Separator 的类图。

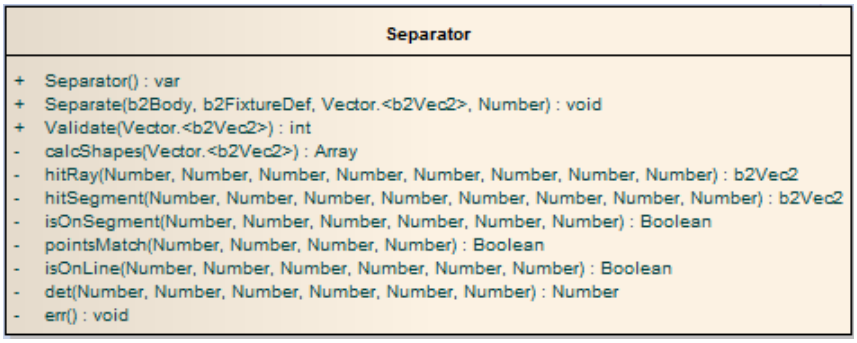

图 5-5 Separator 工具类

Fig. 5-5 Separator utility

5.1.6 event 包

event 包目前只是对碰撞事件的支持，以后将根据实际情况来调整

5.1.7 utils 包

utils 包提供了简化数据转换的一些工具类。图 5-6 为 BoxUtil 类图。

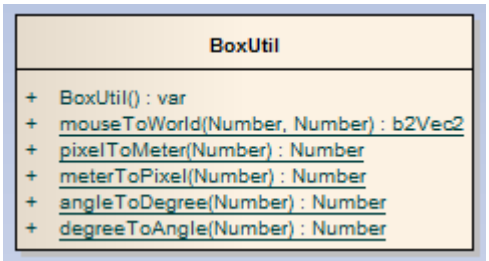

图 5-6 BoxUtil 类

Fig. 5-6 BoxUtil Class

5.2 抛投类游戏主要功能的实现

5.2.1 抛投功能的实现

在抛投功能的设计章节，我们分析了几种可能的抛投公式，下面将以 Flash Box2D 为基础，详细描述一下这一功能的实现步骤。

（1） 确定抛投的角度。角度的计算主要是通过确定抛投物与原点之间在 X 轴方向上的距离与在 Y 轴方向上的距离，然后根据几何三角公式 Math.atan2()来计算出抛投的角度。

（2） 确定抛投的力度。这可以通过测量拖拽抛投物离开原点的距离来确定，离原点越远，力度越大。

（3） 转换角度与力度为水平和垂直向量 b2Vec2(velocityX,velocityY)。

（4） 应用 Flash Box2D 模拟抛投 SetLinearVelocity(b2Vec2);

如果需要实现特殊的抛物曲线或者飞行模式，则可以在抛投过程中适当施加不同方向上的作用力来实现抛投轨迹的多样化。

5.2.2 碰撞功能的实现

Flash Box2D 是一个非常强大的 2D 物理引擎，可以实现精确的碰撞检测。下面来看看具体的实现方法。

1、用 world.GetContactList()方法获取碰撞对象。world.GetContactList()会返回一个 b2Contact 对象。b2Contact 用来管理碰撞的 shape，任何有超过两个及以上接触点的刚体，

Box2D 都认为发生了碰撞，并用 b2Contact 来管理。通过 b2Contact 的 GetFixtureA()和 GetFixtureB()方法，可以获取碰撞对象的 b2Fixture 属性引用，进而获取碰撞对象。

2、用 b2ContactListener 获取碰撞对象。b2ContactListener 是 Box2D 引擎用于碰撞检测的侦听器，只要刚体之间发生了碰撞，Box2D 就会自动调用 b2ContactListener 的相应方法，而程序员则可以根据实际情况进行相应的处理。能够被应用的方法主要有 BeginContact()和 EndContact()，前者是当碰撞发生时触发的方法，而后者则是当碰撞结束时才会触发的方法，主要是对碰撞物体的存留进行处理。

在 Flash 中编写碰撞检测代码时，通常会用 A 对象(如游戏主角)的 hitTest()或 hitTestObject()方法检查它与另外一个对象的碰撞，当碰撞发生时，分别对 A 和 B 进行碰撞处理。在这里可以很清楚哪个是 A 对象(游戏主角)，那个是 B 对象(敌人)。

但是在 b2Contact 或 b2ContactListener 中，获取的 bodyA 和 bodyB 无法知道哪个是游戏主角，哪个是敌人，比如想找到碰撞的对象是否是游戏主角，那么就得分别确认一下 bodyA 和 bodyB 了，游戏碰撞对象种类越多，判断越复杂。

需要检测大量碰撞的时候，所有需要在碰撞时触发的事件都要写到那一个继承自 b2ContactListener 的类中未免显得杂乱，不易管理，如果能把刚体的碰撞处理分开写到各自的 UserData 类中看起来就好多了。

5.2.3 破碎功能的实现

破碎功能主要就是对破碎后形成的多个多边形顶点的排序。因为 Flash Box2D 在创建多边形时只支持顺时针排序的顶点序列，所以要对新的刚体的顶点做顺时针排序。具体算法如下：

（1）根据每个顶点的 x 坐标，按升序排列的所有给定的点。

（2）找出最左边和最右边的点（简称为 L 和 R），并创建 tempVec，将 LR 点按顺时针顺序保存到 Vec 中。

（3）遍历其它顶点，并使用 determinant 函数，把 LR 线以上的顶点按顺序序列插入 LR 点之间，并将位于 LR 线以下的点按逆序排列。determinant 函数作用就是用来判断目标顶点(简称为 O 点)是在 LR 线的上向，下方还是线上。在计算之前确定了 3 个点 L、R、O，根据计算 3×3 矩阵的行列式，可以判定，如果返回一个正数，那么三个点是顺时针顺序，如果是负数，那么他们是逆时针顺序，如果为零，那么他们在同一行。下面的公式就是 3×3 矩阵行列式的计算方法

$$\begin{vmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{vmatrix} = a_{1,1}\ a_{2,2}\ a_{3,3} + a_{1,2}\ a_{2,3}\ a_{3,1} + a_{1,3}\ a_{2,1}\ a_{3,2} - a_{1,3}\ a_{2,2}\ a_{3,1} - a_{1,1}\ a_{2,3}\ a_{3,2} - a_{1,2}\ a_{2,1}\ a_{3,3}$$

（5-1）

在破碎的之后，还要剔除面积较小的多边形，以提高游戏运行效率。而前提是必须

先计算出多边形面积的大小，这也有是应用 3×3 矩阵的行列式来帮助计算的。
因为多边形顶点只有 X，Y 属性，因此行列式也就变形成以下公式

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1y_2 + x_2y_3 + x_3y_1 - x_2y_1 - x_3y_2 - x_1y_3 \quad (5\text{-}2)$$

而三角形的面积公式为

$$S=1/2\ (x_1y_2 + x_2y_3 + x_3y_1 - x_2y_1 - x_3y_2 - x_1y_3) \quad (5\text{-}3)$$

计算多边形面积的方法是将多边形分割成多个三角形，然后分别计算三角形的面积，最后汇总成多边形的面积。

5.3 游戏编辑器的逻辑实现

5.3.1 搭建场景模块

搭建场景模块的实现主要是通过预先设计好的构建部件，通过选取相应部件，然后以拖拽的方式在场景是进行搭建。具体算法如下：

（1） 根据预置部件的形状，编制好相应的 Box2D 的刚体数据。
（2） 通过 XML 配置文件将预置部件与 Box2D 的刚体数据相关联。
（3） 在编辑场景里通过拖拽的方式进行场景的搭建。
（4） 遍历场景中所有的对象，获取它们的位置、长宽、旋转角度等信息。
（5） 依据 Flash Box2D 刚体的创建规则完成场景的创建。

5.3.2 绘制场景模块

如果只需要绘制 Flash Box2D 原生形状，比如方形，圆形，凸多边形等，那么就可能应用 Flash 原生的绘图 API Graphic-draw() 来绘制显示对象，然后通过遍历所有对象，获取它们的位置、长宽、旋转角度等信息，依据简单多边形的创建规则就可以完成场景的绘制。下面主要描述一下多边形的绘制。

（一）凸多边形的绘制

凸多边形虽然是 Flash Box2D 原生支持的形状，但其创建方式也比较为复杂。创建多边形刚体通常有两种方法，一种是组合法，另一种是顶点构造法。下面我来详细看看这两种方法的具体实现过程。

（1）组合法

顾名思义，组合法就是用多个基本刚体（矩形刚体和圆形刚体）组合成的一个目标刚体，其实确切的讲，是用多个图形 shape，组成一个整体的图形；而刚体只有一个。图 5-7 为边形刚体组合法的示意图。

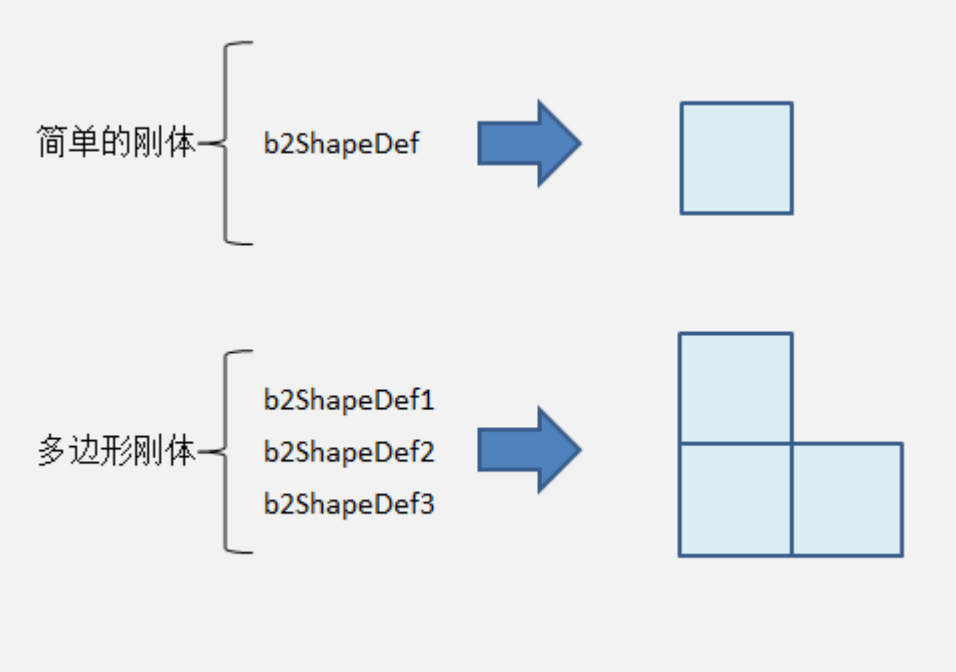

图 5-7 多边形刚体组合法

Fig. 5-7 Polygon rigid combination method

从上图看，多边形刚体的创建过程就简单多了，首先创建多个 b2ShapeDef 需求，然后根据这些需求创建一个复杂的刚体。

（2）顶点构造法

与组合法不同，顶点构造法是通过创建一个 b2ShapeDef 需求，然后在这个 b2ShapeDef 中详细描述每个顶点的坐标。需求的原则是先指定多边形的顶点个数，然后把每个顶点的坐标描述清楚。在 Box2D 中这些顶点都保存在 b2ShapeDef.vertices 数组中。在设置的时候也要遍历 shapeRequest.vertices 数组里的每个顶点，然后调用 Set 方法设置顶点的坐标。

（二）凹多边形的绘制

除了凸多边形的创建，在游戏开发中不可避免还会碰到凹多边形的绘制，凹多边形仅仅依靠 Flash Box2D 的 API 就不够了。在设计阶段考虑到通过对凹多边形进行切割，从而形成一系列小的凸多边形，再通过组合的方式来完成凹多边形的绘制。

对于凹多边形的切割，引用了 Antoan Angelov 发布的一个工具类 b2Separator. b2Separator 类可以根据 b2Shape 顶点，把它分成一个个凸多边形，然后用组合法构成复杂的多边形刚体。b2Separator 主要有下面两个函数：

（1）Separator 方法用来将 verticeVec 中的顶点分成多个小的凸多边形，并赋值给 fixtrueDef，创建 body 多边形刚体。每个参数说明如下：

1、body：多边形刚体对象

2、fixtureDef：fixture 需求对象

3、verticeVec：存储多边形所顶点的 Vector 数组

4、scale：Box2D 模拟时的缩放比例，这个值与 b2DebugDraw 中的 SetDrawScale 的参数值是一样的。

（2）Validate 的功能是检测多边形的刚体是否符合创建的标准，在不符合标准时，返回不符合标准的原因。参数说明如下：

verticeVec：存储多边形所顶点的 Vector 数组

返回值类型有下面几种：

0：顶点符合绘制多边形的标准

1：顶点连接的线段之间有交叉

2：顶点的顺序非顺时针绘制

3：出现 1 和 2 两种错误

值得一提的是，因为非顺时针顺序的顶点是无法创建多边形的，所以在 validate 检测顶点返回 2 之后，调用 verticesList.reverse()方法，反转顶点的顺序。只有返回值为 0 时，才可以用 Separator 方法创建多边形。

具体算法如下：

（1） 初始化一个 Point Vector 容器，存放绘制点。

（2） 在绘制屏幕上轻点以确立绘制的起始点，并将起始点放入 Point Vector 中。

（3） 移动绘制点到任意位置，继续轻点，记录下点的位置，继续存放入 Point Vector 中。

（4） 重复步骤 2-3 以完成凹多边形顶点的绘制。

（5） 把凹多边形的顶点传入 B2Separator，完成凹多边形的分割，并返回组合好的凹多边形。

5.3.3 特殊图形的扫描

特殊图形扫描法主要是通过导入一张透明背景的位图，应用最小二乘算法跟踪的图像的轮廓，从而获得所需要的物体的多边形顶点。具体的思路如下：

（1）找到一个放置在图像边缘的像素。

（2）考虑 2×2 像素的正方形，其中包括当前像素一般位于在左上角或右下角的平方。

（3）此时有 4 个像素，和他们相邻的可以是透明或不透明。因此，将有 16 个可能的 2×2 的正方形全透明像素或所有不透明的像素的情况下，虽然不会发生，因为正在走动的图像的边缘。

（4）根据在 2×2 平方不透明像素的数量和位置，可以猜测轮廓的方向，当前像素在这样的方向移动，并继续从第 2 步，直到你再次达到在步骤 1 中发现的像素。

通过上面的步骤，找出了图形的边界顶点，然后应用在上面已经使用过的

B2Separator 的方法，来创建特殊图形的刚体。

5.4 本章小结

1、预置模块搭建法前期的工作比较大，但搭建比较快速，而且比较具象。

2、通过组合法来创建的刚体中包含多个图形，所以在 Box2D 模拟碰撞检测时，就会对这些图形逐个进行检测计算，效率较低。

3、通过顶点构造法来创建的刚体只有一个图形，所以也只需要一次碰撞检测计算，计算效率相对要高一些。

4、通过指定顶点的坐标，可以创建出任意的形状，而不是简单的矩形和圆形刚体的组合。

5、通过扫描位图，获取顶点坐标，创建任意形态刚体的效率较高，但是通常只适合单一物体，而且要没有背景干扰，有一定的局限性。

6 基于 EasyBox2D 框架的物理游戏开发实例

6.1 游戏架构概述

为了验证 Flash 抛投类手机游戏设计框架的实际应用价值，我们开发的是一款基于 EasyBox2D 的跨平台（IOS/Android）的手机游戏，命名为“SiegeDuel(攻城对决)”。“攻城对决”主要讲述的是游戏体验者对自我城堡的构建，以及对他人城堡的进攻，以此来积累自身的荣誉，实现游戏世界中的价值。

游戏体验者通过游戏编辑器进行自我城堡的构建，构建完成后可以通过模拟攻城来对城堡的坚固程序进行测试，最后发布自己的城堡地图。

游戏体验者可以在城堡列表中选择自己想要攻打的城保，在有限的攻力能力范围内对敌对城堡进行攻打，试图摧毁对方的城堡，以此获得不同的荣誉。

游戏主要由三个模块组成：用户注册，建造城堡，攻打城堡。

图 6-1 为游戏总体架构图。

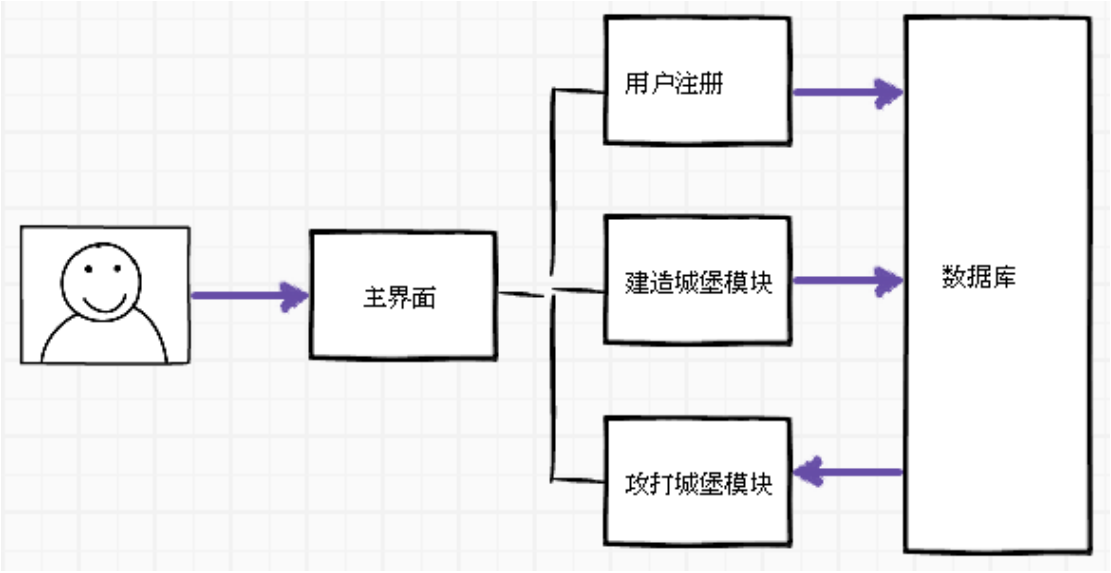

图 6-1 《攻城对决》总体架构

Fig. 6-1 "Siege Duel" overall architecture

6.1.1 用户注册模块

用户注册主要是对用户数据进行生成与存储的模块，用于辨识不同的用户。这是一个游戏通常都有的模块。

对新用户的引导，不应在用户注册之后便结束。将新用户丢弃在一个陌生的页面，

并且没有任何的提示，会让他们觉得不知所措。相反的，比较合适的做法是用一个欢迎信息和隐晦或明确的引导来告知用户他们接下来能做的事情。

SiegeDuel 在通过用户注册后，会显示友好的欢迎信息，并且提供二个可选择的选项：建造城堡和攻打城堡。在选项的设计上不需太多，并且会描述一下每一个选项的作用，而不只是给出链接。当然描述信息可以以悬停提示等方法给出。

6.1.2 城堡建造模块

建造城堡模块是一个类似编辑器的游戏界面，这个界面根据游戏体验者的不同等级，会提供二种建造模式：绘制模式和搭建模式。

绘制模式是游戏体验者通过选择绘制形状，通过拉拽，平移，旋转等动作来搭建游戏城堡的各个功能模块。

搭建模式是城堡构建材料是预先设计好的，用户可以在选择列表中选取相应的材料，通过拖拽的方法来完成城堡的搭建。

不论是绘制模式还是搭建模式，材料的数据都是从数据库中读取的，具体提供那些城堡构建材料，则可以根据用户的不同等级加以适当的配置，这也是游戏可玩性的一方面体现。

6.1.3 攻打城堡模块

攻打城堡模块是基于 EasyBox2D 来实现的。主要思想是通过投射装置，进行物理刚体的抛投，当物理刚体与城堡刚体碰撞时，进行相应的碰撞检测，以完成相应破碎策略与物体生命值的设定，最后通过记分或者攻打轮数来记录用户的得分高低。

6.2 游戏中物理城堡的实现

6.2.1 物理城堡概述

物理城堡是用不同矩形块组成的，矩形块分类为石块，木块，屋顶以及城堡主。不同的材质决定了不同的物理属性，比如木块最为脆弱，石块则是最坚硬的。游戏要搭建的城堡是为了保护城堡主的安全，城堡主的灭亡则视为城堡攻破，因此用户要通过不同的策略，对有限的城堡资源进行合理的搭建，从而实现最坚固的游戏城堡。

6.2.2 物理城堡的构建

针对不同的建筑块，会有不同的物理属性，为了便于测试以及扩展，可以将这些属性保存在相应的配置文件中。图 6-2 为配置文件的结构图。

```xml
<?xml version="1.0" encoding="UTF-8"?>

<Boxdata>
    <box type="ground" density="80" friction="100" restitution="0" hp="0" damageOthers="1"/>
    <box type="hero" density="20" friction="50" restitution="0.1" hp="100" damageOthers="1"/>
    <box type="stone" density="75" friction="90" restitution="0" hp="25" damageOthers="2"/>
    <box type="roof" density="60" friction="90" restitution="0" hp="20" damageOthers="4"/>
    <box type="pillar" density="20" friction="20" restitution="0" hp="15" damageOthers="1"/>
    <box type="king" density="50" friction="50" restitution="0.1" hp="30" damageOthers="1"/>
</Boxdata>
```

图 6-2 配置文件数据

Fig. 6-2 Configuration file data

在进入攻城模块之后，会读取相应的存储于数据库中的城堡信息，然后根据每一个城堡建筑块的类型，去匹配相关的物理属性，最后根据建筑块的类型，调用框架中相应的工厂（Factory）去创建具有物理属性的建筑块。图 6-3 为城堡示意图。

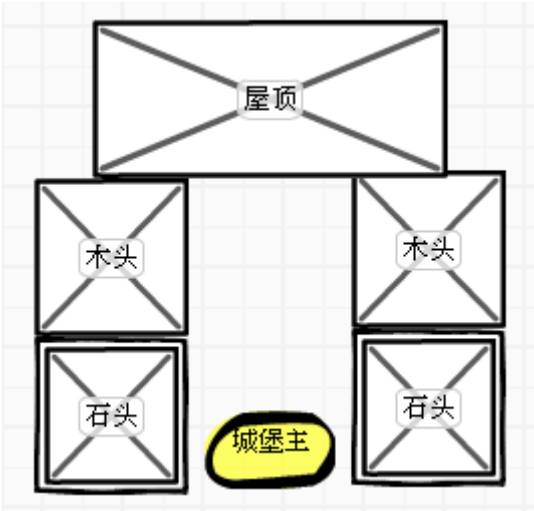

图 6-3 城堡

Fig. 6-3 Castle

6.3 游戏中投弹攻打的实现

6.3.1 投弹攻打概述

投弹机制类似愤努的小鸟，投射物可以围绕圆中心内旋转，最后释放时的角度将决定投射的角度，而距离中心点的远近则决定投射的力度，在拖拽投射物的过程中，将实理绘制投射预测曲线。图 6-4 为投射角度与力度的相关性示意图。

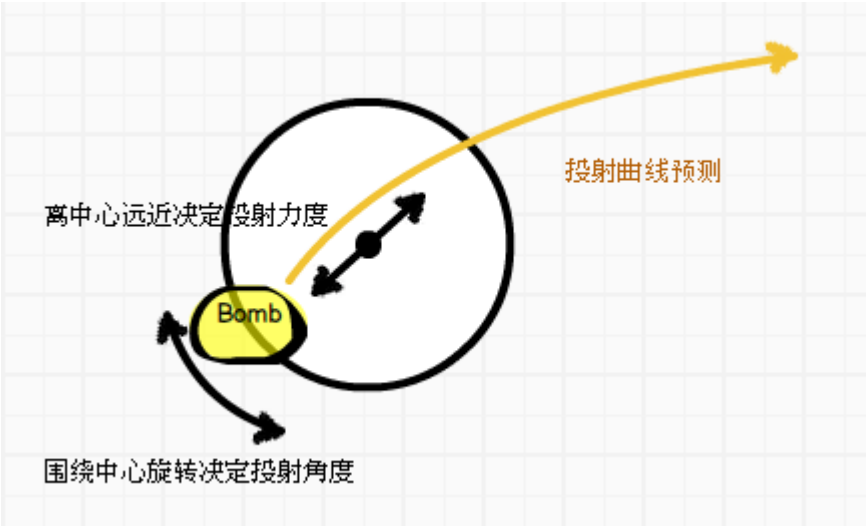

图 6-4 投射

Fig. 6-4 Casting

6.3.2 相应公式介绍

角度公式，主要应用三角函数，通过对抛投物偏离原点的距离来计算的

```
_distanceX=_heroClip.x-_originPoint.x;
_distanceY=_heroClip.y-_originPoint.y;
_heroAngle=Math.atan2(_distanceY,_distanceX);
```

速度公式，这也是应用抛投物偏离原点的程度作为参数来确定的，通过分别计算水平方向和垂直方向上的速度，然后形成速度向量。

```
var velocityX:Number=-_distanceX/_physicsData._velocityFactor;
var velocityY:Number=-_distanceY/_physicsData._velocityFactor;
_arrowVelocity = new b2Vec2(velocityX,velocityY);
```

投射功能主要是应用 Box2D 物理引擎，对刚体设置线性速度来实现，伪代码如下

```
_hero.SetLinearVelocity(_arrowVelocity);
```

6.4 本章小节

本章应用 EasyBox2D 的游戏框架，实现了城堡的搭建。

通过对投射机制的分析，计算出相应的作用力，并通过 Box2D 物理引擎将相应的作用力加到投射物上，使之产生相应的动能，实现抛物投射的效果。

当投射物与城堡碰撞时，进行相应的碰撞检测，并实行不同的生命值耗损策略来控制城堡的损坏。

7 总结与展望

7.1 总结

随着手机游戏软硬件技术的不断发展，以及手机游戏用户数据的急剧增长，手机游戏市场规模也得到了空前的扩展。在这个不断发展的市场中，通过数据挖掘，分析归纳，我们找到了手机抛投类游戏这一市场热点。通过对 Flash 抛投类手机游戏需求的深入分析，以及抛投游戏技术基础的详细了解，我们设计了 EasyBox2D 游戏开发框架，并在相应的实例开发中得到的验证。

本次论文研究所展开的工作和所取得的成果总结如下：

（1）对 FLASH 开发手机游戏进行了概述。分析了 FLASH 抛投类手机游戏的物点；概述了 Flash 物理引擎现状，Flash Box2D 的应用优势，以及 Flash Box2D 在手机游戏开发中存在的问题和优化的方向。

（2）论述了 Flash Box2D 技术基础，详细介绍了 FLASH BOX2D 刚体创建基本步骤，并在此基础上对常用物理刚体的创建进行了深入分析。而对于游戏开发中比较常用的 Box2D 特殊效果，也进行了一定程度的揭示，为实际开发提供了思路的方向。

（3）针对 FLASH BOX2D 抛投类手机游戏开发的需求与难点，应用设计模式的思想，对 EASYBOX2D 游戏框架进行了设计，并应用模块开发的思路对抛投类游戏的主要功能进行了设计。为了便于游戏的测试，以及有户对游戏场景的参与感，对游戏编辑器进行了结构设计，充分考虑了场景数据的格式，以适应不同层级的场景数据收集模式，为游戏实际开发奠定了坚实的基础。

（4）对 EasyBox2D 游戏框架的设计进行了逻辑实现。通过 GameWorld 实现了对物理世界的初始化封装，简化了开发的流程，并重点对物理对象的创建进行了工厂模式的设计，极大的简化了物理对象的创建。

7.2 展望

本次研究虽然得到了本人所在软件公司的大力支持，但公司为了游戏开发的保密性，导致某些功能模块未能涉及。同时由于本人水平有限，也限制了在本课题研究的发挥。因此，本文存在着诸多待进一步完善和验证之处。主要包括：

（1）目前只是对 Body 的创建实现了封装，对相对简单的应用已经足够。而对于需要表达相关性的物理性方面，则需要引入关节的概念；

（2）对于复杂刚体的创建，目前通过图形扫描法已基本实现了顶点数据获取的自

动化，但与实际开发的结合还比较松散；

（3）本课题仅对 FLASH 抛投类手机游戏中的基本形刚体创建进行了验证，对抛投功能也只验证了设计中的部分功能，对于复杂刚体的创建以及多变的抛投策略还有待进一步的探究与实践。

针对上述问题，本课题研究对下一步工作有如下展望：

（1）对关节的创建还需要进行相应的封装，并进一步整合到游戏开发框架中；

（2）对于任意多边形的顶点的采集，图形扫描法已基本实现了自动化，在将来的开发中将作与开发框架进一步整合，希望能实现界面化操作，进一步提高开发效率；

（3）抛投功能还有很多需要完善、优化的地方，抛投策略的多样化也有待进一步探索验证。

因为能力时间有限，还有很多不足之处，希望批评指正，共同进步。

参考文献

[1] 李惠，丁革建. 智能手机操作系统概述[J]. 电脑与电信，2009 (03).

[2] 拓颖，沈浩. 浅析手机游戏的发展现状及未来趋势[J]. 甘肃科技，2013(05).

[3] 王彦恩，2013 年中国手机游戏用户调查研究报告[J]. 互联网消费调研中心，2013.

[4] 赵静静，韦凯，师华，解析 Flash 动画的技术特性[J]. 美术大观，2010(09).

[5] 闫丰亭，刘畅，贾金原，Flash3D 引擎的发展现状剖析及若干关键技术研究[J]. 系统仿真学报，2013(10).

[6] Thibault Imbert．Introducing Starling [M]．O'Reilly Media，2012.

[7] Adobe Developer Library ． Flash ActionScript 3.0 API Doc [EB/OL]．http://help.adobe.com/zh_CN/FlashPlatform/reference/actionscript/3/，2014.

[8] 邱彦林．AIR Android 应用开发实战[M]．机械工业出版社，2012(8).

[9] Richard Wagner．Professional Flash Mobile Development: Creating Android and iPhone Applications [M]．Wiley，2012.

[10] Veronique Brossie．Android 移动应用开发——基于 Adobe AIR [M]．O'Reilly Media，2012（7）.

[11] Wendy Stahler．游戏编程-数学和物理基础[M]．机械工业出版社，2008.

[12] Rosenzweig,G．ActionScript 3.0 游戏编程(第 2 版) [M]．人民邮电出版社，2012（1）.

[13] 卢胤，卜军义．手机游戏策划设计[M]．电子工业出版社，2010.

[14] Keith Peters．Flash ActionScript 3.0 动画高级教程[M]．人民邮电出版社，2010

[15] Emanuele Feronato．Box2D For Flash Game [M]．Packt Publishing Ltd., November 2012.

[16] Erin Catto．Flash Box2D API [EB/OL]．www.box2dflash.org/docs/, 2014.

[17] William Sanders, Chandima Cumaranatunge．ActionScript 3.0 Design Patterns [M]．O'Reilly Media，July 2007.

[18] Joey Lott, Danny Patterson ActionScript 3 设计模式[M]．清华大学出版社，2008.3.

[19] 杜恩宽.一种基于 Flash 的区块地图编辑器实现[J].计算机应用与软件，2008 (05) ．

[20] 高静，段会川．JSON 数据传输效率研究[J]．计算机工程与设计，2011 (07).

致　谢

经过整个毕业论文阶段的努力，终于完成了我的毕业论文，在此，我诚挚的感谢在我毕业论文写作过程中帮助过我的导师和同学。

首先，我要感谢我的论文指导老师，导师对我在论文写作和算法设计中遇到的问题进行了耐心细致的分析，并向我提出了指导性建议；导师在整个过程中的言传身教，使我积累了较为丰富的 FLASH 抛投类手机游戏设计与实验经验，而且在精神方面给了我许多鼓励，才使得我的设计能够如期完成。

其次，我要感谢在学习期间帮助过我的老师和同学们，是你们让我体会到一个团队是多么的重要，和大家在一起协作是多么的愉快，因为有了你们，我的毕业论文才这么顺利地完成，我的学习生活才如此的精彩。

最后，我还要感谢上海交通大学为我们提供了一个非常良好的硬件环境，使我可以动手实践。感谢软件工程学院的领导们，没有您们的辛勤教诲，也不会有我现在的成绩。

最后，再次向所有给过我帮助的导师和同学表示衷心的感谢！